Decoding Error-Correcting Codes via Linear Programming

by

Jon Feldman

S.M. Computer Science, Massachusetts Institute of Technology, 2000

A.B. Computer Science, Dartmouth College, 1997

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Massachusetts Institute of Technology

September 2003

Author ....
Department of Electrical Engineering and Computer Science
June 27, 2003

Certified by ....
David R. Karger
Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Decoding Error-Correcting Codes via Linear Programming

by

Jon Feldman

Submitted to the Department of Electrical Engineering and Computer Science
on June 27, 2003, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract. Error-correcting codes are fundamental tools used to transmit digital information over unreliable channels. Their study goes back to the work of Hamming [Ham50] and Shannon [Sha48], who used them as the basis for the field of information theory. The problem of decoding the original information up to the full error-correcting potential of the system is often very complex, especially for modern codes that approach the theoretical limits of the communication channel.

In this thesis we investigate the application of *linear programming* (LP) *relaxation* to the problem of decoding an error-correcting code. Linear programming relaxation is a standard technique in approximation algorithms and operations research, and is central to the study of efficient algorithms to find good (albeit suboptimal) solutions to very difficult optimization problems. Our new "LP decoders" have tight combinatorial characterizations of decoding success that can be used to analyze error-correcting performance. Furthermore, LP decoders have the desirable (and rare) property that whenever they output a result, it is guaranteed to be the optimal result: the most likely (ML) information sent over the channel. We refer to this property as the *ML certificate* property.

We provide specific LP decoders for two major families of codes: *turbo* codes and *low-density parity-check* (LDPC) codes. These codes have received a great deal of attention recently due to their unprecedented error-correcting performance. Our decoder is particularly attractive for analysis of these codes because the standard message-passing algorithms used for decoding are often difficult to analyze.

For turbo codes, we give a relaxation very close to min-cost flow, and show that the success of the decoder depends on the costs in a certain residual graph. For the case of rate-1/2 repeat-accumulate codes (a certain type of turbo code), we give an inverse polynomial upper bound on the probability of decoding failure. For LDPC codes (or any binary linear code), we give a relaxation based on the *factor graph* representation of the code. We introduce the concept of *fractional distance*, which is a function of the relaxation, and show that LP decoding always corrects a number of errors up to half the fractional distance. We show that the fractional distance is exponential in the girth of the factor graph. Furthermore, we give an efficient algorithm to compute this fractional distance.

We provide experiments showing that the performance of our decoders are comparable to the standard message-passing decoders. We also give new provably convergent message-passing decoders based on linear programming duality that have the ML certificate property.

Thesis Supervisor: David R. Karger
Title: Professor

Acknowledgments

From the first time I stepped in the door of LCS, until now, as I am putting the finishing touches on my thesis, David Karger has taught me every aspect of how to do research in the field of theoretical computer science. Not only is his knowledge and experience a valuable resource; his instinct and intuition is an inspiration to work with. I am forever indebted to him for the knowledge and experience (and funding...) I have gained while at MIT.

The results in this thesis are joint work with David and also Martin Wainwright, who has been an invaluable colleague during the past year. His knowledge in this area has been key to the success of the research, and I thank him very much for the time he has put into discussions, writing and research. I have also enjoyed many helpful conversations with David Forney on this subject, and I thank him very much for his time and attention. I also thank Ralf Koetter, Rudi Urbanke, Jonathan Yedidia, Muriel Medard, Piotr Indyk, Dan Spielman, Madhu Sudan, Vanu Bose, Matteo Frigo and Ibrahim Abou-Faycal for helpful conversations concerning this research.

Besides David, many other mentors and teachers have contributed to my growth as a researcher during my time at MIT. These include Piotr Indyk, David Williamson, Madhu Sudan, David Forney, Dan Spielman and Santosh Vempala (who is also an extremely valuable member of the theory volleyball team). A special thank you to David (F) and Madhu for serving on my thesis committee along with David (K). I'd also like to thank the professors I worked with as a teaching assistant that I have not mentioned already, including Michel Goemans, Seth Teller, Mike Sipser, Michael Kleber, Shafi Goldwasser and Bruce Maggs.

I spent the last two summers at Vanu, Inc., a company in Cambridge specializing in software radio. Not only did Vanu provide me with a wonderful place to work during the summer, but they also introduced me to the world of coding theory, the subject of this thesis, for which I am forever grateful. Thanks go out to Vanu Bose, Andy Beard and John Chapin for giving me this opportunity. I would also like to thank those researchers with whom I worked closely, including Matteo Frigo, Ibrahim Abou-Faycal, Andrew Russell and Desmund Lun. A special thanks goes out to Matteo, from whom I have learned a great deal, and whose persistence in the effort to do things the *right* way has been enlightening. Finally, a thanks to those who made the atmosphere at Vanu so enjoyable, including those I have already mentioned, in addition to Rick, Alok, Jon S, Stan, Victor (the best acronym-scrabble player I know), Steve, Andrew, Mary, Mike I, Jeremy, Ariel, Mike S, Jess, Elizabeth, and others to whom I apologize for leaving out.

My summer at Tel-Aviv University was also an enriching experience that I am grateful for having had. I thank Guy Even for his support and guidance during my time there, and afterward; Guy is also a pleasure to work with. I also thank Guy Kortarz, Zeev Nutov and Zvika Brakerski for working with me during my time in Israel.

I also thank Cliff Stein, Javed Aslam and Daniela Rus for teaching me computer science at Dartmouth, strongly suggesting that I attend graduate school, and then undoubtedly helping me to get into MIT. A special thanks to Cliff for encouraging me to write an undergraduate thesis, and sponsoring me for the upcoming two years at Columbia.

My fellow MIT theory students deserve as much of the credit as anyone else for my education. Every personality that has walked the hallways of the third floor at LCS has in some way contributed to my experience here. The incredible intelligence of the students here has been inspiring, and at times even intimidating, but has always been what I point to as the single most important reason that MIT is a fabulous environment in which to do graduate research.

Matthias Ruhl has been a great research partner during my time here, and I enjoy working with him very much; I have learned a great deal from Matthias about how to do work in this field, and also how to write good papers.

It is hard to determine whether I had a net gain or loss in research progress from having known Adam Klivans. However, Adam has without a doubt been a reliable (albeit opinionated) source of knowledge and guidance, an often times much-needed distraction, and a good friend.

I have also gotten to know Ryan O'Donnell very well during my MIT life. It has been especially good to have him as a friend and comrade through the sometimes hair-raising experience of the last year of grad school.

I thank the other students and postdocs who have contributed to the environment here, and with whom I have had many wonderful and enlightening discussions, classes, seminars, lunches, games of anagrams and scrabble, grading sessions, Christmas parties and volleyball matches, including Nati, Dunagan, Rudi, Maria, Adrian, Prahladh, Alantha, Matt L (both of them), Venkat, Salil, Amit, Anna, Leo, Sofya, Yevgeniy, Lars, Kalai, Eli, Rocco, Wizong, DLN, Peikert, Hanson, Steve, April, Brian D, Mike R, Carly, Abhi, Mohammad, Moses, Susan, Yoav, Mihai, Adam S, and others.

I would also like to thank Be, Chuck and Joanne for making all our lives easier. The importance of Be's everlasting M&M jar (as well her energy and enthusiasm) cannot be overstated. I also thank Greg and Matt for their talent in making things work.

Even though the probability that those responsible will actually read this is quite low, I need to mention a Cambridge institution that served as my second office throughout my studies: the 1369 Coffeehouse. Their coffee is the standard to which I hold all other coffee. Their bagels are actually pretty good too.

Finally, I would like to thank my friends and family, who have always been the most important part of my life, and have believed in me and supported me throughout. You know who you are, and I love you all.

Jon Feldman
Cambridge, Massachusetts
May, 2003

I acknowledge David Karger, the Laboratory of Computer Science at MIT and the Department of Electrical Engineering and Computer Science at MIT for the financial support of this research.

Contents

List of Figures

Chapter 1

Introduction

Error-correcting codes are the basic tools used to transmit digital information over an unreliable communication channel. The channel can take on a variety of different forms. For example, if you are sending voice information over a cellular phone, the channel is the air; this channel is unreliable because the transmission path varies, and there is interference from other users.

A common abstraction of channel noise works as follows. We first assume that the information is a block of k bits (a block of k `0`s and `1`s). When the sending party transmits the bits over the channel, some of the bits are flipped — some of the `0`s are turned into `1`s, and vice-versa. The receiving party must then try to recover the original information from this corrupt block of bits.

In order to counter the effect of the "noise" in the channel, we send *more* information. For example, we can use a *repetition code*: for each bit we want to send, we send it many times, say five times. This longer block of bits we send is referred to as the *codeword*. The receiving party then uses the following process to recover the original information: for every group of five bits received, if there are more `0`s than `1`s, assume a `0` was transmitted, otherwise assume a `1` is transmitted. Using this scheme, as long as no more than two out of every group of five bits are flipped, the original information is recovered.

The study of error-correcting codes began with the work of Hamming [Ham50] and Shannon [Sha48], and is central to the field of information theory. Volumes of research have been devoted to the subject, and many books have written [MS81, Bla83, Wic95, vL99]. We give more background in coding theory in Section 2.1.

This thesis is a study of a particular approach to the problem of *decoding* an error-correcting code under a probabilistic model of channel noise. (Decoding is the process used on the receiving end to recover the original information.) Our approach is based on *linear programming relaxation*, a technique often used in computer science and combinatorial optimization in the study of approximation algorithms for very difficult optimization problems. We introduce the "LP decoder," study its error-correcting performance, and compare it to other decoders used in practice.

Linear Programming Relaxation. Linear programming is the problem of solving a system of linear inequalities under a linear objective function [BT97]. While many useful optimization problems can be solved using linear programming, there are many that cannot; one problem is that the optimum solution to the system of inequalities may contain real values, whereas the variables may only be meaningful as integers. (For example, a variable could represent the number of seats to build on an airplane.) If we add to the LP the restriction that all variables must be integers, we obtain an *integer linear programming* (ILP) problem. Unfortunately, integer linear programming is NP-hard in general, whereas linear programs (LPs) can be solved efficiently.

A natural strategy for finding a good solution to an ILP is to remove the integer constraints, solve the resulting LP, and somehow massage the solution into one that is meaningful. For example, in many problems, if we simply round each value to an integer, we obtain a decent solution to the ILP problem we wish to solve. This technique is referred to as *linear programming relaxation*. Some of the most successful approximation algorithms to NP-hard optimization problems use this technique [Hoc95].

An LP Relaxation of the Maximum-Likelihood Decoding Problem. The problem of maximum-likelihood (ML) decoding is to find the codeword that maximizes the likelihood of what was received from the channel, given that that particular codeword was sent. This codeword is then translated back into the original information. For many common channel noise models, the likelihood of a particular codeword may be expressed as a linear cost function over variables representing code bits. In this thesis we give integer linear programming (ILP) formulations of the ML decoding problem for large families of codes, using this likelihood cost function as the ILP objective function.

We focus our attention on the modern families of *turbo codes* and *low-density parity-check* (LDPC) codes. We give generic ILPs that can be used for any code within these families. When we relax the ILP, we obtain a solvable LP. However, in the ILP, each code bit is constrained to be either 0 or 1, whereas in the LP, the code bits may take on values *between* 0 and 1.

Despite this difficulty, the relaxed LP still has some useful properties. In the ILP, the solution space is exactly the set of codewords that could have been sent over the channel. In the LP, the solution space extends to real values; however, it is still the case that every *integral* setting of the variables corresponds to a codeword, and that every codeword is a feasible solution. Given these properties, suppose the optimal solution to the LP sets every bit to an integer value (either 0 or 1). Then, this must be a codeword; furthermore, this solution has a better objective value than all other codewords, since they are all feasible LP solutions. Therefore, in this case the decoder has found the ML codeword.

The "LP decoder" proceeds as follows. Solve the LP relaxation. If the solution is integral, output it as the ML codeword. If the solution is fractional, output "error." Note that whenever the decoder outputs a codeword, it is guaranteed to be the ML codeword. We refer to this property as the *ML certificate* property. This procedure may surprise those that are familiar with LP relaxation, since usually the interesting case is when the solution is fractional. However, in the decoding application, we are not concerned with finding approximately good solutions, we want to find the *right* solution: the codeword that was transmitted. Therefore, we design our relaxation to increase the probability, over the noise in the channel, that the LP decoder succeeds (i.e., that the transmitted codeword is the optimal LP solution).

The relaxations we give in this thesis all have the property that when there is no noise in the channel, the transmitted codeword is indeed the LP solution. Additionally, we show that for many codes and channels, this remains true as long as not too much noise is introduced. More precisely, we give turbo codes with the property that as long as the noise in the channel is under a certain constant threshold, the probability that the LP decoder fails decreases as $1/n^{\alpha}$, where n is the length of the codeword and α is a constant. We also show that there exist LDPC codes where our LP decoder succeeds as long as no more than $n^{1-\epsilon}$ bits are flipped by the channel, where $0 < \epsilon < 1$ is a constant.

Turbo Codes and Low-Density Parity-Check Codes. The work in this thesis is focused on the application of the LP decoder to decoding the families of *turbo codes* [BGT93] and *low-density parity-check codes* [Gal62]. These two families have received a lot of attention in the past ten years due to their excellent performance. Although much is understood about these codes, their excellent performance has yet to be fully explained. One of the main difficulties in analyzing these codes lies in the "message-passing" decoders most often used for these families; the success conditions of these decoders are very difficult to characterize for many codes. Remarkable progress has been made, however, particularly in the case of LDPC codes where the length of the codewords is allowed to be arbitrarily large [RU01, LMSS98]. We give more detail on previous work in these areas within the appropriate chapters of the thesis.

We introduce two major types of LP relaxations for decoding, one for turbo codes, and one for LDPC codes. The turbo code LP is very similar to the LP for the *min-cost network flow* problem [AMO93]. In the min-cost network flow problem, we are given a directed network, and we must set "flow rates" to the edges in order to meet a resource demand at a destination node. The goal is to minimize a cost function while fulfilling the demand and obeying the capacities of the network. Turbo codes are made up of a set of simpler "trellis-based" codes, where the trellis is a directed graph modeling the code. We formulate the LP for a single trellis-based code as a min-cost flow problem, using the trellis as the directed flow network. We then extend this formulation to any turbo code by applying constraints between the LP variables used in each component code. For LDPC codes, our relaxation is based on the *factor graph* modeling the code. The factor graph is an undirected bipartite graph, with nodes for each code bit, and nodes representing local codeword constraints. Our LP has a variable for each code bit node, and a set of constraints for each check node, affecting the code bit variables for the nodes in its neighborhood.

One of the main advantages of the LP decoder is that its success conditions can be described exactly, based on combinatorial characteristics of the code. We define success conditions for both turbo codes and low-density parity-check codes, and use them to derive performance bounds for LP decoding. These bounds apply to codes of any length, whereas most other bounds that we are aware of for the conventional "message-passing" algorithms suffer from the limitation that the code length must be very large for the bound to apply (e.g., [RU01, LMSS98]).

Specifically, we define precisely the set of *pseudocodewords*, a superset of the codewords, with the property that the LP always finds the pseudocodeword that optimizes the likelihood objective function. This definition unifies many notions of "pseudocodewords" known for specific channels [DPR+02], codes [FKMT01] and decoders [Wib96] under a single definition. Our pseudocodewords extend beyond these cases, and are well defined for *any* turbo code or LDPC code, under any binary-input memoryless channel. (A channel is memoryless if the noise affects each bit independently.)

New Message-Passing Decoders. The message-passing decoders often used in turbo codes and LDPC codes show superb performance in practice. However, in many cases these algorithms have no guarantee of convergence; furthermore, even if the algorithm does converge, the point of convergence is not always well understood.

In this thesis we use the *dual* of our linear programs to show that if a message-passing decoder follows certain restrictions, then it gains the ML certificate property, since its solution corresponds to a dual optimum. Furthermore, we use a Lagrangian dual form of our LP to derive specific message update rules. Using the subgradient algorithm, we show that our new message-passing decoders converge to the LP optimum.

1.1 Thesis Outline

- In **Chapter 2** we give definitions and terminology necessary for any work in coding theory. We discuss how to evaluate the quality of an error-correcting code and decoder. We also establish some coding notation used throughout the thesis. We review the family of binary linear codes and low-density parity-check (LDPC) codes. We also derive a general linear cost function for use with LPs over any binary-input memoryless channel.

- In **Chapter 3** we give a basic tutorial on the LP relaxation technique, as well as some background on the min-cost flow problem.

- In **Chapter 4**, we introduce LP decoding as a generic method; we first describe the LP relaxation abstractly, as it applies to any binary code. We then describe how to turn this relaxation into a decoder, and how to analyze its performance. The *fractional distance* of an LP relaxation for decoding is defined in this chapter. Maximum-likelihood decoders correct a number of errors up to half the classical distance of the code; we show that LP decoders correct a number of errors up to half the fractional distance. We discuss the application of LP decoding to binary linear codes, and establish symmetry conditions on the relaxation. We show that if a relaxation meets these conditions, then some simplifying assumptions can be made in the analysis, and the fractional distance can be computed efficiently.

- We discuss general binary linear codes and low-density parity-check (LDPC) codes in **Chapter 5**. We define a linear programming relaxation using the *factor* *graph* representation of the code. We define the notion of a pseudocodeword, which is a generalization of a codeword; the LP decoder finds the minimum-cost pseudocodeword. We show that there exist codes with a polynomial lower bound on the fractional distance. In this chapter we also discuss the idea of using various techniques to tighten the LP relaxation.

- In **Chapter 6**, we define the family of turbo codes, and give the LP decoder for this family. We begin by defining *trellis-based* codes and *convolutional* codes, which form the basic building blocks for turbo codes; the LP for these codes is a simple min-cost flow LP on a directed graph. We then give a general LP relaxation for any turbo code. We describe in detail the special case of *Repeat-Accumulate* codes, a particularly simple (yet still powerful) family of turbo codes, and prove bounds on the performance of LP decoding on these codes.

- In **Chapter 7**, we compare the LP decoder with some standard message-passing decoders used in practice. We begin by showing that the success conditions for LP decoding are identical to standard message-passing decoders for several specific codes and channels. These results unify many known notions of a "pseudocodeword" [DPR+02, FKMT01, Wib96] under a single definition, and show that LP decoders and message-passing decoders have identical performance in many cases. This chapter also includes some experimental work comparing the performance of LP decoding with two standard message-passing decoders: the *min-sum* algorithm and the *sum-product* (belief propagation) algorithm.

- Finally, in **Chapter 8** we give new message-passing algorithms for both turbo codes and LDPC codes. We use the duals of our linear programs from previous chapters in order to derive message update rules. We use the subgradient algorithm to derive new message-passing decoders that provably converge to the LP optimum. We also show that if *any* message-passing decoder follows certain restrictions, then it gains the ML certificate property, since its solution corresponds to a dual LP optimum.

- We conclude our work in **Chapter 9**, and survey some of the major open questions in LP decoding of turbo codes and LPDC codes. We also suggest a number of general ideas for future research in this area.

1.2 Intended Audience

This thesis represents the application of a technique from theoretical computer science and operations research to a problem in communication and information theory. It is intended for any researcher within these broad areas. Consequently, we include a background chapter (2) on coding theory intended for computer scientists, as well as a background chapter (3) on linear programming intended for coding theorists. The reader should be aware that since the author is from a computer science background, the language is inevitably tilted toward the computer science audience.

Chapter 2

Coding Theory Background

In this chapter we review the basics of coding theory. We discuss the various components and parameters that go into the design of an error-correcting code, as well as the different measures of performance that are used to evaluate a code. We review the high-level abstraction of a binary-input memoryless communication channel, and show some common examples. We formulate the problem of maximum-likelihood (ML) decoding, and show that for any binary-input memoryless channel, ML decoding can be expressed as minimizing a linear cost function on the code bits

Those readers who are well-versed in the basics of coding theory may skip this section, although we do establish notation here that is used throughout the thesis, and we state some general assumptions.

2.1 Components of an Error-Correcting Code

The repetition code example we gave in the introduction illustrates every basic component of an error-correcting code. Below we list these components to familiarize the reader with the terminology used throughout this thesis, and in all the coding theory literature. Figure 2-1 illustrates the high-level model of an error-correcting code.

- The ***information word*** is a block of symbols that the sending party wishes to transmit. This information could be a sound, a picture of Mars, or some data on a disk. For our purposes, the information word $x \in \{0,1\}^k$ is simply an arbitrary binary word of k bits.

  There are some error-correcting codes built on information streams of arbitrary (infinite) length, but we will restrict ourselves to ***block codes*** in this thesis (codes defined for some finite length n). With a block code, in order to send more than k bits of information, multiple blocks are transmitted. It is assumed that each block is independent with respect to information content, and the effect of noise; therefore, we concentrate on sending a single block over the channel. We also note that some codes use non-binary alphabets; i.e., the symbols of the information word are not just bits, but are symbols from a larger alphabet. We restrict ourselves to ***binary codes*** in this thesis; those built on the alphabet $\{0,1\}$.

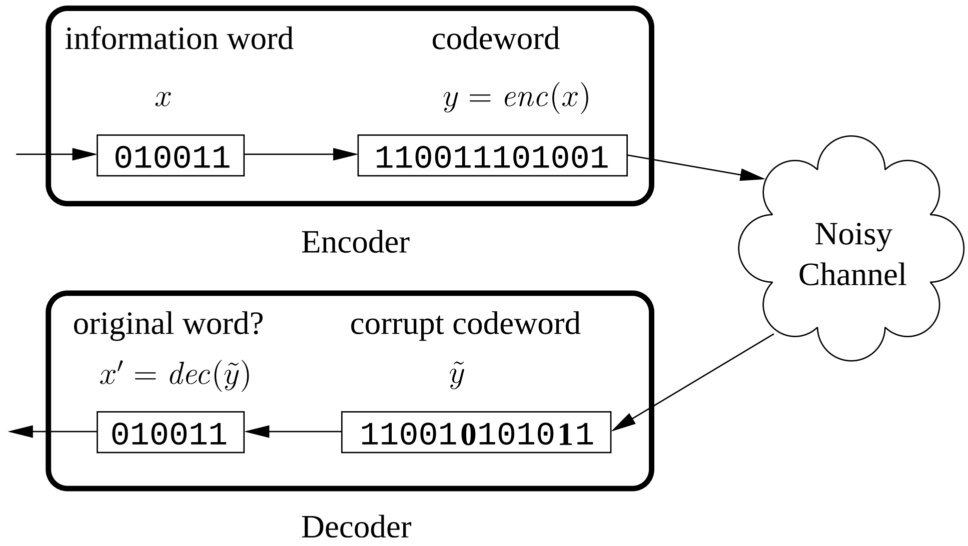


Figure 2-1: The high-level model of an error-correcting code. An information word x of length k is encoded into a longer codeword $y = enc(x)$ of length $n > k$. This codeword y is sent over a noisy channel, and a corrupt word $\tilde{y}$ is received at the other end. This corrupt word is then decoded to a word $x' = dec(\tilde{y})$, which hopefully is equal to the original information x.

- The ***encoder*** is the process the sending party uses to build redundancy into the information word. The encoder is simply a function $enc : \{0,1\}^k \rightarrow \{0,1\}^n$ that given an information word, outputs a longer binary word y of length $n > k$. In our repetition code example, the encoder is the function that repeats every bit five times, and so $n = 5k$. For example, if $x = 101$, then our encoder would output $y = enc(101) = 111110000011111$.

- The ***codeword*** is the binary word output by the encoder. This is the word that is transmitted over the channel.

- The ***code*** $\mathcal{C}$ is the set of codewords that could possibly be transmitted, i.e., they are the binary words that are encodings of some information word. Formally, $\mathcal{C} = \{y : y = enc(x), x \in \{0,1\}^k\}$. In our repetition code example, the code is exactly the set of binary words of length n where the first five bits are the same, and the next five bits are the same, etc.

- The ***(block) length*** of the code is equal to n, the length of the codewords output by the encoder. The parameter k, the length of the information word, is often referred to as the ***dimension*** of the code. This term is used for *linear codes*, when the code is a vector space (we detail linear codes in Section 2.7).

- The ***rate*** r of the code is the ratio between the dimension and the length. More precisely, $r = k/n$. The rate of our example code is $1/5$. It is desirable to have a high rate, since then information can be transmitted more efficiently. One of the main goals in the study of error-correcting codes is to transmit reliably using a code of high rate (close to 1).

- The ***channel*** is the model of the communication medium over which the transmission is sent. We have already described a channel that flips bits of the codeword arbitrarily. Another common model is called the ***binary symmetric channel*** **(BSC)**, where each bit is flipped independently with some fixed ***crossover probability*** p. There are many other models, some of which we will discuss in later parts of this chapter.

- The ***received word*** $\tilde{y}$ is the output of the channel, the corrupted form of the codeword y. Each symbol $\tilde{y}_i$ of the received word is drawn from some alphabet Σ. In the BSC, $\Sigma = \{0, 1\}$. In the ***additive white Gaussian noise*** **(AWGN)** channel (the details of which we review in Section 2.4.1), each symbol of the received word is a real number, so $\Sigma = \mathbb{R}$.

- The ***decoder*** is the algorithm that receiving party uses to recover the original information from the received word $\tilde{y}$. In the repetition code example, the decoder is the following algorithm: (1) examine every five bits of the codeword; (2) for each set of five bits, if there are more 0s than 1s, output a 0, otherwise output a 1. We can think of the decoder as a function $dec : \Sigma^n \rightarrow \{0, 1\}^k$, which takes as input the corrupt code word $\tilde{y}$ and outputs a decoded information word x'.

  Usually the difficulty of the decoding process is not in translating from codewords back into information words, but rather in finding a codeword that was likely sent, given the received word $\tilde{y}$. In most codes used in practice, and in all the codes of this thesis, given a codeword $y \in \mathcal{C}$, finding the information word x such that $enc(x) = y$ is straightforward. Thus, in the discussion we will simply talk about a decoder finding a codeword, rather than an information word.

2.2 Evaluating an Error-Correcting Code

What do we want out of an error-correcting code? This depends highly on the application. The various parameters listed above play against each other in complex ways. We measure the efficiency of the code using the rate. We need to understand the computational complexity of the both the encoder and decoder, since this can play a role in the feasibility of a system. The latency of transmission is affected by both the encoder and decoder complexity, as well as the code length.

Finally, we need a measure of the error-correcting power of the code. Listed below are three possible such measures.

- A *decoding success* is when the decoder recovers the original transmitted codeword. A *decoding failure* is when the decoder outputs the incorrect codeword (even if only one bit is incorrect). The ***word error rate*** (WER) is the probability, taken over the random noise in the channel, of decoding failure. To measure this quantity, we need to make an assumption about the distribution of information words. Since the model is a general one, and all words in $\{0,1\}^k$ are possible, we will assume throughout the thesis that they are all equally likely.

- Let $\Delta(y, y')$ denote the Hamming distance between binary vectors y and y'. The ***minimum distance*** d of a code is a the minimum Hamming distance between any pair of distinct code words in the code:

$$d = \min_{y,y' \in \mathcal{C},\, y' \neq y} \Delta(y, y')$$

  This is also referred to as simply the ***distance*** of the code. We would like the distance to be large, so that many bits need to be flipped by the channel before a codeword starts to "look" like a different code word. In our repetition code example, the distance of the code is 5.

The art and science of coding theory is to discover the best possible system within a given range of parameters. For example, suppose we were designing a scheme for storing bits on a hard drive. We know that when we read each bit, there is an independent 1/100 chance of an error. Furthermore, suppose we are required to have a system where for every block of 128 bits of information read from the drive, there is only a 10^{-6} (one in a million) chance of reading that block incorrectly. However, we would like to reduce the amount of extra space taken up on the hard disk by our code, and we are limited to a decoder whose running time is linear in the number of bits read (otherwise reading from your hard drive would be quite inefficient). In coding theory language, this problem is the following: Given a BSC with crossover probability 1/100, what is the highest rate code of dimension 128 with a decoder that runs in linear time and achieves a WER of 10^{-6}?

2.3 Decoding an Error-Correcting Code

The design of a decoding algorithm is perhaps the most difficult task in the design of an error-correcting code, especially those that approach the theoretical limits of error-correcting capability. Given a particular code $\mathcal{C}$, a natural question to ask is: what is the best possible decoder, if our goal is to minimize WER? The ***maximum-a-posteriori*** (MAP) codeword y^* is the one that was most likely transmitted, given the received vector $\tilde{y}$:

$$y^* = \arg\max_{y \in \mathcal{C}} \Pr_{\text{noise}} [\, y \text{ transmitted} \mid \tilde{y} \text{ received} \,]$$

Using Bayes' rule, and the assumption that all information words have equal probability, the MAP codeword is the same codeword y that maximizes the probability $\tilde{y}$ was received, given that y was sent:

$$y^* = \arg\max_{y \in \mathcal{C}} \Pr_{\text{noise}} [\, \tilde{y} \text{ received} \mid y \text{ transmitted} \,] \tag{2.1}$$

This is referred to as the ***maximum-likelihood*** (ML) codeword.

Consider the binary symmetric channel, with crossover probability $p < 1/2$. Given a particular transmitted codeword y, the probability that a word $\tilde{y}$ is received on the other end of the channel is a strictly decreasing function of the Hamming distance between $\tilde{y}$ and y. Thus, under the BSC, the ML codeword y^* is the codeword that is closest in Hamming distance to the received codeword $\tilde{y}$:

$$y^* = \arg\min_{y \in \mathcal{C}} \quad \Delta(y, \tilde{y}) \qquad \text{(Under the BSC)}$$

Note that the ML codeword is not necessarily the original transmitted codeword, so even the optimal decoding algorithm can suffer decoding failure.

An ***ML decoder*** is one that always finds the ML codeword. This is also often called ***optimal decoding***. For many codes, there is no known polynomial-time ML decoding algorithm. In fact, the problem is NP-hard in general, and remains NP-hard for many families of codes used in practice [Mac03]. Therefore in most cases, one would settle for a sub-optimal (but efficient) decoder. The goal then is to show that the WER of that decoder is still low.

A ***bounded-distance*** decoder is one that always succeeds when fewer than $\lceil d/2 \rceil$ errors occur in the BSC, where d is the distance of the code. It is simple to see that ML decoders are also bounded-distance decoders: If fewer than $\lceil d/2 \rceil$ errors occur, then the Hamming distance between the transmitted word y and the received word $\tilde{y}$ is less than $\lceil d/2 \rceil$. If there were some other codeword y' at distance less than $\lceil d/2 \rceil$ from $\tilde{y}$, then y and y' would have distance less than d from each other, a contradiction of the fact that the code has minimum distance d.

2.3.1 Distance vs. WER

A lot of work in the field of coding theory focuses on code constructions that maximize the distance d. The distance of a code is certainly an important, fundamental and mathematically beautiful property of a code, and finding codes with good distance has applications outside of digital communication. However, the minimum distance is a "worst-case" measure of decoding performance, especially for ML decoding.

If only slightly more than $d/2$ errors occur in the channel, it still may be extremely unlikely that decoding fails. To analyze this effect, more of the distance spectrum of the code must be considered, not just the minimum. Thus, if our goal is good performance, then considering the distance as the only measure of quality ignores other important attributes of the code and the decoder. The WER provides a more practical measure of quality, and applies to any noise model, not just the BSC.

2.4 Channel Model

In this thesis we assume a binary-input memoryless symmetric channel. The channel being "binary-input" means that the data is transmitted as discrete symbols from $\{\mathtt{0}, \mathtt{1}\}$. The channel being "memoryless" means that each symbol is affected by the noise in the channel independently. Finally, "symmetric" refers to the fact that the noise affects $\mathtt{0}$s and $\mathtt{1}$s in the same way.

Formally, let Σ be the space of possible received symbols $\tilde{y}_i$. For example, in the BSC, $\Sigma = \{0, 1\}$. In other channels, we may have $\Sigma = \mathbb{R}$. We use the notation $\Pr[\ \tilde{y} \mid y\ ]$ to mean the probability, over the random noise in the channel, that $\tilde{y}$ was received, given that y was sent. If Σ is a continuous space, then $\Pr[\ \tilde{y} \mid y\ ]$ denotes the probability density function at the point $\tilde{y}$.

Symmetry tells us that Σ can be partitioned into pairs $(\mathtt{a}, \mathtt{a}')$ such that

$$\Pr[\ \tilde{y}_i = \mathtt{a} \mid y_i = 0\ ] \quad = \quad \Pr[\ \tilde{y}_i = \mathtt{a}' \mid y_i = 1\ ], \text{ and} \tag{2.2}$$

$$\Pr[\ \tilde{y}_i = \mathtt{a} \mid y_i = 1\ ] \quad = \quad \Pr[\ \tilde{y}_i = \mathtt{a}' \mid y_i = 0\ ]. \tag{2.3}$$

Clearly, the BSC is symmetric (as its name would suggest). Note also that $\mathtt{a} = \mathtt{a}'$ is a possibility. For example, the **Binary Erasure Channel (BEC)** is where each bit is *erased* with some fixed probability p. In the BEC, we have $\Sigma = \{0, 1, \mathtt{x}\}$. If a $\mathtt{0}$ is transmitted, then a $\mathtt{0}$ is received with probability $1 - p$, and an $\mathtt{x}$ is received with probability p. If a $\mathtt{1}$ is transmitted, then a $\mathtt{1}$ is received with probability $1 - p$, and a $\mathtt{x}$ is received with probability p. Thus, in the definition of a symmetric channel, our partitioned pairs are $(0, 1)$ and $(\mathtt{x}, \mathtt{x})$.

2.4.1 AWGN Channel

Everybody believes in the exponential law of errors:
the experimenters, because they think it can be proved by mathematics;
and the mathematicians, because they believe it has been established by observation.
-Whittaker and Robinson

Another common channel is the Additive White Gaussian Noise (AWGN) channel. In this channel, a Gaussian is added to each bit, and a real number is received. In this case it is convenient to use $\{-1, +1\}$ to represent a bit, rather than $\{0, 1\}$. We use the common convention and map $\mathtt{0} \to +1$ and $\mathtt{1} \to -1$.

Formally, if y_i is transmitted where $y_i \in \{+1, -1\}$, then

$$\tilde{y}_i = y_i + z_i$$

where $z_i \in \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable with 0 mean and variance σ^2. The Gaussian added to each bit is independent, so this channel is memoryless. Note that in this case $\Sigma = \mathbb{R}$.

To evaluate the likelihood of a a received symbol $\tilde{y}_i$, we use the probability density function given by

$$\Pr[\ \tilde{y}_i \mid y_i\ ] = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(\tilde{y}_i - y_i)^2}{2\sigma^2}\right) \tag{2.4}$$

To show that this channel is symmetric, we partition $\Sigma = \mathbb{R}$ into the pairs $(\mathtt{a}, -\mathtt{a})$ for all non-negative $\mathtt{a} \in \mathbb{R}$. We must show the relationships in Equation (2.2) and (2.3).

Since σ and π are constant, we just need to show that

$$\begin{aligned}((\mathtt{a}) - (+1))^2 &= ((-\mathtt{a}) - (-1))^2, \text{ and} \\ ((\mathtt{a}) - (-1))^2 &= ((-\mathtt{a}) - (+1))^2,\end{aligned}$$

for all $\mathtt{a} \in \mathbb{R}$. This is clear.

2.5 A Linear Likelihood Cost Function

In this section we define a linear cost function to measure the likelihood of a received word $\tilde{y}$, given that a particular codeword y is sent. We need the function to be linear in the variables y, which will take on values between zero and one in our LP relaxations.

Given a particular received word $\tilde{y}$, we define the *log-likelihood ratio* γ_i of a code bit y_i to be:

$$\gamma_i = \ln\left(\frac{\Pr[\tilde{y}_i \mid y_i = \mathtt{0}]}{\Pr[\tilde{y}_i \mid y_i = \mathtt{1}]}\right) \tag{2.5}$$

The sign of the log-likelihood ratio γ_i determines whether transmitted bit y_i is more likely to be a $\mathtt{0}$ or a $\mathtt{1}$. If y_i is more likely to be a $\mathtt{1}$ then γ_i will be negative; if y_i is more likely to be a $\mathtt{0}$ then γ_i will be positive. We will usually refer to γ_i as the *cost* of code bit y_i, where γ_i represents the cost incurred by setting a particular bit y_i to $\mathtt{1}$. We refer to $\sum_i \gamma_i y_i$ as the cost of a particular codeword y.

Theorem 2.1 *For any binary-input memoryless channel, the codeword of minimum cost is the maximum-likelihood codeword.*

Proof: In the following, every probability is taken over the noise in the channel. The ML codeword y^* is defined (equation 2.1) as

$$y^* = \arg\max_{y \in \mathcal{C}} \quad \Pr[\,\tilde{y} \mid y\,].$$

By the fact that the channel is memoryless, we have

$$\begin{aligned} y^* &= \arg\max_{y \in \mathcal{C}} \quad \left(\prod_{i=1}^{n} \Pr[\,\tilde{y}_i \mid y_i\,]\right) \\ &= \arg\min_{y \in \mathcal{C}} \quad \left(-\ln \prod_{i=1}^{n} \Pr[\,\tilde{y}_i \mid y_i\,]\right) \\ &= \arg\min_{y \in \mathcal{C}} \quad \left(-\sum_{i=1}^{n} \ln \Pr[\,\tilde{y}_i \mid y_i\,]\right). \end{aligned} \tag{2.6}$$

If we add the constant

$$\sum_{i=1}^{n} \ln \Pr[\,\tilde{y}_i \mid \mathtt{0}\,]$$

to the right-hand side of equation (2.6), we obtain

$$\begin{aligned} y^* &= \arg\min_{y\in\mathcal{C}} \left(\sum_{i=1}^{n} \ln \Pr[\,\tilde{y}_i \mid \mathtt{0}\,] - \ln \Pr[\,\tilde{y}_i \mid y_i\,] \right) \\ &= \arg\min_{y\in\mathcal{C}} \left(\sum_{i\,:\,y_i=1} \ln \left(\frac{\Pr[\,\tilde{y}_i \mid \mathtt{0}\,]}{\Pr[\,\tilde{y}_i \mid \mathtt{1}\,]} \right) \right) \\ &= \arg\min_{y\in\mathcal{C}} \left(\sum_{i=1}^{n} \gamma_i y_i \right). \end{aligned}$$

■

Rescaling for Particular Channel Models. We will frequently exploit the fact that the cost vector γ can be uniformly rescaled without affecting the solution of the ML problem. For example, given a binary symmetric channel (BSC) with crossover probability p, we have $\gamma_i = \ln[(1-p)/p]$ if the received bit $\tilde{y}_i = 0$, and $\gamma_i = \ln[p/(1-p)]$ if $\tilde{y}_i = 1$. Rescaling by $-\ln[p/(1-p)]$ allows us to assume that $\gamma_i = -1$ if $\tilde{y}_i = 1$, and $\gamma_i = +1$ if $\tilde{y}_i = 0$.

This rescaling makes sense in the context of the BSC. We said before that the ML decoding problem in the BSC is equivalent to finding the codeword that minimizes the Hamming distance to the received word. The rescaled cost function γ described above provides a proof of this fact, since the cost of a codeword under this function is equal to the Hamming distance to the received word, normalized by the distance of the all-zeros vector to the received word. In other words, for a codeword y, we have

$$\sum_i \gamma_i y_i = \Delta(y, \tilde{y}) - \Delta(0^n, \tilde{y}),$$

which, along with Theorem 2.1 shows $y^* = \arg\min_{y\in\mathcal{C}} \Delta(y, \tilde{y})$, since $\Delta(0^n, \tilde{y})$ is constant.

The AWGN also simplifies when we rescale. Recall that in the AWGN, we map the bit $\mathtt{0} \to +1$, and the bit $\mathtt{1} \to -1$. So, we have

$$\begin{aligned} \gamma_i &= \ln \left(\frac{\Pr[y_i = +1 \mid \tilde{y}_i]}{\Pr[y_i = -1 \mid \tilde{y}_i]} \right) \\ &= \ln \left(\frac{\frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(\tilde{y}_i - (+1))^2}{2\sigma^2} \right)}{\frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(\tilde{y}_i - (-1))^2}{2\sigma^2} \right)} \right) \qquad \text{(using (2.4))} \\ &= \frac{(\tilde{y}_i + 1)^2 - (\tilde{y}_i - 1)^2}{2\sigma^2} \\ &= \frac{2\tilde{y}_i}{\sigma^2} \end{aligned}$$

Thus, if we rescale by the constant $\sigma^2/2$, we may assume $\gamma_i = \tilde{y}_i$.

2.6 List Decoding vs. ML Decoding

Recently, there has been a lot of attention on the method of ***list decoding*** [Gur01]. With list decoding (usually defined for channels where Σ is some finite alphabet), the decoder is allowed to output not just one codeword, but many. It is then up to a different external process to determine the proper codeword. The decoder is said to be successful if the transmitted codeword is in the output list. A parameter of a list decoder is the *error tolerance*. A list decoder with error tolerance e outputs all the codewords within Hamming distance e from the received word. It is assumed that the error tolerance e is set such that the output list size is always polynomial in the block length n. Clearly if e is set to $\lceil d/2 \rceil - 1$, then the list decoder is a minimum-distance decoder. Thus list decoding goes beyond minimum-distance decoding by allowing many choices for the candidate information word.

Strictly speaking, ML decoding and list decoding are incomparable. In the case of fewer than e errors in the channel, list decoding can give the ML codeword by outputting the most likely codeword in its output list. Thus in this case, list decoding is more powerful. However, if there are more than e errors in the channel, and the transmitted codeword is still the ML codeword, the ML decoder will output the transmitted codeword, whereas list decoding may not.

2.7 Binary Linear Codes

A binary code $\mathcal{C}$ of length n is a ***binary linear code*** if $\mathcal{C}$ is a *linear subspace* of the vector space $\{0,1\}^n$. Recall that a linear subspace of a vector space is any subspace closed under addition and scalar multiplication. So a binary code $\mathcal{C}$ is linear if

$$0^n \in \mathcal{C},$$

and for all pairs of codewords $y, y' \in \mathcal{C}$,

$$(y + y') \in \mathcal{C},$$

where

$$(y + y') = (y_1 + y'_1, y_2 + y'_2, \ldots, y_n + y'_n).$$

Note that addition is performed mod 2.

2.7.1 Generator Matrix

A linear subspace of a vector space can be specified by the *basis* of the subspace. For our purposes, the basis is linearly independent set of codewords $B = \{y^1, \ldots, y^k\}$, each of length n, such that every codeword in $\mathcal{C}$ can be expressed as the sum of a subset of codewords in that set. In other words,

$$\mathcal{C} = \left\{ \sum_{i \in B'} y^i : B' \subseteq B \right\}.$$

Note that defined in this way, there are exactly 2^k codewords, each of length n, which is exactly what we would expect. Thus the dimension of the subspace is equal to the dimension of the code (length of the information word).

The *generator matrix* $\mathcal{G}$ of a binary linear code is a $k \times n$ binary matrix whose rows are the basis vectors $\{y^1, \ldots, y^k\}$. The encoder for a binary code simply multiplies the information word by the generator matrix $\mathcal{G}$. In other words,

$$enc(x) = x\mathcal{G}$$

and we have

$$\mathcal{C} = \left\{ x\mathcal{G} : x \in \{0,1\}^k \right\}.$$

Thus for an arbitrary binary linear code, the encoder takes $O(nk)$ time to compute the codeword.

Example. As an example of a linear code, we will construct the (7,4) Hamming code. This code has $k = 4$, $n = 7$, and therefore the rate is equal to $r = 4/7$. We will define the code by the following basis B:

$$B = \{(1,1,1,0,0,0,0), (1,0,0,0,1,0,1), (0,0,1,0,0,1,1), (1,0,1,1,0,0,1)\}$$

From the basis, we form the generator matrix $\mathcal{G}$:

$$\mathcal{G} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Now suppose we would like to encode the information word $x = (1,0,1,1)$. Then we apply the encoder to obtain the codeword:

$$y = enc(x) = x\mathcal{G} = \begin{pmatrix} 1 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

2.7.2 Parity Check Matrix

Denote the inner product of binary vectors y and y' as $\langle y, y' \rangle = \sum_{i=1}^{n} y_i y'_i \mod 2$. Every linear subspace of dimension k has an orthogonal linear subspace $\mathcal{C}^\perp$ of dimension $n-k$ such that for all $y \in \mathcal{C}$ and $y' \in \mathcal{C}^\perp$, we have $\langle y, y' \rangle = 0$. This new subspace $\mathcal{C}^\perp$ can be though of as a new code, and is often called the *dual code* to $\mathcal{C}$. This dual code also has a basis, and so we can write down a generator matrix $\mathcal{H}$ for this code as well.

Since we have $\langle y, y' \rangle = 0$ for all $y \in \mathcal{C}$ and $y' \in \mathcal{C}^\perp$, it must be the case that for all $y \in \mathcal{C}$, we have

$$y^T \mathcal{H} = 0,$$

and so we have

$$\mathcal{G}^T \mathcal{H} = 0.$$

In fact the converse is also true; i.e., for all $y \in \{0,1\}^n$ such that $y^T \mathcal{H} = 0$, we have $y \in \mathcal{C}$. Thus $\mathcal{H}$ is another way to specify the original code $\mathcal{C}$. The matrix $\mathcal{H}$ is called the *parity check matrix* of the code $\mathcal{C}$, and has the property that a word y is in the code if and only if y is orthogonal to every row of $\mathcal{H}$. The matrix $\mathcal{H}$ is called the parity check matrix because every row induces a parity check on the codeword; i.e., the elements of the row that are equal to $\mathtt{1}$ define a subset of the code bits that must have even parity (sum to 0, mod 2).

Example. We return to our (7,4) Hamming code, with the following parity check matrix:

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

Now consider the codeword $y = (0,1,1,1,0,1,0) \in \mathcal{C}$. We have:

$$y^T \mathcal{H} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}^T \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = 0$$

2.7.3 Factor Graph Representation

A linear code $\mathcal{C}$ with parity check matrix $\mathcal{H}$ can represented by a *Tanner* or *factor* graph G, which is defined in the following way. Let $\mathcal{I} = \{1, \ldots, n\}$ and $\mathcal{J} = \{1, \ldots, m\}$ be indices for the columns (respectively rows) of the $n \times m$ parity check matrix of the code. With this notation, G is a bipartite graph with independent node sets $\mathcal{I}$ and $\mathcal{J}$.

We refer to the nodes in $\mathcal{I}$ as *variable nodes*, and the nodes in $\mathcal{J}$ as *check nodes*. All edges in G have one endpoint in $\mathcal{I}$ and the other in $\mathcal{J}$. For all $(i,j) \in \mathcal{I} \times \mathcal{J}$, the edge (i,j) is included in G if and only if $\mathcal{H}_{ij} = 1$. The neighborhood of a check node j, denoted by $N(j)$, is the set of variable nodes i that are incident to j in G. Similarly, we let $N(i)$ represent the set of check nodes incident to a particular variable node i in G. Generally, we will use i to denote variable nodes (code bits), and j to denote parity checks.

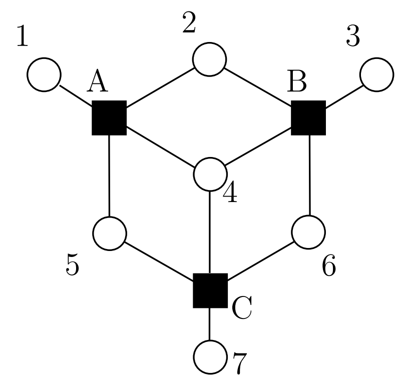


Figure 2-2: A factor graph for the (7,4) Hamming Code. The nodes $\{i\}$ drawn in open circles correspond to variable nodes, whereas nodes $\{j\}$ in black squares correspond to check nodes.

Imagine assigning to each variable node i a value in $\{0, 1\}$, representing the value of a particular code bit. A parity check node j is "satisfied" if the collection of bits assigned to the variable nodes in its neighborhood $N(j)$ have even parity; i.e., if $\sum_{i \in N(j)} y_i = 0 \mod 2$. The binary vector $y = (y_1, \ldots, y_n)$ forms a code word if and only if all check nodes are satisfied. Figure 2-2 shows the factor graph associated with the parity check matrix specified earlier for the (7,4) Hamming code. In this code, if we set $y_2 = y_3 = y_4 = y_6 = 1$, and $y_1 = y_5 = y_7 = 0$, then the neighborhood of every check node has even parity. Therefore, this represents a code word (the same code word as in our previous example), which we can write as `0111010`.

Note that the factor graph depends on the parity check matrix of the code. Since there are many possible parity check matrices for a given code, there are may possible equivalent factor graphs for the code. This becomes important when we consider decoding algorithms, many of which depend on structural properties of the factor graph.

2.7.4 Low-Density Parity-Check (LDPC) Codes

Let $\mathcal{C}$ be a linear code, let $\mathcal{H}$ be the parity check matrix for the code, and let G be a factor graph for the code. Let deg_ℓ^+ denote the maximum variable (left) degree of the factor graph; i.e., the maximum, among all nodes $i \in \mathcal{I}$, of the degree of i. Let deg_ℓ^- denote the minimum variable degree. Let deg_r^+ and deg_r^- denote the maximum and minimum check (right) degree of the factor graph.

We say that $\mathcal{C}$ is a *low-density parity-check* code if both deg_ℓ^+ and deg_r^+ are constant. In other words, a family of codes (parameterized by their length n) is considered to be a family of LDPC codes if deg_ℓ^+ and deg_r^+ stay constant as n grows.

More Information

For more information on error-correcting codes, we refer the reader to some standard textbooks. MacWilliams and Sloane [MS81] is perhaps the most common text, and has a mostly algebraic approach to coding theory. For a more combinatorial perspective, the textbooks of van Lint [vL99] and Blahut [Bla83] are good sources. For an engineering approach, the reader is referred to the textbook of Wicker [Wic95]; this is also a good source for convolutional codes and the Viterbi algorithm. A comprehensive reference is the *Handbook of Coding Theory* by Pless and Huffman [PH98]. For Turbo codes, there is the text by Vucetic and Yuan [VY00]. For LDPC codes, there is an excellent introductory chapter in the book by MacKay [Mac03].

Chapter 3

Linear Programming and Network Flow Background

The principal algorithmic technique used in this thesis is *linear programming* (LP) *relaxation*. This technique is central to the study of *approximation algorithms*; the method of finding valid solutions to NP-hard optimization problems whose cost is close to the optimum. In this chapter we give a brief introduction to the method of LP relaxation. We also review the network flow problem, which is central to the understanding our turbo code relaxation. We refer the reader to various textbooks for more background on linear programming [BT97, Sch87], approximation algorithms [Hoc95] and network flow [AMO93].

Those readers who are familiar with LP relaxation and network flow may skip this chapter.

3.1 Linear Programming Relaxation

As a running example of an optimization problem, we will use the VERTEX COVER problem: given an undirected graph $G = (V, E)$, and a cost γ_i for each node $i \in V$, find the lowest-cost subset $S \subseteq V$ of the nodes such that every edge in E has at least one endpoint in S.

3.1.1 Writing the Integer Linear Program

The first step in the LP relaxation technique is to formulate the problem as an *integer linear program* (ILP). An ILP consists of a set of variables, linear constraints defined on the variables, and a linear cost function on the variables. The variables are further constrained to be integers. Each constraint is an inequality on the LP variables, and must be linear. Equality constraints are also allowed, as they can be simulated by two inequalities.

For example, in the vertex cover problem, we have a variable v_i for each node $i \in V$. This variable indicates whether or not to include node i in the set S. Since this variable is an indicator, it must be either 0 or 1, so we enforce the linear constraint

$$\forall i \in V, \quad 0 \le v_i \le 1.$$

We also need to formulate a constraint that forces at least one of the nodes incident to each edge to be in S. The following linear constraint serves that function:

$$\forall (i,j) \in E, \quad v_i + v_j \ge 1 \tag{3.1}$$

Our cost function should minimize the cost of S, which can be expressed as $\sum_i \gamma_i v_i$. Overall, the ILP is the following:

$$\begin{aligned} \text{minimize} \quad & \sum_i \gamma_i v_i \quad \text{s.t.} \\ \forall i \in V, \quad & v_i \in \{0,1\}; \\ \forall (i,j) \in E, \quad & v_i + v_j \ge 1. \end{aligned} \tag{3.2}$$

Note that every feasible solution to the ILP above represents the incidence vector of a legal vertex cover S, where $S = \{i : v_i = 1\}$. Furthermore, every vertex cover S can be represented by an incidence vector v that is a feasible solution to these equations. Since the cost of a cover S with incidence vector v is exactly $\sum_i \gamma_i v_i$, solving the ILP in equation (3.2) solves the VERTEX COVER problem.

3.1.2 LP Relaxation

Unfortunately, integer linear programming is NP-hard. In fact, we have just proven this, since VERTEX COVER is NP-hard [GJ79]. So, the next step is to *relax* the integer linear program into something solvable. A *linear program* (LP) is the same as an ILP, but the variables are no longer constrained to be integers. Linear programs are solvable efficiently, which we discuss at the end of the section. In our example, when we relax the integer constraints, we now have the following linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_i \gamma_i v_i \quad \text{s.t.} \\ \forall i \in V, \quad & 0 \le v_i \le 1; \\ \forall (i,j) \in E, \quad & v_i + v_j \ge 1. \end{aligned} \tag{3.3}$$

Many LP relaxations are of this form, where binary variables are relaxed to take on arbitrary values between zero and one. Note that all the integral solutions to the ILP remain feasible for the above LP, but now the feasible set includes *fractional solutions*: settings of the variables to non-integral values.

A *polytope* is a set of points that satisfy a set of linear constraints. Every LP has an associated polytope: the points that satisfy all of the constraints of the LP. The *vertices* of a polytope are the points in the polytope that cannot be expressed as convex combinations of other points. These are the *extreme points* of the feasible solution space. The optimum point in an LP is always obtained at a vertex of the polytope [Sch87]. (Note that there may be multiple LP optima; however, at least one of these optima must be a vertex.)

3.1.3 Integrality Gap

For our minimization problem, the cost of the LP solution is certainly a lower bound on the true optimum, since all integral solutions remain feasible for the relaxed LP. However, the lower bound is not always tight; for example, suppose in our vertex cover problem we have as input a cycle on three vertices, each with a cost of 1. The optimal solution to the LP is to set every $v_i = 1/2$, for a total cost of 3/2. However, the minimum-cost vertex cover must contain at least two nodes, for a total cost of 2. Thus we see that for this relaxation, the true optimum can have a cost of 4/3 times the relaxed optimum.

This is an example of when the LP has its optimum at a *fractional vertex*: a vertex whose coordinates are not all integers. This example also illustrates the *integrality gap* of a relaxation, which is the ratio between the cost of the optimal integral solution and the cost of the optimal fractional solution. It turns out that in this case, for all graphs, the integrality gap is always at most 2. This is tight; i.e., there are graphs that exhibit an integrality gap of 2 (actually, $2(1 - 1/n)$ to be precise).

We conclude that LP relaxation can sometimes be used to obtain an *approximation* of the cost of an optimal solution. In the case of vertex cover, this approximation is guaranteed to be at least half of the true optimum.

3.1.4 Rounding to Obtain an Integral Solution

Often the solution to the LP relaxation can be used to obtain (efficiently) an integral solution to the original problem whose cost is within the integrality gap. In the example of vertex cover, this is the case. The algorithm is the following:

- Solve the linear program in equation 3.3 to obtain the optimal vertex v. Note that for all i, we have $0 \leq v_i \leq 1$.

- For all vertices $i \in V$, if $v_i \geq 1/2$, include i in S. Otherwise, do not.

The LP constraints (3.1) guarantee that for each edge, at least one of its incident nodes i has value $v_i \geq 1/2$. Therefore the solution S is a valid vertex cover, since for each edge, at least one of the incident nodes will be included in the vertex cover. Now consider the cost of S, compared to the cost of the LP optimum v; each node i in S contributes γ_i to the cost of S. However, if $i \in S$, we must have $v_i \geq 1/2$, and so v_i contributes at least $1/2\ \gamma_i$ to the cost of v. Therefore the overall cost $\sum_{i \in S} \gamma_i$ of S is at most twice the cost $\sum_i \gamma_i v_i$ of the LP optimum v. Since the cost of the LP optimum is a lower bound on the true optimum, the solution S has cost at most twice that of the true optimum.

3.1.5 Polytopes and LP Solving

The size of an LP problem instance is proportional to the number of variables and constraints in the associated polytope. Linear programs are solvable efficiently using a variety of techniques. The most commonly used algorithm in practice is the simplex algorithm [Sch87]. Although not guaranteed to run in time polynomial in the input size, in practice the simplex algorithm is usually quite efficient. In fact, this situation has been explained theoretically by the recent work in "smoothed analysis" [ST01]. The ellipsoid algorithm [GLS81] is guaranteed to run in polynomial time, given the constraints of the LP as input. However, this algorithm is considered impractical, and so is mainly of theoretical interest.

In fact, in some cases we can solve an LP efficiently without giving the constraints explicitly. For the ellipsoid algorithm to run efficiently, one only needs to provide a "separation oracle;" a polynomial-time procedure that determines whether or not a given point satisfies the LP constraints (and if not, provides a hyperplane that separates the point from the polytope).

3.2 Network Flow

Suppose we have a network of pipes carrying water, with a single source of water, and a destination (sink) for the water. We can set the flow rate for every pipe in the system, and our goal is to achieve a certain overall flow rate from the source to the sink. However, each pipe has a certain flow rate capacity that we cannot exceed.

We model this as the following combinatorial optimization problem. The pipe network is given as a directed graph, with two special nodes denoted as the source and the sink. Each edge e in the graph has a *capacity* c_e, and a fixed *demand* d of flow is given at the sink. Our goal is to assign *flow* values f_e to each edge e in the graph such that:

- Every flow value is less than or equal to the capacity of the edge; i.e., $f_e \leq c_e$ for all edges e.

- For every node besides the source and the sink, the total flow entering the node equals the total flow leaving the node; i.e., for all nodes v (besides the source and the sink), we have

  $$\sum_{e \text{ entering } v} f_e = \sum_{e \text{ leaving } v} f_e \ .$$

  This is referred to as *flow conservation*.

- The flow demand is met at the sink; i.e.,

  $$\sum_{e \text{ entering sink}} f_e = d \ .$$

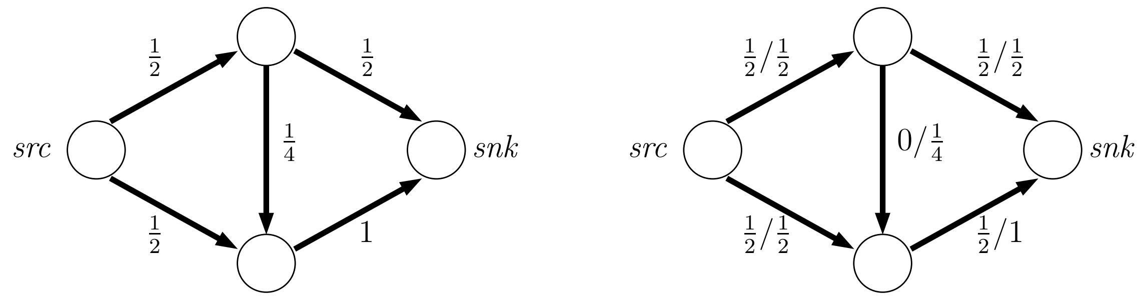


Figure 3-1: An instance of the *network flow* problem, with a demand of one at the sink. The graph on the left shows the problem instance, where each edge is labeled with its capacity. The graph in the right shows the flow values of a feasible flow from the source to the sink (the labels on the edges are flow / capacity).

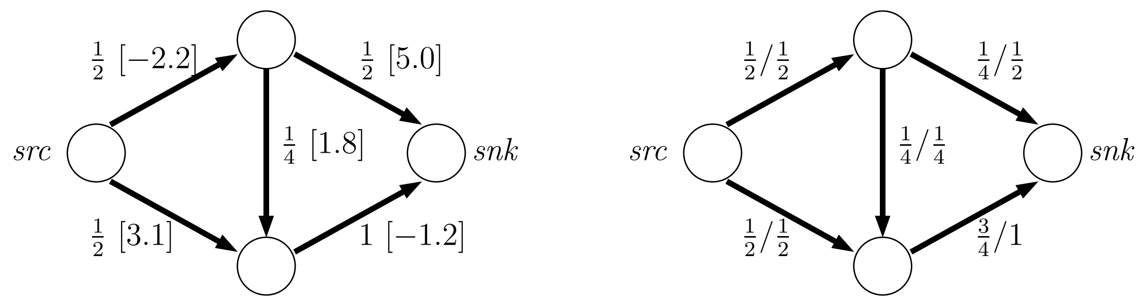


Figure 3-2: An instance of the *min-cost flow* problem, with a demand of one at the sink. The graph on the left shows the problem instance, where each edge is labeled with its capacity and its cost (in brackets). The graph in the right shows the flow values of the optimal feasible flow from the source to the sink (the labels on the edges are flow / capacity).

A flow that meets all the constraints above is said to be a *feasible flow*. An example instance is given in Figure 3-1. In this graph, the demand is 1 at the sink. The solution shown is one of many possible solutions.

This problem is known as *network flow*, and there are many well-studied variations [AMO93]. For the fundamental flow problem above, and many of the basic extensions, efficient algorithms to find feasible flows are known. For a basic introduction to network flow (and many other algorithmic topics), a good source is the textbook of Cormen, Leiserson, Rivest and Stein [CLRS01]. For a thorough presentation of the network flows, we refer the reader to the textbook of Ahuja, Magnanti and Orlin [AMO93].

In the remainder of this section, we discuss the *min-cost flow* problem, and present some of the basic analytical tools that will be needed when analyzing LP relaxations for turbo codes.

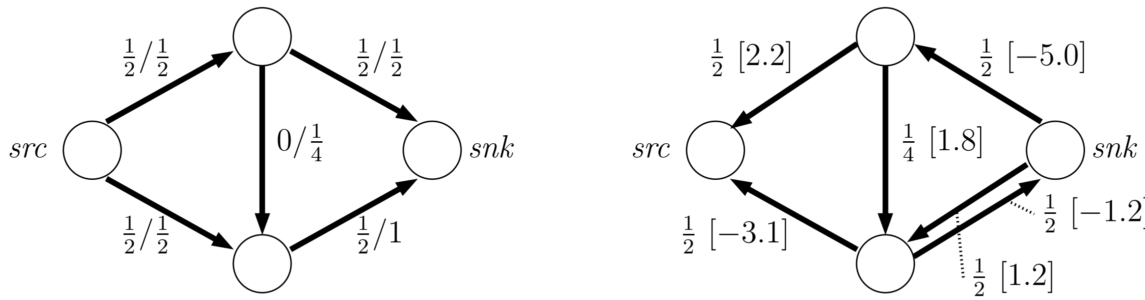


Figure 3-3: A flow and its residual graph. The flow f on the left is a feasible flow for the min-cost flow instance in Figure 3-2. The residual graph G_f on the right contains a negative-cost cycle, so the flow must not be optimal. Sending 1/4 unit of flow around this cycle results in the solution in Figure 3-2, which is optimal.

3.2.1 Min-Cost Flow

In the *min-cost flow* problem, costs are associated with each edge in the graph. The goal in the min-cost flow problem is to find a feasible flow that minimizes cost, where the cost of a flow is the sum, over the edges in the graph, of the cost of that edge times the flow on that edge. For example, suppose we add costs to the graph in Figure 3-1, and obtain the min-cost flow instance in Figure 3-2. Note that the optimal solution to this instance takes advantage of the fact that there is more than one way to construct a feasible flow.

It is not hard to see that min-cost flow is a special case of linear programming. If we make an LP variable f_e for each edge, we can model the feasibility constraints (capacity, conservation, and demand) using LP constraints on the flow variables. The objective function of the LP is the the sum, over the edges in the graph, of the cost times the flow on the edge. Since the min-cost flow algorithm has more structure than a generic LP, combinatorial algorithms are known for min-cost flow that are more efficient than using the simplex or ellipsoid algorithms [AMO93].

The *min-cost circulation* problem is a variant of the min-cost flow problem where the graph contains no source or sink. In this case, a flow is referred to as a *circulation*, since it simply circulates flow around the graph. In other words, a circulation is a setting of flow values that obey edge capacities and flow conservation. The goal of min-cost circulation is to find a feasible circulation that minimizes cost. Note that if the costs are all positive, the optimal solution is to set all flow values to zero; the interesting case of min-cost circulation is when some of the costs are negative.

Residual Graphs. Algorithms for min-cost flow take advantage of an easy test for optimality of a solution, using the *residual graph* of a flow. The residual graph G_f of a flow f (or a circulation) is a graph that models the possible changes to the flow one can make. It is obtained by subtracting out the flow f from the capacities of the graph. An example is shown in Figure 3-3. The residual graph G_f has two edges for each edge in the original graph G, one going in each direction.

Formally, consider an edge $(u \to v)$ in G with capacity c_{uv}. The residual graph

G_f has edges $(u \to v)$ and $(v \to u)$. The capacity of the edge $(u \to v)$ is equal to $c_{uv} - f_{uv}$, where f_{uv} is the flow along edge $(u \to v)$. Thus, the capacity of G_f represents the remaining available capacity along $(u \to v)$ after applying flow f to the original graph G. The cost of edge $(u \to v)$ in G_f is the same as the cost of $(u \to v)$ in G. The capacity of edge $(v \to u)$ in G_f is equal to f_{uv}, and represents the flow that can be sent *back* from v to u by decreasing the flow along $(u \to v)$. The cost of $(v \to u)$ in G_f is equal to the negative of the original cost of $(u \to v)$ in G, since that cost is recovered when flow is sent back. If an edge has zero capacity, it is removed from the residual graph.

Theorem 3.1 **[AMO93]** *A feasible flow f is an optimal min-cost flow or min-cost circulation iff its residual graph G_f contains no negative-cost cycles.*

We can see an example of this theorem in the residual graph of Figure 3-3. In this graph, there is a cycle among the three rightmost vertices, with total cost of -4.4. Now imagine routing $1/4$ unit of flow around this cycle, changing the original flow into the solution in Figure 3-2. We now have a new feasible flow with less cost, since the cycle had negative cost in the residual graph.

In general, the difference $f' - f$ between two feasible flows f' and f becomes a feasible circulation in the residual graph G_f. The cost of the circulation $f' - f$ in G_f is equal to the difference in cost between the two flows in the original graph. So, if there is a flow in the original graph G with cost less than f, it will manifest itself as a negative-cost circulation in G_f. In fact, the converse is also true; all circulations in G_f represent the difference $f' - f$ between feasible flow f' and the flow f. Furthermore, the cost of the circulation is exactly the difference in cost between f' and f.

Path and Cycle Decompositions. A *path flow* is a feasible flow that only sends flow along a single simple path from the source to the sink. In min-cost flow, a *path decomposition* of a flow f is a set of path flows in the graph G from the source to the sink, such that f is the sum of those flows. Every feasible flow has a path decomposition.

In min-cost circulation, a *cycle flow* is a feasible circulation that only sends flow around a single simple cycle. A *cycle decomposition* of a circulation is a set of cycle flows such that the circulation is the sum of the cycle flows. Every circulation has a cycle decomposition.

An *integral* flow is one where all the flow values are integers. Integral flows have integral decompositions; i.e., any integral flow f can be decomposed into *integral* path flows. Similarly, any integral circulation can be decomposed into integral cycle flows.

Chapter 4

LP Decoding of Error-Correcting Codes

In this thesis we give a new approach to the problem of decoding an error-correcting code. The approach is motivated by viewing the ML decoding problem as a combinatorial optimization problem. A classic example of such a problem is the VERTEX COVER problem defined in the previous chapter.

Abstractly, an optimization problem can be viewed as finding an element in a set that minimizes (or maximizes) some predefined function on elements of the set. In the VERTEX COVER problem, we have the set of all legal vertex covers, and the function is the cost of the vertices in the particular cover. In the ML decoding problem, we have the set of all codewords, and the function is the likelihood that that $\tilde{y}$ was received, given that the particular codeword was sent.

In this thesis we design LP relaxations for the maximum-likelihood decoding problem. We refer to our new decoder as the "LP decoder." While the ML decoding problem does have the basic elements of an optimization problem, it differs from conventional optimization problems in the sense of what we want out of an approximate solution. In VERTEX COVER, in order to be able to run an efficient algorithm, we settle for a solution whose cost is within a factor of two of the optimum. However, in ML decoding we are not concerned with finding an approximate solution (i.e., a "fairly likely" codeword). We are concerned with the probability, over the noise in the channel, that we actually find the transmitted codeword. Additionally, in coding we often have the freedom to choose the code itself, which defines the subset over which we are optimizing; in most other optimization problems, we are expected to handle arbitrary instances of the problem.

Chapter Outline. In this chapter we outline the general technique of LP decoding, as it applies to any binary code, over any binary-input memoryless channel. We define the notion of a *proper* polytope for an LP relaxation, which is required for the LP decoder to have the ML certificate property. We discuss the details of using the LP relaxation as a decoder, and how to describe the success conditions of the decoder exactly.

We also define the *fractional distance* of an LP relaxation, which is a generalization of the classical distance. Recall that ML decoders correct a number of errors up to half the (classical) distance in the binary symmetric channel. We prove that LP decoders correct a number of errors up to half the *fractional* distance.

Finally, we discuss the application of LP decoding to binary linear codes. We define the notion of a polytope being $\mathcal{C}$-*symmetric* for a binary linear code $\mathcal{C}$. We show that if the polytope is $\mathcal{C}$-symmetric, one may assume that the all-zeros codeword is transmitted over the channel. This greatly simplifies analysis. Furthermore, polytope symmetry has some implications in terms of the fractional distance. In later chapters we will give polytopes that exhibit symmetry.

4.1 An LP Relaxation of ML Decoding

Our LP relaxations for decoding will have LP variables f_i for each code bit. We would like these to take on binary values; however, in the relaxation, they will be relaxed to take on values between zero and one. We will define some additional linear constraints on the variables that are a function of the structure of the code itself, and obtain a polytope $\mathcal{P} \subseteq [0,1]^n$. We call a polytope $\mathcal{P}$ *proper* if the set of integral points in $\mathcal{P}$ is exactly the set $\mathcal{C}$ of codewords; i.e.,

$$\mathcal{P} \cap \{0,1\}^n = \mathcal{C}. \tag{4.1}$$

Let $\mathcal{V}(\mathcal{P})$ be the set of vertices of the polytope $\mathcal{P}$. Recall that a vertex is a point in the polytope that cannot be expressed as the convex combination of other points in the polytope. Note that for any proper polytope $\mathcal{P}$ (where equation (4.1) holds), we have that every codeword $y \in \mathcal{C}$ is a vertex of $\mathcal{P}$. This is the case because in the unit hypercube $[0,1]^n$, every binary word of length n cannot be expressed as the convex combination of points in the hypercube; since $\mathcal{P}$ is contained within the hypercube $[0,1]^n$, we have that every binary word of length n (including all the codewords) cannot be expressed as the convex combination of points inside $\mathcal{P}$. It may not be the case that every polytope vertex is a codeword, however, since the polytope could have fractional vertices. So, in general we have

$$\mathcal{C} \;\subseteq\; \mathcal{V}(\mathcal{P}) \;\subseteq\; \mathcal{P} \;\subseteq\; [0,1]^n.$$

The objective function of our LP will measure the likelihood of a particular setting of the $\{f_i\}$ variables, given the received word $\tilde{y}$. In Section 2.5 we defined a cost γ_i for a code bit such that the minimum-cost codeword is the ML codeword, for any binary-input memoryless channel. We will use this as the objective function of our LP. Overall, our LP solves the following system:

$$\text{minimize} \sum_{i=1}^{n} \gamma_i f_i \quad \text{s.t.} \quad f \in \mathcal{P} \tag{4.2}$$

Define the *cost* of a point $f \in \mathcal{P}$ as $\sum_{i=1}^{n} \gamma_i f_i$. Our LP will find the point in $\mathcal{P}$ with minimum cost. Notice that the only part of the LP relaxation that depends on the received vector is the objective function.

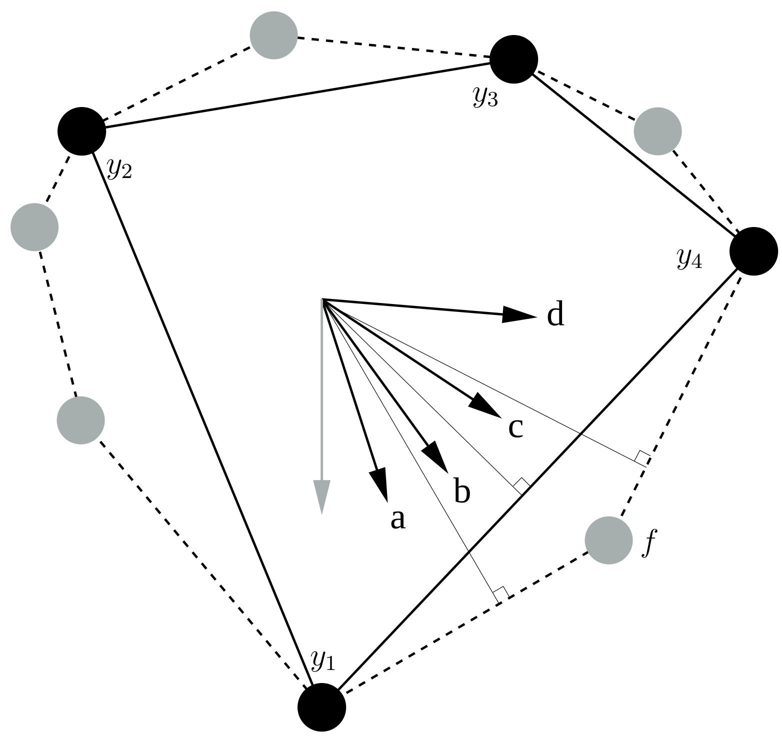


Figure 4-1: An illustration of decoding an error-correcting code using linear programming relaxation, and the possible cases for the objective function.

Figure 4-1 provides an abstract visualization of the relaxation $\mathcal{P}$. This figure is of course two-dimensional, but all the elements of the LP relaxation can still be seen. The dotted line represents the polytope $\mathcal{P}$, and the circles represent vertices of the polytope. The black circles in the figure represent codewords, and the gray circles represent fractional vertices that are not codewords. The inner solid line encloses the *convex hull* of the codewords, the set of points that are convex combinations of codewords. The arrows inside the polytope represent various cases for the objective function, which depend on the noise in the channel. We will go through these cases explicitly later in the discussion.

4.2 The LP Relaxation as a Decoder

Maximum-likelihood (ML) decoding can be seen as optimizing the objective function over points inside the convex hull of the codewords (solid line in Figure 4-1); since every point in this convex hull is a convex combination of codewords, then the optimum will obtained at a codeword. Unfortunately, this convex hull will be too complex to represent explicitly for any code for which ML decoding is NP-hard (unless P = NP).

Our decoding algorithm using the relaxed polytope $\mathcal{P}$ is the following: solve the LP given in equation (4.2). If the LP solution is integral (a black vertex in Figure 4-1), output the corresponding codeword. If the LP solution is fractional (a gray vertex in Figure 4-1), output "error." Note that if this algorithm outputs an integral solution (codeword), then we know it outputs the ML codeword. This is because the cost of the codeword found is at most the cost of all the points in $\mathcal{P}$, including all the other codewords. Therefore this decoder has what we call the *ML certificate* property: if it outputs a codeword, it is guaranteed to be the ML codeword. This property is one of the unique advantages of LP decoding.

4.2.1 Noise as a perturbation of the LP objective.

Suppose the relaxation has the following reasonable property: when there is no noise in the channel, the transmitted codeword y will be the optimum point of the LP, and thus the LP decoder will succeed. (All the relaxations we give in this thesis have this property.) Noise in the channel then amounts to a perturbation of the objective function away from the "no noise" direction. If the perturbation is small, then y will remain the optimal point of the LP. If the perturbation is large (there is a lot of noise in the channel), then y will no longer be optimal, and the LP decoder will fail.

In Figure 4-1, an objective function can be seen as a *direction* inside the polytope; solving the LP amounts to finding the point in the polytope that is furthest in that direction. The following mental exercise often helps visualize linear programming. Rotate the polytope such that the objective function points "down." Then, the objective function acts like gravity; if we drop a ball inside this polytope, it will settle at the point that optimizes the objective function. In Figure 4-1 we have rotated the polytope so that when there is no noise in the channel, then the objective function points "down," directly at the transmitted codeword (y_1 in the figure).

There are four cases describing the success of LP decoding, related to ML decoding. These cases are illustrated in Figure 4-1 by four arrows (a, b, c, d), representing directions inside the polytope. The gray arrow is the objective function without noise in the channel. The cases are described as follows:

(a) If there is very little noise, then both ML decoding and LP decoding succeed, since both still have y_1 as the optimal point.

(b) If more noise is introduced, then ML decoding succeeds, but LP decoding fails, since the fractional vertex f is optimal for the relaxation.

(c) With still more noise, ML decoding fails, since y_2 is now optimal; LP decoding still has a fractional optimum (f), so this error is detected.

(d) Finally, with a lot of noise, both ML decoding and LP decoding have y_2 as the optimum, so both fail, and the error is undetected.

Note that in the last two cases when ML decoding fails, this is in some sense the fault of the code itself, rather than the decoder.

4.2.2 Success Conditions for LP Decoding

Overall, the LP decoder succeeds if the transmitted codeword is the unique optimal solution to the LP. The decoder fails if the transmitted codeword is not an optimal solution to the LP. In the case of multiple LP optima (which for many noise models has zero probability), we will be conservative and assume that the LP decoder fails. Therefore,

Theorem 4.1 *For any binary-input memoryless channel, an LP decoder using polytope $\mathcal{P}$ will fail if and only if there is some point in $\mathcal{P}$ other than the transmitted codeword y with cost less than or equal to the cost of y.*

This theorem allows us to characterize the probability of error, given a particular transmitted codeword y, as:

$$\Pr[\text{err} \,|\, y] = \Pr\left[\exists f \in \mathcal{P}, f \neq y \, : \, \sum_i \gamma_i f_i \leq \sum_i \gamma_i y_i\right] \tag{4.3}$$

In the upcoming chapters we will derive specific relaxations for turbo codes and LDPC codes. We will apply the generic characterization of LP decoding success to these relaxations, giving more precise conditions for success based on the combinatorial properties of the code and the relaxation. We then use these more precise characterizations to derive bounds on the performance of LP decoding.

4.3 The Fractional Distance

We motivate the definition of fractional distance by providing an alternate definition for the (classical) distance in terms of a proper polytope $\mathcal{P}$. Recall that in a proper polytope $\mathcal{P}$, there is a one-to-one correspondence between codewords and integral vertices of $\mathcal{P}$; i.e., $\mathcal{C} = \mathcal{P} \cap \{0,1\}^n$. The Hamming distance between two points in the discrete space $\{0,1\}^n$ is equivalent to the l_1 distance between the points in the space $[0,1]^n$. Therefore, given a proper polytope $\mathcal{P}$, we may define the distance of a code as the minimum l_1 distance between two integral vertices, i.e.,

$$d \;=\; \min_{\substack{y,y' \in (\mathcal{V}(\mathcal{P}) \cap \{0,1\}^n) \\ y \neq y'}} \; \sum_{i=1}^{n} |y_i - y'_i|.$$

The LP polytope $\mathcal{P}$ may have additional *non-integral* vertices, as illustrated in Figure 4-1. We define the *fractional distance* d_{frac} of a polytope $\mathcal{P}$ as the minimum l_1 distance between an integral vertex (codeword) and any *any other* vertex of $\mathcal{P}$; i.e.,

$$d_{frac} \;=\; \min_{\substack{y \in \mathcal{C} \\ f \in \mathcal{V}(\mathcal{P}) \\ f \neq y}} \; \sum_{i=1}^{n} |y_i - f_i|$$

This fractional distance has connections to the minimum weight of a pseudocodeword, as defined by Wiberg [Wib96], and also studied by Forney et. al [FKKR01].

Note that this fractional distance is always a lower bound on the classical distance of the code, since every codeword is a polytope vertex (in the set $\mathcal{V}(\mathcal{P})$). Moreover, the performance of LP decoding is tied to this fractional distance, as we make precise in the following:

Theorem 4.2 *Let $\mathcal{C}$ be a binary code and $\mathcal{P}$ a proper polytope in an LP relaxation for $\mathcal{C}$. If the fractional distance of $\mathcal{P}$ is d_{frac}, then the LP decoder using $\mathcal{P}$ is successful if at most $\lceil d_{frac}/2 \rceil - 1$ bits are flipped by the binary symmetric channel.*

Proof: Let y be the codeword transmitted over the channel. Suppose the LP decoder fails; i.e., y is not the unique optimum solution to the LP. Then there must be some other vertex $f^* \in \mathcal{V}(\mathcal{P})$ (where $f^* \neq y$) that is an optimum solution to the LP, since the LP optimum is always obtained at a vertex. By the definition of fractional distance, we have

$$\sum_{i=1}^{n} |f_i^* - y_i| \geq d_{frac}.$$

For all bits $i \in \{1, \ldots, n\}$, let $f_i = |f_i^* - y_i|$. From the above equation, we have

$$\sum_{i=1}^{n} f_i \geq d_{frac}. \tag{4.4}$$

Let $\mathcal{E} = \{i : \tilde{y}_i \neq y_i\}$ be the set of bits flipped by the channel. By assumption, we have that

$$|\mathcal{E}| \leq \lceil d_{frac}/2 \rceil - 1,$$

and so

$$\sum_{i \in \mathcal{E}} f_i \leq \lceil d_{frac}/2 \rceil - 1, \tag{4.5}$$

since all $f_i \leq 1$. From (4.4) and (4.5), it follows that

$$\sum_{i \notin \mathcal{E}} f_i \geq \lfloor d_{frac}/2 \rfloor + 1. \tag{4.6}$$

Therefore, from (4.5) and (4.6), we have

$$\sum_{i \notin \mathcal{E}} f_i - \sum_{i \in \mathcal{E}} f_i > 0. \tag{4.7}$$

Since f^* is an optimum solution to the LP, its cost must be less than or equal to the cost of y under the objective function γ; i.e.,

$$\sum_{i=1}^{n} \gamma_i f_i^* - \sum_{i=1}^{n} \gamma_i y_i \leq 0. \tag{4.8}$$

We can rewrite the left side of equation (4.8) as follows:

$$\begin{aligned}
\sum_{i=1}^{n} \gamma_i f_i^* - \sum_{i=1}^{n} \gamma_i y_i &= \sum_{i=1}^{n} \gamma_i (f_i^* - y_i) \\
&= \sum_{i:y_i=0} \gamma_i f_i^* - \sum_{i:y_i=1} \gamma_i (1 - f_i^*) \\
&= \sum_{i:y_i=0} \gamma_i f_i - \sum_{i:y_i=1} \gamma_i f_i \qquad (4.9) \\
&= \sum_{i \notin \mathcal{E}} f_i - \sum_{i \in \mathcal{E}} f_i. \qquad (4.10)
\end{aligned}$$

Equation (4.9) follows from the fact that

$$f_i = \begin{cases} f_i^* & \text{if } y_i = 0 \\ 1 - f_i^* & \text{if } y_i = 1 \end{cases} .$$

Equation (4.10) follows from the fact that under the BSC, we have $\gamma_i = -1$ if $\tilde{y}_i = 1$, and $\gamma_i = +1$ if $\tilde{y}_i = 0$. From (4.8) and (4.10), it follows that

$$\sum_{i \notin \mathcal{E}} f_i - \sum_{i \in \mathcal{E}} f_i \leq 0,$$

which contradicts (4.7). ■

Note the analogy to the classical case: just as ML decoding has a performance guarantee in terms of classical distance, Theorem 4.2 establishes that the LP decoder has a performance guarantee specified by the fractional distance of the code.

In Chapter 5, we will give a polytope for LDPC codes, and show that there exist LDPC codes where the fractional distance of the polytope grows like $\Omega(n^{1-\epsilon})$, for a constant ϵ.

4.3.1 The Max-Fractional Distance

We can slightly refine the notion of fractional distance by observing some slack in the proof of Theorem 4.2. To get equation 4.5, the proof essentially assumes that there is a fractional vertex that can set $f_i = 1 - y_i$ for every bit flipped by the channel. As a first step toward refining this argument, we define the notion of the *max-fractional distance*. In fact, our bounds for LDPC codes apply to this refined notion of fractional distance.

Formally, the max-fractional distance of a proper polytope $\mathcal{P} \in [0, 1]^n$ for code $\mathcal{C}$ is equal to

$$d_{frac}^{\max} = \min_{\substack{y \in \mathcal{C} \\ f \in \mathcal{V}(\mathcal{P}) \\ f \neq y}} \left[\frac{\sum_{i=1}^{n} |y_i - f_i|}{\max_{i=1}^{n} |y_i - f_i|} \right]$$

Using identical arguments, we can easily derive a theorem for the max-fractional distance analogous to Theorem 4.2 for the fractional distance:

Theorem 4.3 *Let $\mathcal{C}$ be a binary code and $\mathcal{P}$ a proper polytope in an LP relaxation for $\mathcal{C}$. If the max-fractional distance of $\mathcal{P}$ is $d_{frac}^{\max}$, then the LP decoder using $\mathcal{P}$ is successful if at most $\lceil d_{frac}^{\max}/2 \rceil - 1$ bits are flipped by the binary symmetric channel.*

The exact relationship between d_{frac} and $d_{frac}^{\max}$ is an interesting question. Clearly $d_{frac}^{\max} \geq d_{frac}$ in general, since $\max_i f_i$ is always at most 1. For the LDPC relaxation we give in Chapter 5, the two quantities differ by at most a constant.

4.4 Symmetric Polytopes for Binary Linear Codes

Binary linear codes have some special algebraic structure that we can exploit when we analyze decoding algorithms. For example, for most message-passing decoders, one may assume without loss of generality that the all-zeros codeword 0^n was transmitted. (Recall that 0^n is always a codeword of a binary linear code.) This greatly simplifies analysis (and notation). Furthermore, the distance of a binary linear code is equal to the lowest *weight* of any non-zero codeword, where the weight of a codeword y is equal to $\sum_i y_i$.

In this section we discuss the application of LP decoding to binary linear codes. We define the notion of a polytope being *$\mathcal{C}$-symmetric* for a particular binary linear code $\mathcal{C}$. We then show that if a decoder uses a $\mathcal{C}$-symmetric polytope, then the all-zeros assumption is valid, and the fractional distance is equal to the lowest weight of any non-zero polytope vertex. This not only simplifies analysis in later chapters, it also allows us to *efficiently compute* the fractional distance of a polytope.

4.4.1 $\mathcal{C}$-symmetry of a polytope

For a point $f \in [0,1]^n$, we define its *relative point* $f^{[y]} \in [0,1]^n$ with respect to codeword y as follows: for all $i \in \{1, \dots, n\}$, let $f_i^{[y]} = |f_i - y_i|$. Note that this operation is its own inverse; i.e., the relative point of $f^{[y]}$ with respect to y is the original point f. Intuitively, the point $f^{[y]}$ is the point that has the same spatial relation to the point 0^n as f has to the codeword y (and vice-versa).

Definition 4.4 *A proper polytope $\mathcal{P}$ for the binary code $\mathcal{C}$ is* **$\mathcal{C}$-symmetric** *if, for all points f in the polytope $\mathcal{P}$ and codewords y in the code $\mathcal{C}$, the relative point $f^{[y]}$ is also contained in the polytope $\mathcal{P}$.*

Note that the definition of $\mathcal{C}$-symmetry only allows for proper polytopes for binary linear codes. In a sense, the definition generalizes the notion of a binary linear code. We make this formal in the following:

Theorem 4.5 *If a polytope $\mathcal{P}$ is proper for the binary code $\mathcal{C}$, and $\mathcal{P}$ is $\mathcal{C}$-symmetric, then $\mathcal{C}$ must be linear.*

Proof: Recall that a binary code is linear if $0^n \in \mathcal{C}$ and $(y + y') \in \mathcal{C}$ for all distinct $y, y' \in \mathcal{C}$. For any codeword $y \in \mathcal{C} \subseteq \mathcal{P}$, we have $y^{[y]} = 0^n$. Because $\mathcal{P}$ is $\mathcal{C}$-symmetric, we conclude that $0^n \in \mathcal{P}$; since $\mathcal{P}$ is proper, $0^n \in \mathcal{C}$. Furthermore, for two distinct codewords $y, y' \in \mathcal{C}$, we have $y^{[y']} = y + y'$; therefore, $(y+y') \in \mathcal{P}$, and so $(y+y') \in \mathcal{C}$, and $\mathcal{C}$ must be linear. ■

4.4.2 Turning Fractional Distance into Fractional Weight

The (classical) distance of a binary linear code is equal to the minimum weight of a non-zero codeword. It would be very convenient to have a similar result for fractional distance. In this section we establish that the fractional distance of a $\mathcal{C}$-symmetric polytope is equal to the the minimum weight of a non-zero vertex of $\mathcal{P}$. Before proving this fact, we must show that relative points of vertices are also vertices:

Theorem 4.6 *Let $\mathcal{P}$ be a $\mathcal{C}$-symmetric polytope $\mathcal{P}$. Then, for all vertices f of $\mathcal{P}$, for all codewords $y \in C$, the relative point $f^{[y]}$ is also a vertex of $\mathcal{P}$.*

Proof: Suppose not, and there is some vertex f and codeword y such that $f^{[y]}$ is not a vertex of $\mathcal{P}$. By definition of $\mathcal{C}$-symmetric, we have $f^{[y]} \in \mathcal{P}$. Since $f^{[y]}$ is not a vertex, it must be a convex combination of two points $a, b \in \mathcal{P}$, where the three points a, b and $f^{[y]}$ are all distinct. In other words,

$$f^{[y]} = \lambda a + (1 - \lambda) b, \tag{4.11}$$

for some scalar λ where $0 < \lambda < 1$.

Consider the relative points $a^{[y]}$ and $b^{[y]}$, both of which must be in $\mathcal{P}$, since $\mathcal{P}$ is $\mathcal{C}$-symmetric. Note that $a^{[y]}$, $b^{[y]}$ and f must also be distinct, which follows from the fact that the operation of taking a relative point is its own inverse. We claim that $f = \lambda a^{[y]} + (1 - \lambda) b^{[y]}$. This contradicts the fact that f is a vertex.

It remains to show that $f = \lambda a^{[y]} + (1 - \lambda) b^{[y]}$. In other words, we must show the following:

$$f_i = \lambda a_i^{[y]} + (1 - \lambda) b_i^{[y]} \ \text{ for all } \ i \in \{1, \ldots, n\} \tag{4.12}$$

Consider some code bit y_i. If $y_i = 0$, then

$$f_i = f_i^{[y]}, \ \ a_i = a_i^{[y]} \ \text{ and } \ b_i = b_i^{[y]},$$

and (4.12) follows from (4.11). If $y_i = 1$, we have

$$\begin{aligned} f_i &= 1 - f_i^{[y]} \\ &= 1 - (\lambda a_i + (1 - \lambda) b_i) \\ &= 1 - \Big(\lambda (1 - a_i^{[y]}) + (1 - \lambda)(1 - b_i^{[y]})\Big) \\ &= \lambda a_i^{[y]} + (1 - \lambda) b_i^{[y]}, \end{aligned}$$

giving (4.12). ■

Now we are ready to prove the main result of the section. Recall that the weight of a codeword y is equal to $\sum_i y_i$. Define the weight of a point f in $\mathcal{P}$ to be equal to $\sum_i f_i$.

Theorem 4.7 *The fractional distance of a $\mathcal{C}$-symmetric polytope $\mathcal{P}$ for a binary linear code $\mathcal{C}$ is equal to the minimum weight of a non-zero vertex of $\mathcal{P}$.*

Proof: Let $f^{\min}$ be the minimum-weight non-zero vertex of $\mathcal{P}$; i.e.,

$$f^{\min} = \underset{f \in (\mathcal{V}(\mathcal{P}) \setminus 0^n)}{\arg\min} \sum_i f_i.$$

Since the l_1 distance between $f^{\min}$ and 0^n is equal to the weight of $f^{\min}$, we have that the fractional distance of $\mathcal{P}$ is at most the weight of $f^{\min}$. Suppose it is strictly less; i.e., $d_{frac} < \sum_i f_i^{\min}$. Then, there is some vertex $f \in \mathcal{V}(\mathcal{P})$ and codeword $y \neq f$ where

$$\sum_i |f_i - y_i| < \sum_i f_i^{\min}. \tag{4.13}$$

Consider the point $f^{[y]}$. By Theorem 4.6, $f^{[y]}$ is a vertex of $\mathcal{P}$. Furthermore, $f^{[y]}$ must be non-zero since $y \neq f$. Since $f_i^{[y]} = |f_i - y_i|$ for all i, equation (4.13) implies that the weight of $f^{[y]}$ is less than the weight of $f^{\min}$, a contradiction. ■

Define the *normalized weight* of a point f in $\mathcal{P}$ as $(\sum_i f_i)/(\max_i f_i)$. Using this normalized weight, we obtain a theorem for max-fractional distance analogous to Theorem 4.7:

Theorem 4.8 *The max-fractional distance of a $\mathcal{C}$-symmetric polytope $\mathcal{P}$ for a binary linear code $\mathcal{C}$ is equal to the minimum normalized weight of a non-zero vertex of $\mathcal{P}$.*

4.4.3 Computing the Fractional Distance

In contrast to the classical distance, the fractional distance of a $\mathcal{C}$-symmetric polytope $\mathcal{P}$ for a binary linear code $\mathcal{C}$ can be computed efficiently. This can be used to bound the worst-case performance of LP decoding for a particular code and polytope. Since the fractional distance is a lower bound on the real distance, we thus have an efficient algorithm to give a non-trivial lower bound on the distance of a binary linear code.

Let $\mathcal{V}^-(\mathcal{P}) = \mathcal{V}(\mathcal{P}) \setminus 0^n$ be the set of non-zero vertices of $\mathcal{P}$. To compute the fractional distance, we must compute the minimum weight vertex in $\mathcal{V}^-(\mathcal{P})$. We consider instead a more general problem: given the m constraints of a polytope P over variables $(x_1, \ldots, x_n)$, a specified vertex x^0 of P, and a linear function $\ell(x)$, find the vertex x^1 in P *other than* x^0 that minimizes $\ell(x)$. For our problem, we are interested in the $\mathcal{C}$-symmetric polytope $\mathcal{P}$, the special vertex $0^n \in \mathcal{P}$, and the linear function $\sum_i f_i$.

An efficient algorithm for this general problem is the following: let $\mathcal{F}$ be the set of all constraints of P for which x^0 is not tight. (We say a point is *tight* for a constraint if it meets the constraint with equality.) Now for each constraint in $\mathcal{F}$ do the following.

Define a new P' by making the constraint into an equality constraint, then optimize $\ell(x)$ over P'. The minimum value obtained over all constraints in $\mathcal{F}$ is the minimum of $\ell(x)$ over all vertices x^1 other than x^0. The running time of this algorithm is equal to the time taken by $|\mathcal{F}| < m$ calls to an LP solver.

This algorithm is correct by the following argument. It is well known that a vertex of a polytope of dimension D is uniquely determined by giving D linearly independent constraints of the polytope for which the vertex is tight. Using this fact, it is clear that the vertex x^1 we are looking for must be tight for some constraint in $\mathcal{F}$; otherwise, it would be the same point as x^0. Therefore, at some point in our procedure, each potential x^1 is considered. Furthermore, for each P' considered during the algorithm, we have that all vertices of P' are vertices of P not equal to x^0. Therefore the point x^1 we are looking for will be output by the algorithm.

4.4.4 All-Zeros Assumption

When analyzing linear codes, it is common to assume that the codeword sent over the channel is the all-zeros vector (i.e., $y = 0^n$), since it tends to simplify analysis. In the context of our LP decoder, however, the validity of this assumption is not immediately clear. In this section, we prove that one *can* make the all-zeros assumption when analyzing LP decoders, as long as the polytope used in the decoder is $\mathcal{C}$-symmetric.

Theorem 4.9 *For any LP decoder using a $\mathcal{C}$-symmetric polytope to decode $\mathcal{C}$ under a binary-input memoryless symmetric channel, the probability that the LP decoder fails is independent of the codeword that is transmitted.*

Proof: We use $\Pr[err \,|\, y]$ to denote the probability that the LP decoder makes an error, given that y was transmitted.

For an arbitrary transmitted word y, we need to show that $\Pr[\text{err} \,|\, y] = \Pr[\text{err} \,|\, 0^n]$. Define $\text{BAD}(y) \subseteq \Sigma^n$ to be the set of received words $\tilde{y}$ that cause decoding failure, assuming y is transmitted. The set $\text{BAD}(y)$ consists of all the possible received words that cause the transmitted codeword *not* to be the unique LP optimum:

$$\text{BAD}(y) = \left\{ \tilde{y} : \exists f \in \mathcal{P}, f \neq y, \text{ where } \sum_i \gamma_i f_i \leq \sum_i \gamma_i y_i \right\}$$

Note that in the above, the cost vector γ is a function of the received word $\tilde{y}$. Furthermore, we have considered the case of multiple LP optima to be decoding failure. Rewriting equation (4.3), we have that for all codewords y,

$$\Pr[\text{err} \,|\, y] = \sum_{\substack{\tilde{y} \in \Sigma^n, \\ \tilde{y} \in \text{BAD}(y)}} \Pr[\ \tilde{y} \mid y\ ]. \tag{4.14}$$

Applying this to the codeword 0^n, we get

$$\Pr[\mathrm{err} \,|\, 0^n] = \sum_{\substack{\tilde{y} \in \Sigma^n, \\ \tilde{y} \in \mathrm{BAD}(0^n)}} \Pr[\ \tilde{y} \mid 0^n\ ]. \tag{4.15}$$

We will show that the space Σ^n of possible received vectors can be partitioned into pairs $(\tilde{y}, \tilde{y}^0)$ such that $\Pr[\ \tilde{y} \mid y\ ] = \Pr[\ \tilde{y}^0 \mid 0^n\ ]$, and $\tilde{y} \in \mathrm{BAD}(y)$ if and only if $\tilde{y}^0 \in \mathrm{BAD}(0^n)$. This, along with equations (4.14) and (4.15), gives $\Pr[\mathrm{err} \,|\, y] = \Pr[\mathrm{err} \,|\, 0^n]$.

The partition is performed according to the symmetry of the channel. Fix some received vector $\tilde{y}$. Define $\tilde{y}^0$ as follows: let $\tilde{y}^0_i = \tilde{y}_i$ if $y_i = \mathtt{0}$, and $\tilde{y}^0_i = \tilde{y}'_i$ if $y = \mathtt{1}$, where $\tilde{y}'_i$ is the symmetric symbol to $\tilde{y}_i$ in the channel. (See Section 2.4 for details on symmetry.) Note that this operation is its own inverse and therefore gives a valid partition of Σ^n into pairs.

First we show that $\Pr[\ \tilde{y} \mid y\ ] = \Pr[\ \tilde{y}^0 \mid 0^n\ ]$. From the channel being memoryless, we have

$$\begin{aligned}
\Pr[\ \tilde{y} \mid y\ ] = \prod_{i=1}^{n} \Pr[\tilde{y}_i \mid y_i] &= \prod_{i:y_i=\mathtt{0}} \Pr[\tilde{y}_i \mid 0] \prod_{i:y_i=\mathtt{1}} \Pr[\tilde{y}_i \mid 1] \\
&= \prod_{i:y_i=\mathtt{0}} \Pr[\tilde{y}^0_i \mid 0] \prod_{i:y_i=\mathtt{1}} \Pr[\tilde{y}_i \mid 1] && (4.16) \\
&= \prod_{i:y_i=\mathtt{0}} \Pr[\tilde{y}^0_i \mid 0] \prod_{i:y_i=\mathtt{1}} \Pr[\tilde{y}'_i \mid 0] && (4.17) \\
&= \prod_{i:y_i=\mathtt{0}} \Pr[\tilde{y}^0_i \mid 0] \prod_{i:y_i=\mathtt{1}} \Pr[\tilde{y}^0_i \mid 0] && (4.18) \\
&= \Pr[\ \tilde{y}^0 \mid 0^n\ ]
\end{aligned}$$

Equations (4.16) and (4.18) follow from the definition of $\tilde{y}^0$, and equation (4.17) follows from the symmetry of the channel (equations (2.2) and (2.3)).

Now it remains to show that $\tilde{y} \in \mathrm{BAD}(y)$ if and only if $\tilde{y}^0 \in \mathrm{BAD}(0^n)$. Let γ be the cost vector when $\tilde{y}$ is received, and let γ^0 be the cost vector when $\tilde{y}^0$ is received, as defined in equation (2.5).

Suppose $y_i = 0$. Then, $\tilde{y}_i = \tilde{y}^0_i$, and so $\gamma_i = \gamma^0_i$. Now suppose $y_i = 1$; then $\tilde{y}^0_i = \tilde{y}'_i$, and so

$$\begin{aligned}
\gamma^0_i &= \log\left(\frac{\Pr[\tilde{y}'_i \mid y_i = 0]}{\Pr[\tilde{y}'_i \mid y_i = 1]}\right) \\
&= \log\left(\frac{\Pr[\tilde{y}_i \mid y_i = 1]}{\Pr[\tilde{y}_i \mid y_i = 0]}\right) && (4.19) \\
&= -\gamma_i.
\end{aligned}$$

Equation (4.19) follows from the symmetry of the channel (equations (2.2) and (2.3)). We conclude that

$$\gamma_i = \gamma_i^0 \text{ if } y_i = 0, \text{ and } \gamma_i = -\gamma_i^0 \text{ if } y_i = 1. \tag{4.20}$$

Fix some point $f \in \mathcal{P}$ and consider the relative point $f^{[y]}$. We claim that the difference in cost between f and y is the same as the difference in cost between $f^{[y]}$ and 0^n. This is shown by the following:

$$\begin{aligned}
\sum_i \gamma_i f_i - \sum_i \gamma_i y_i &= \sum_i \gamma_i (f_i - y_i) \\
&= \sum_{i:y_i=0} \gamma_i f_i + \sum_{i:y_i=1} \gamma_i (f_i - 1) \\
&= \sum_{i:y_i=0} \gamma_i f_i^{[y]} - \sum_{i:y_i=1} \gamma_i f_i^{[y]} && (4.21) \\
&= \sum_{i:y_i=0} \gamma_i^0 f_i^{[y]} + \sum_{i:y_i=1} \gamma_i^0 f_i^{[y]} && (4.22) \\
&= \sum_i \gamma_i^0 f_i^{[y]} \\
&= \sum_i \gamma_i^0 f_i^{[y]} - \sum_i \gamma_i^0 0_i && (4.23)
\end{aligned}$$

Equation (4.21) follows from the definition of $f^{[y]}$, and equation (4.22) follows from equation (4.20).

Now suppose $\tilde{y} \in \text{BAD}(y)$, and so by the definition of BAD there is some $f \in \mathcal{P}$, where $f \neq y$, such that $\sum_i \gamma_i f_i - \sum_i \gamma_i y_i \leq 0$. By equation (4.23), we have that $\sum_i \gamma_i^0 f_i^{[y]} - \sum_i \gamma_i^0 0_i^n \leq 0$. Because $\mathcal{P}$ is $\mathcal{C}$-symmetric, $f^{[y]} \in \mathcal{P}$, and by the fact that $f \neq y$, we have that $f^{[y]} \neq 0^n$. Therefore $\tilde{y}^0 \in \text{BAD}(0^n)$. A symmetric argument shows that if $\tilde{y}^0 \in \text{BAD}(0^n)$ then $\tilde{y} \in \text{BAD}(y)$. ■

Since the all-zeros codeword has zero cost, we have the following corollary to Theorem 4.1:

Corollary 4.10 *For any binary linear code $\mathcal{C}$ over any binary-input memoryless symmetric channel, the LP decoder using $\mathcal{C}$-symmetric polytope $\mathcal{P}$ will fail if and only if there is some non-zero point in $\mathcal{P}$ with cost less than or equal to zero.*

4.5 Conclusion

In this chapter we outlined the basic technique of LP decoding of binary codes. We derived general success conditions for an LP decoder, and showed that any decoder using a proper polytope has the ML certificate property. The fractional distance of a polytope was defined, and it was shown that LP decoders correct a number of errors up to half the fractional distance.

Furthermore, for binary linear codes, we established symmetry conditions for the polytope that allow for the all-zeros assumption, and regarding fractional distance as fractional weight.

In the chapters to come, we will use this technique for LDPC codes and turbo codes. We will use the success conditions to derive performance bounds for our decoders, and to compare the performance with other known decoders.

Chapter 5

LP Decoding of Low-Density Parity-Check Codes

Low-density parity-check (LDPC) codes were discovered by Gallager in 1962 [Gal62]. In the 1990s, they were "rediscovered" by a number of researchers [Mac99, Wib96, SS96], and have since received a lot of attention. The error-correcting performance of these codes is unsurpassed; in fact, Chung et al. [CFRU01] have given a family of LDPC codes whose error rate comes within a factor of approximately 1.001 (0.0045 dB) of the capacity of the channel, as the block length goes to infinity. The decoders most often used for this family are based on the *belief-propagation* algorithm [MMC98], where messages are iteratively sent across a factor graph for the code. While the performance of this decoder is quite good in practice, analyzing its behavior is often difficult when the factor graph contains cycles.

In this chapter, we introduce an LP relaxation for an arbitrary binary linear code. The polytope for the relaxation is a function of the factor graph representation of the code. We have an LP variable for each code bit node in the factor graph, and a set of constraints for each check node. Experiments have shown that the relaxation is more useful for LDPC codes than for higher-density codes, hence the title of the chapter. Experiments on LDPC codes show that the performance of the LP decoder is better than the iterative min-sum algorithm, a standard message-passing algorithm used in practice. In addition, the LP decoder has the ML certificate property; none of the standard message-passing methods are known to have this desirable property on LDPC codes.

We introduce a variety of techniques for analyzing the performance of the LP decoder. We give an exact combinatorial characterization of the conditions for LP decoding success, even in the presence of cycles in the factor graph. This characterization holds for any binary-input memoryless symmetric channel. We define the set of *pseudocodewords*, which is a superset of the set of codewords, and we prove that the LP decoder always finds the lowest cost pseudocodeword. Thus, the LP decoder succeeds if and only if the lowest cost pseudocodeword is actually the transmitted codeword.

We prove that the max-fractional distance of our polytope on any binary linear code with check degree at least three is at least exponential in the girth of the graph associated with that code. (The girth of a graph is the length of its shortest cycle.) Thus, given a graph with logarithmic girth (which are known to exist), the max-fractional distance is at least $\Omega(n^{1-\epsilon})$, for some constant ϵ, where n is the code length. This shows that LP decoders can correct $\Omega(n^{1-\epsilon})$ errors in the binary symmetric channel.

We also discuss various generic techniques for *tightening* our LP relaxation in order to obtain a better decoder. One method of tightening is to add redundant parity checks to the factor graph, thus obtaining more constraints for the polytope (without cutting off any codewords). Another method is to use generic LP tightening techniques (e.g., [LS91, SA90]); we discuss the "lift-and-project" [LS91] technique, as it applies to our decoding polytope.

The results in this chapter are joint work with David Karger and Martin Wainwright. Most of this work has appeared in conference form [FWK03a], or has been submitted for journal publication [FWK03b].

Error Thresholds for Large Block Lengths. The techniques used by Chung et al. [CFRU01] are similar to those of Richardson and Urbanke [RU01] and Luby et. al. [LMSS98], who give an algorithm to calculate the *threshold* of a randomly constructed LDPC code. This threshold acts as a limit on the channel noise; if the noise is below the threshold, then reliable decoding (using belief propagation) can be achieved (with high probability) by a random code as the block length goes to infinity.

The threshold analysis is based on the idea of considering an "ensemble" of codes for the purposes of analysis, then averaging the behavior of this ensemble as the block length of the code grows. It has been shown [RU01] that for certain ensembles, as long as the error parameter of the channel is under the threshold, any word error rate is achievable by the ensemble average with large enough block length. For many ensembles, it is known [RU01] that for any constant ϵ, the difference in error rate (under belief-propagation decoding) between a random code and the average code in the ensemble is less than ϵ with probability exponentially small in the block length.

Calculating the error rate of the ensemble average can become difficult when the belief network contains cycles; because message-passing algorithms can traverse cycles repeatedly, noise in the channel can affect the final decision in complicated, highly dependent ways. This complication is avoided by fixing the number of iterations of the decoder, then letting the block length grow; then, the probability of a cycle in the belief network goes to zero. However, this sometimes requires prohibitively large block lengths [RU01], whereas smaller block lengths are desirable in practice [DPR+02].

Therefore, it is valuable to examine the behavior of a code ensemble at fixed block lengths, and try to analyze the effect of cycles. The finite length analysis of LDPC codes under the binary erasure channel (BEC) was taken on by Di et. al [DPR+02]. Key to their results is the notion of a purely combinatorial structure known as a *stopping set*. Belief propagation fails if and only if a stopping set exists among the erased bits; therefore the error rate of belief propagation is reduced to a purely combinatorial question.

With LP decoding we provide a similar combinatorial characterization through the notion of a pseudocodeword. In fact, in the BEC, stopping sets and pseudocodewords are equivalent (we prove this in the next chapter). However, in contrast to stopping sets, LP pseudocodewords for LDPC codes are defined over arbitrary binary-input memoryless symmetric channels.

Chapter Outline. In Section 5.1, we give our LP relaxation using the factor graph, and give intuition on the structure of fractional solutions. We also show that the relaxation is proper, and that it is $\mathcal{C}$-symmetric for any binary linear code $\mathcal{C}$. In Section 5.2 we define the notion of a pseudocodeword for this relaxation, which is a combinatorial way to view the success conditions of the decoder. This will be useful for proving our fractional distance bound, and in later chapters for comparing the performance of LP decoding with message-passing algorithms. We prove our bound on the fractional distance of the polytope in Section 5.3. In Section 5.4 we discuss generic methods for tightening the relaxation, including adding extra parity check constraints and using the "lift-and-project" method. The relaxation we give in Section 5.1 will only have a polynomial-sized description when the code is an LDPC code; we remedy this situation is Section 5.5 by providing a polynomial-sized description of an equivalent polytope. In Section 5.6, we include a proof deferred from Section 5.1.

5.1 An LP relaxation on the Factor Graph

Recall that a *factor graph* is an undirected bipartite graph, with *variable* nodes for each code bit, and *check* nodes representing local parity check constraints. (See Section 2.7.3 for background.) Our LP has a variable for each variable node, and a set of linear constraints for each check node. The constraints for a check node affect only the code bit variables for the nodes in its neighborhood. We will also introduce *auxiliary* LP variables to enforce the check node constraints. These variables will not play a role in the objective function, but will simplify the presentation and analysis of the relaxation.

We motivate our LP relaxation with the following observation. Each check node in a factor graph defines a local code; i.e., the set of binary vectors of even weight on its neighborhood variables. The global code corresponds to the intersection of all the local codes. In LP terminology, each check node defines a local codeword polytope (the set of convex combinations of local codes), and our global relaxation polytope will be the intersection of all of these polytopes.

5.1.1 The Polytope $\mathcal{Q}$

In this section we formally define the polytope $\mathcal{Q}$ that we use for the remainder of the chapter. The polytope has variables $(f_1, \ldots, f_n)$ to denote the code bits. Naturally, we have:

$$\forall i \in \mathcal{I}, \qquad 0 \leq f_i \leq 1 \tag{5.1}$$

To define a local codeword polytope, we consider the set of variable nodes $N(j)$ that are neighbors of a particular check node $j \in \mathcal{J}$. Of interest are subsets $S \subseteq N(j)$ that contain an even number of variable nodes; each such subset corresponds to a local codeword set, defined by setting $y_i = 1$ for each index $i \in S$, $y_i = 0$ for each $i \in N(j)$ but $i \notin S$, and setting all other y_i arbitrarily.

For each S in the set $E_j = \{S \subseteq N(j) : |S| \text{ even}\}$, we introduce an auxiliary LP variable $w_{j,S}$, which is an indicator for the local codeword associated with S. Note that the variable $w_{j,\emptyset}$ is also present for each parity check, and represents setting all variables in $N(j)$ equal to zero.

As indicator variables, the variables $\{w_{j,S}\}$ must satisfy the constraints:

$$\forall S \in E_j, \qquad 0 \leq w_{j,S} \leq 1. \tag{5.2}$$

The variable $w_{j,S}$ can also be seen as indicating that the codeword "satisfies" check j using the configuration S. In a codeword, each parity check is satisfied with one particular even-sized subset of nodes in its neighborhood set to one. Therefore, we may enforce:

$$\sum_{S \in E_j} w_{j,S} = 1. \tag{5.3}$$

Finally, the indicator f_i at each variable node i must be consistent with the point in the local codeword polytope defined by w for check node j. This leads to the constraint:

$$\forall\ i \in N(j), \qquad f_i = \sum_{\substack{S \in E_j \\ S \ni i}} w_{j,S}. \tag{5.4}$$

We define, for all $j \in \mathcal{J}$, the polytope $\mathcal{Q}_j$ as follows:

$$\mathcal{Q}_j = \{(f, w) : \text{equations } (5.1), (5.2), (5.3) \text{ and } (5.4) \text{ hold}\}$$

Let $\mathcal{Q} = \cap_j \mathcal{Q}_j$ be the intersection of these polytopes; i.e., the set of points (f, w) such that equations (5.1), (5.2), (5.3) and (5.4) hold for *all* $j \in \mathcal{J}$. Overall, the LCLP relaxation corresponds to the problem:

$$\text{minimize} \quad \sum_{i=1}^{n} \gamma_i f_i \quad \text{s.t.} \quad (f, w) \in \mathcal{Q} \tag{5.5}$$

5.1.2 Fractional Solutions to the Polytope $\mathcal{Q}$

Given a cycle-free factor graph, it can be shown that any optimal solution to LCLP is integral. Therefore LCLP is an exact formulation of the ML decoding problem in the cycle-free case. In contrast, for a factor graph with cycles, the optimal solution to LCLP may *not* be integral.

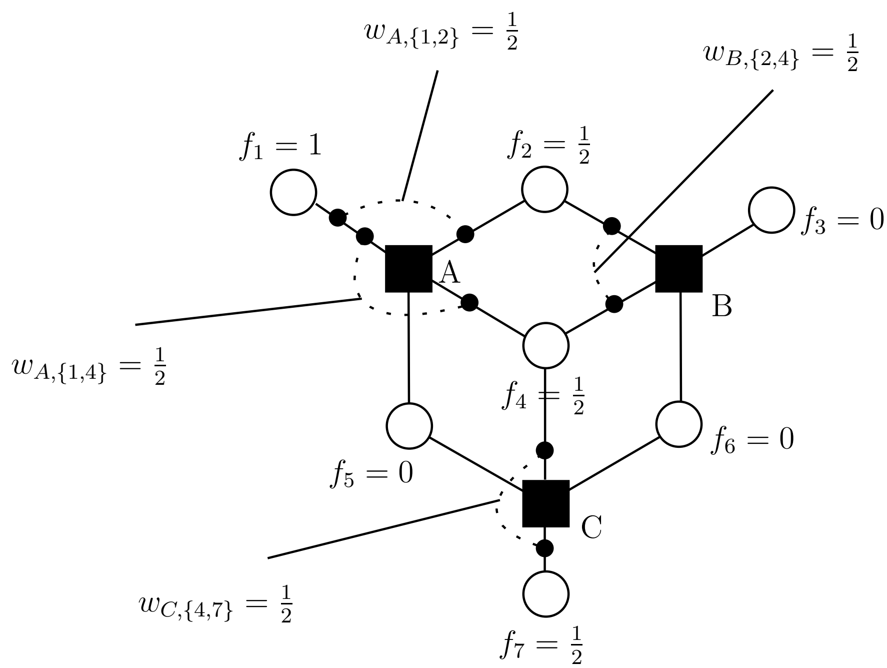


Figure 5-1: A fractional solution $f = [1, 1/2, 0, 1/2, 0, 0, 1/2]$ to the LP for a factor graph of the (7,4,3) Hamming code. For check node A, we have $w_{A,\{1,2\}} = w_{A,\{1,4\}} = 1/2$. Check node B has $w_{B,\{2,4\}} = w_{B,\emptyset} = 1/2$. Check node C has $w_{C,\{4,7\}} = w_{C,\emptyset} = 1/2$. This fractional solution has cost $-1/4$ under the cost function $\gamma = [-7/4, 1, 1, 1, 1, 1, 1]$, whereas every integral solution (codeword) has non-negative cost.

Take, for example, the Hamming code in Figure 5-1 (also in Figure 2-2). Suppose that we define a cost vector γ as follows: for variable node 1, set $\gamma_1 = -7/4$, and for all other nodes $\{2, 3, 4, 5, 6, 7\}$, set $\gamma_i = +1$. It is not hard to verify that under this cost function, all codewords have non-negative cost: any codeword with negative cost would have to set $y_1 = 1$, and therefore set at least two other $y_i = 1$, for a total cost of at least $+1/4$. Consider, however, the following fractional solution to LCLP: first, set $f = [1, 1/2, 0, 1/2, 0, 0, 1/2]$ and then for check node A, set $w_{A,\{1,2\}} = w_{A,\{1,4\}} = 1/2$; at check node B, assign $w_{B,\{2,4\}} = w_{B,\emptyset} = 1/2$; and lastly at check node C, set $w_{C,\{4,7\}} = w_{C,\emptyset} = 1/2$. It can be verified that (f, w) satisfies all of the LCLP constraints. However, the cost of this solution is $-1/4$, which is strictly less than the cost of any codeword.

Note that this solution is not a convex combination of codewords. This solution exploits the local perspective of the relaxation: check node B is satisfied by using the configuration $\{2, 4\}$, whereas in check node A, the configuration $\{2, 4\}$ is not used.

The analysis to follow will provide further insight into the nature of such fractional (i.e., non-integral) solutions to LCLP.

We note that the local codeword constraints (5.4) are analogous to those enforced in the Bethe formulation of belief propagation [YFW02].

5.1.3 Polytope Representation

The polytope $\mathcal{Q}$ contains auxiliary variables, and so we cannot simply plug in the generic results on LP decoding from Chapter 4. However, consider the projection $\dot{\mathcal{Q}}$ of the polytope $\mathcal{Q}$ onto the subspace defined by the $\{f_i\}$ variables; i.e.,

$$\dot{\mathcal{Q}} = \{f : \exists w \text{ s.t. } (f, w) \in \mathcal{Q}\}.$$

Since the objective function of LCLP only involves the $\{f_i\}$ variables, optimizing over $\dot{\mathcal{Q}}$ and $\mathcal{Q}$ will produce the same result. Furthermore, this polytope sits in the space $[0,1]^n$, and so fits into the paradigm of Chapter 4. In this section we give an explicit description of the projection $\dot{\mathcal{Q}}$. This will be useful for analysis, and also gives a more concrete perspective on the polytope $\mathcal{Q}$.

An Explicit Description of the Projected Polytope. In this section we derive an explicit description of the polytope $\dot{\mathcal{Q}}$. The following definition of $\dot{\mathcal{Q}}$ in terms of constraints on f was derived from the *parity polytope* of Jeroslow [Jer75, Yan91]. We first enforce $0 \leq f_i \leq 1$ for all $i \in \mathcal{I}$. Then, for every check j, we explicitly forbid every *bad* configuration of the neighborhood of j. Specifically, we require that for all $j \in \mathcal{J}$, for all $S \subseteq N(j)$ such that $|S|$ *odd*,

$$\sum_{i \in S} f_i \; + \sum_{i \in (N(j) \setminus S)} (1 - f_i) \; \leq \; |N(j)| - 1. \tag{5.6}$$

Let Ω_j be the set of points f that satisfy the constraints (5.6) for a particular check j and all $S \in N(j)$ where $|S|$ odd. Let the polytope $\Omega = \cap_{j \in \mathcal{J}} \Omega_j$, i.e., the set of points in $[0,1]^n$ that satisfy equation (5.6) for all $j \in \mathcal{J}$ and $S \in N(j)$ where $|S|$ odd. Let $\dot{\mathcal{Q}}_j = \{f : \exists w \text{ s.t.} (f, w) \in \mathcal{Q}_j\}$. In other words, $\dot{\mathcal{Q}}_j$ is the convex hull of local codeword sets defined by sets $S \in E_j$.

Theorem 5.1 *The polytopes Ω and $\mathcal{Q}$ are equivalent. In other words, $\Omega = \dot{\mathcal{Q}} = \{f : \exists w \text{ s.t. } (f, w) \in \mathcal{Q}\}$.*

Proof: It suffices to show $\Omega_j = \dot{\mathcal{Q}}_j$ for all $j \in \mathcal{J}$, since $\dot{\mathcal{Q}} = \cap_{j \in \mathcal{J}} \dot{\mathcal{Q}}_j$ and $\Omega = \cap_{j \in \mathcal{J}} \Omega_j$. This is shown by Jeroslow [Jer75]. For completeness, we include a proof of this fact in Section 5.6. ■

Here we offer some motivation for understanding the constraints in Ω_j. We can rewrite equation 5.21 as follows:

$$\sum_{i \in (N(j) \setminus S)} f_i + \sum_{i \in S} (1 - f_i) \geq 1. \tag{5.7}$$

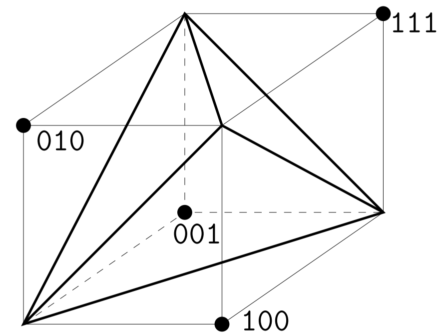


Figure 5-2: The equivalence of the polytopes Ω_j and $\dot{\mathcal{Q}}_j$ in three dimensions. The polytope Ω_j is defined as the set of points inside the unit hypercube with l_1 distance at least one from all odd-weight hypercube vertices. The polytope $\dot{\mathcal{Q}}_j$ is the convex hull of even-weight hypercube vertices.

In other words, the l_1 distance between (the relevant portion of) f and and the incidence vector for set S is at least one. This constraint makes sure that f is far away from the illegal configuration S. In three dimensions (i.e, $|N(j)| = 3$), it is easy to see that these constraints are equivalent to the convex hull of the even-sized subsets $S \in E_j$, as shown in Figure 5-2.

5.1.4 The properness of the polytope

Recall that a coding polytope is *proper* if there is a one-to-one correspondence between integral points and codewords. In this section we show that $\dot{\mathcal{Q}}$ is proper, thus showing that LP decoding with $\mathcal{Q}$ has the ML certificate property.

Lemma 5.2 *The polytope $\dot{\mathcal{Q}}$ for code $\mathcal{C}$ is proper; i.e., $\dot{\mathcal{Q}} \cap \{0,1\}^n = \mathcal{C}$.*

Proof: First we show that every codeword is contained within $\dot{\mathcal{Q}} = \Omega$. Suppose some codeword y is not in Ω, and so does not satisfy equation (5.6). Then, for some $j \in \mathcal{J}$, $S \subseteq N(j)$ where $|S|$ odd, we have

$$\sum_{i \in S} y_i + \sum_{i \in (N(j) \setminus S)} (1 - y_i) > |N(j)| - 1.$$

Since all $y_i \in \{0,1\}$, we have

$$\sum_{i \in S} y_i + \sum_{i \in (N(j) \setminus S)} (1 - y_i) = |N(j)|.$$

We may conclude that y_i is exactly the incidence vector for the set S. However, S has odd size, contradicting the fact that y is a codeword.

Now we show that every integral point of Ω is a codeword. Let $f \in (\Omega \cap \{0,1\}^n)$ be an arbitrary integral point of Ω. Suppose f is not a codeword. Then, for some parity check j, it must be that $|S = \{i : i \in N(j), f_i = 1\}|$ is odd. However, this implies that

$$\sum_{i \in S} f_i + \sum_{i \in (N(j) \setminus S)} (1 - f_i) = |N(j)|,$$

contradicting the fact that $f \in \Omega$. ■

Recall that the LP decoding algorithm based on polytope $\mathcal{Q}$ consists of the following steps. We first solve the LP to obtain (f^*, w^*). If $f^* \in \{0,1\}^n$, we output it as the ML codeword; otherwise, f^* is fractional, and we output "error." From Lemma 5.2, we get the following:

Proposition 5.3 *LP decoding using polytope $\mathcal{Q}$ has the* ML certificate *property: if the algorithm outputs a codeword, it is guaranteed to be the ML codeword.*

Proof: If the algorithm outputs a codeword y, then (y, w^*) has cost less than or equal to all points in $\mathcal{Q}$, and so y has cost less than or equal to all points in $\dot{\mathcal{Q}}$. For some codeword $y' \neq y$, we have that y' is a point in $\dot{\mathcal{Q}}$ by Lemma 5.2. Therefore y has cost less than or equal to y'. (Recall that the minimum-cost codeword is the ML codeword.) ■

5.1.5 The $\mathcal{C}$-symmetry of the polytope

In this section, we prove that the projected polytope $\dot{\mathcal{Q}}$ is $\mathcal{C}$-symmetric for any binary linear code $\mathcal{C}$. Using the generic results from Chapter 4, this implies that we can make the all-zeros assumption when analyzing the polytope $\mathcal{Q}$, and that the fractional distance of $\dot{\mathcal{Q}}$ is equal to the minimum weight of a vertex in $\dot{\mathcal{Q}}$.

Theorem 5.4 *The polytope $\dot{\mathcal{Q}}$ is $\mathcal{C}$-symmetric for any binary linear code $\mathcal{C}$.*

Proof: Fix an arbitrary point $f \in \dot{\mathcal{Q}}$ and codeword $y \in \mathcal{C}$. Recall that $f^{[y]}$ is the relative point to y, where $f_i^{[y]} = |f_i - y_i|$ for all $i \in \mathcal{I}$. We must show that $f^{[y]} \in \dot{\mathcal{Q}}$. To show this, we will use the fact that $\Omega = \dot{\mathcal{Q}}$.

We must show that $f^{[y]}$ obeys the odd-set constraints (5.6) of Ω. Let $j \in \mathcal{J}$ be some arbitrary check node, and let S be some arbitrary odd-sized subset of $N(j)$. We use the operator $\oplus$ to denote the symmetric difference of two sets. Let

$$S' = S \oplus \{i \in N(j) : y_i = 1\}.$$

Since y is a codeword, we have $|\{i \in N(j) : y_i = 1\}|$ even, and so $|S'|$ is odd. In the remainder of the proof, it is assumed that all values i that we sum over are contained in the set $N(j)$. From the fact that $f \in \Omega$, we have

$$\sum_{i \in S'} f_i + \sum_{i \notin S'} (1 - f_i) \leq |N(j)| - 1$$

We separate the sums above by values of y_i to obtain

$$\sum_{i \in S', y_i = 0} f_i \; + \sum_{i \in S', y_i = 1} f_i \; + \sum_{i \notin S', y_i = 0} (1 - f_i) \; + \sum_{i \notin S', y_i = 1} (1 - f_i) \; \leq \; |N(j)| - 1.$$

By the definition of S', we have

$$\sum_{i \in S, y_i = 0} f_i \; + \sum_{i \notin S, y_i = 1} f_i \; + \sum_{i \notin S, y_i = 0} (1 - f_i) \; + \sum_{i \in S, y_i = 1} (1 - f_i) \; \leq \; |N(j)| - 1.$$

By the definition of $f^{[y]}$, we have

$$\sum_{i \in S, y_i = 0} f_i^{[y]} \; + \sum_{i \notin S, y_i = 1} (1 - f_i^{[y]}) \; + \sum_{i \notin S, y_i = 0} (1 - f_i^{[y]}) \; + \sum_{i \in S, y_i = 1} f_i^{[y]} \; \leq \; |N(j)| - 1,$$

and therefore

$$\sum_{i \in S} f_i^{[y]} + \sum_{i \notin S} (1 - f_i^{[y]}) \leq |N(j)| - 1.$$

This is exactly the odd-set constraint for j and S. Since the check j and set S were chosen arbitrarily, we may conclude that $f^{[y]}$ obeys all odd-set constraints of Ω, and so $f^{[y]} \in \Omega$. ■

From this point forward in our analysis of the polytope $\mathcal{Q}$, we assume that the all-zeros codeword is the transmitted codeword. From Theorem 5.4, we can use Corollary 4.10 and obtain the following:

Corollary 5.5 *Given that the all-zeros codeword was transmitted (which we may assume by Theorem 5.4), the LP decoder using $\mathcal{Q}$ will fail if and only if there is some point in $(f, w) \in \mathcal{Q}$ with cost less than or equal to zero, where $f \neq 0^n$.*

5.2 Pseudocodewords

In this section, we introduce the concept of a pseudocodeword. A pseudocodeword is essentially a scaled and normalized version of a point in the polytope $\mathcal{Q}$. As a consequence, Corollary 5.5 will hold for pseudocodewords in the same way that it holds for points in $\mathcal{Q}$. In Chapter 7, we will use this notion to connect the success conditions of LP decoding with that of other message-passing algorithms for various special cases. We will also use pseudocodewords to derive our bound on fractional distance.

5.2.1 Definitions

The following definition of a codeword motivates the notion of a pseudocodeword. Recall that E_j is the set of even-sized subsets of the neighborhood of check node j. Let $E_j^- = E_j \setminus \{\emptyset\}$, the set of even-sized subsets not including the empty set. Let h be a vector in $\{0,1\}^n$, and let u be a setting of non-negative integer weights, one weight $u_{j,S}$ for each check j and $S \in E_j^-$. In a codeword, the vector h will represent the codeword, and the variable $u_{j,S}$ will be set to 1 if the codeword has the configuration S for check node j, and 0 otherwise.

We can enforce that h is a codeword with consistency constraints; for all edges (i,j) in the factor graph G, we have

$$h_i = \sum_{S \in E_j^-, S \ni i} u_{j,S}. \tag{5.8}$$

This corresponds exactly to the consistency constraints (5.4) in $\mathcal{Q}$. It is not difficult to see that the constraints guarantee that the binary vector h is always a codeword of the original code.

We obtain the definition of a *pseudocodeword* (h,u) by removing the restriction $h_i \in \{0,1\}$, and instead allowing each h_i to take on arbitrary non-negative *integer* values. In other words, a pseudocodeword is a vector $h = (h_1, \ldots, h_n)$ of non-negative integers such that, for every parity check $j \in \mathcal{J}$, the neighborhood $\{h_i : i \in N(j)\}$ is the sum of local codewords (incidence vectors of even-sized sets in E_j).

With this definition, any (global) codeword is (trivially) a pseudocodeword as well; moreover, any sum of codewords is a pseudocodeword. However, in general there exist pseudocodewords that *cannot* be decomposed into a sum of codewords. As an illustration, consider the fractional solution in Figure 5-1. If we simply scale this fractional solution by a factor of two, the result is a pseudocodeword (h,u) of the following form. We begin by setting $h = [2,1,0,1,0,0,1]$. To satisfy the constraints of a pseudocodeword, set $u_{A,\{1,2\}} = u_{A,\{1,4\}} = u_{B,\{2,4\}} = u_{C,\{4,7\}} = 1$. This pseudocodeword cannot be expressed as the sum of individual codewords.

Note that we do not have variables $u_{j,\emptyset}$. This is because pseudocodewords are normalized to the all-zeros codeword; the values on the variables $u_{j,S}$ represent the total weight of h on non-zero local configurations. Thus the all-zeros codeword is still a pseudocodeword, by setting all $w_{S,j} = 0$.

In the following, we use the fact that all optimum points of a linear program with rational coefficients are themselves rational. We can restate Corollary 5.5 in terms of pseudocodewords as follows:

Theorem 5.6 *Given that the all-zeros codeword was transmitted (which we may assume by Theorem 5.4), the LP decoder will fail if and only if there is some pseudocodeword (h,u), $h \neq 0^n$, where $\sum_i \gamma_i h_i \leq 0$.*

Proof: Suppose the decoder fails. Let (f,w) be the point in $\mathcal{Q}$ that minimizes $\sum_i \gamma_i f_i$. By Corollary 5.5, $\sum_i \gamma_i f_i \leq 0$. Construct a pseudocodeword (h,u) as follows. Let β be an integer such that βf_i is an integer for all bits i, and $\beta w_{j,S}$ is an integer for all for all checks j and sets $S \in E_j^-$. Such an integer exists because (f,w) is the optimal point of the LP, and all optimal points of an LP are rational [Sch87]. For all bits i, set $h_i = \beta f_i$; for all checks j and sets $S \in E_j^-$, set $u_{j,S} = \beta w_{j,S}$.

By the consistency constraints of $\mathcal{Q}$, we have that (h,u) meets the definition of a pseudocodeword. The cost of (h,u) is exactly $\beta \cdot \sum_i \gamma_i f_i$. Since $f \neq 0^n$, $\beta > 0$. This implies that $h \neq 0^n$. Since $\sum_i \gamma_i f_i \leq 0$ and $\beta > 0$, we see that $\sum_i \gamma_i h_i \leq 0$.

To establish the converse, suppose a pseudocodeword (h, u) where $h \neq 0^n$ has $\sum_i \gamma_i h_i \leq 0$. Let $\beta = \max_j (\sum_{S \in E_j} u_{j,S})$. We construct a point $(f, w) \in \mathcal{Q}$ as follows: Set $f_i = h_i/\beta$ for all code bits i. For all checks j, do the following:

(i) Set $w_{j,S} = u_{j,S}/\beta$ for all sets $S \in E_j^-$.

(ii) Set $w_{j,\emptyset} = 1 - \sum_{S \in E_j^-} w_{j,S}$.

We must handle $w_{j,\emptyset}$ as a special case since $u_{j,\emptyset}$ does not exist. By construction, and equation (5.8), the point (f, w) meets all the constraints of $\mathcal{Q}$. Since $h \neq 0^n$, we have $f \neq 0^n$. The cost of (f, w) is exactly $(1/\beta) \sum_i \gamma_i h_i$. Since $\sum_i \gamma_i h_i \leq 0$, the point (f, w) has cost less than or equal to zero. Therefore, by Corollary 5.5, the LP decoder fails. ∎

This theorem will be essential in proving the equivalence to message-passing decoding in the BEC in Section 7.3.

5.2.2 Pseudocodeword Graphs

A codeword y corresponds to a particular subgraph of the factor graph G. In particular, the vertex set of this subgraph consists of all the variable nodes $i \in \mathcal{I}$ for which $y_i = 1$, as well as all check nodes to which these variable nodes are incident. Any pseudocodeword can be associated with a graph H in an analogous way. In this section we define the notion of a *pseudocodeword graph* and give some examples.

In a pseudocodeword graph, we allow multiple copies of a node. Locally, a pseudocodeword graph looks identical to a codeword graph: variable nodes i must connect to exactly one copy of each check in $N(i)$, and check nodes j must connect to some even-sized configuration of the variable nodes in $N(j)$ (without connecting to multiple copies of the same node). Globally, however, the ability to use multiple copies of a node can allow for weight distributions that are not possible using only codewords. We see this in the examples at the end of the section.

This graphical characterization of a pseudocodeword is essential for proving our lower bound on the fractional distance. Additionally, the pseudocodeword graph is helpful in making connections with other notions of pseudocodewords in the literature. We discuss this further in Section 7.3.

Definition. The vertex set of the graph H for a pseudocodeword (h, u) consists of

- h_i copies of each variable node $i \in \mathcal{I}$, and
- $\sum_{S \in E_j^-} u_{j,S}$ copies of each check node $j \in \mathcal{J}$. Each copy of j is "labeled" with its corresponding set $S \in E_j^-$.

We refer to the set of h_i copies of the variable node i as $Y_i = \{[i, 1], [i, 2], \ldots, [i, h_i]\}$. We refer to the set of $u_{j,S}$ copies of the check node j with label S as $Z_{j,S} = \{[j, S, 1], [j, S, 2], \ldots, [j, S, u_{j,S}]\}$.

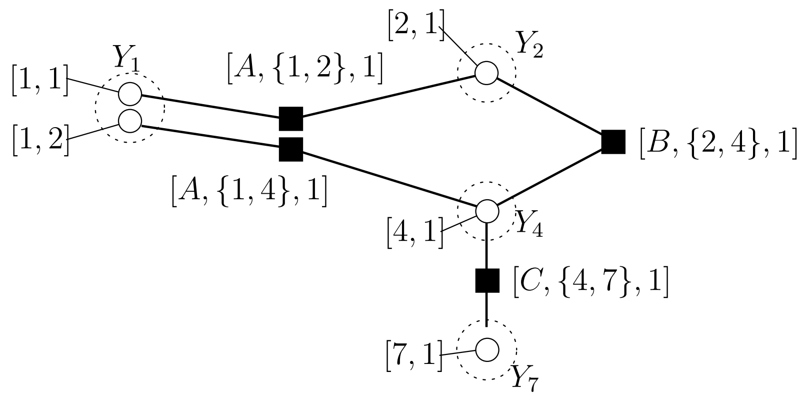


Figure 5-3: The graph of a pseudocodeword for the (7,4,3) Hamming code.

The edges of the graph are connected according to membership in the sets S. More precisely, consider an edge (i, j) in G. There are h_i copies of node i in H, i.e., $|Y_i| = h_i$. Now define $Z_j^{\ni i}$ as the set of nodes in H that are copies of check node j labeled with sets S that include i. In other words, we have

$$Z_j^{\ni i} = \bigcup_{S \in E_j^-, S \ni i} Z_{j,S}.$$

By the definition of a pseudocodeword,

$$h_i = \sum_{S \in E_j^-, S \ni i} u_{j,S},$$

and so $|Z_j^{\ni i}| = h_i = |Y_i|$. In the pseudocodeword graph H, connect the same-sized node sets $Z_j^{\ni i}$ and Y_i using an arbitrary matching (one-to-one correspondence). This process is repeated for every edge (i, j) in G.

Note that in a pseudocodeword graph, every check node in $Z_{j,S}$ appears in exactly $|S|$ sets $Z_j^{\ni i}$, one for each $i \in S$. Therefore, the neighbor set of any node in $Z_{j,S}$ consists of exactly one copy of each variable node $i \in S$. Furthermore, every variable node in Y_i will be connected to exactly one copy of each check node j in $N(i)$.

Examples. As a first example, we will walk through the construction of the pseudocodeword graph in Figure 5-3, which is the graph for the pseudocodeword $h = [2, 1, 0, 1, 0, 0, 1]$, where $u_{A,\{1,2\}} = u_{A,\{1,4\}} = u_{B,\{2,4\}} = u_{C,\{4,7\}} = 1$. We have $h_1 = 2$ copies of node 1, which make up the set $Y_1 = \{[1, 1], [1, 2]\}$; we also have 1 copy of each of the nodes 2,4 and 7, making up the sets $Y_2 = \{[2, 1]\}$, $Y_4 = \{[4, 1]\}$ and $Y_7 = \{[7, 1]\}$. We have 2 copies of check node A, one with label $\{1, 2\}$, one with label $\{1, 4\}$. We have one copy of check nodes B and C, with labels $\{2, 4\}$ and $\{4, 7\}$ respectively. Note that $Z_{A,\{1,2\}} = \{[A, \{1, 2\}, 1]\}$ and $Z_{A,\{1,4\}} = \{[A, \{1, 4\}, 1]\}$, and so the set $Z_A^{\ni 1} = \cup_{S \in E_A^-, S \ni 1} Z_{A,S} = \{[A, \{1, 2\}, 1], [A, \{1, 4\}, 1]\}$. We have $|Y_1| = |Z_A^{\ni 1}|$, and we make an arbitrary matching between these node sets, as shown in the figure.

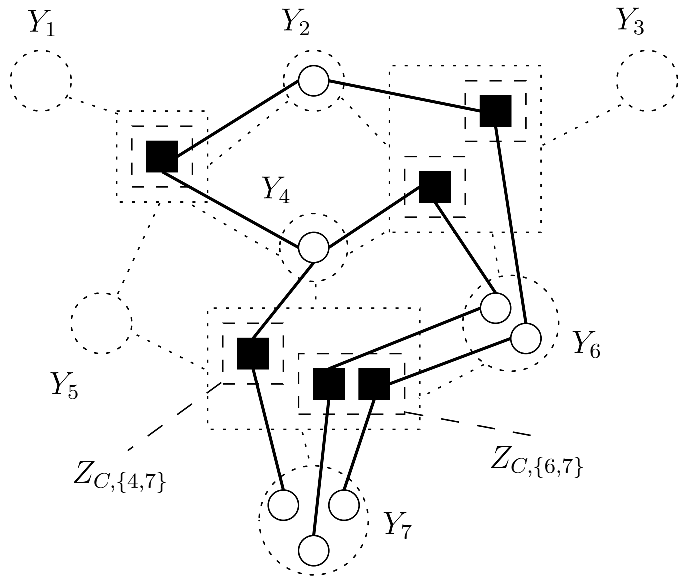


Figure 5-4: A graph H of the pseudocodeword $[0, 1, 0, 1, 0, 2, 3]$ of the (7,4,3) Hamming code. The dotted circles show the original variable nodes i of the factor graph G, which are now sets Y_i of nodes in H. The dotted squares are original check nodes j in G, and contain sets $Z_{j,S}$ (shown with dashed lines) for each $S \in E_j$.

We see another example from Figure 5-4, where we have $h = [0, 1, 0, 1, 0, 2, 3]$. Consider the variable node 7, and the check node C. Since $h_7 = 3$, we have three copies of variable node 3, i.e., $Y_7 = \{[7, 1], [7, 2], [7, 3]\}$. For check node C, we have $u_{C,\{4,7\}} = 1$ and $u_{C,\{6,7\}} = 2$. Therefore, we have 1 copy of C labeled with $\{4, 7\}$ (i.e., $Z_{C,\{4,7\}} = \{[C, \{4, 7\}, 1]\}$), and two copies of C labeled with $\{6, 7\}$ (i.e., $Z_{C,\{6,7\}} = \{[C, \{6, 7\}, 1], [C, \{6, 7\}, 1]\}$). These three copies of C are separated into two groups (using dotted squares in the figure) according to membership in sets $Z_{C,\{4,7\}}$ and $Z_{C,\{6,7\}}$.

We have $Z_C^{\ni 7} = \{[C, \{4, 7\}, 1], [C, \{6, 7\}, 1], [C, \{6, 7\}, 1]\}$, and so we see that $|Y_7| = |Z_C^{\ni 7}|$. An arbitrary matching is made between node sets Y_7 and $Z_C^{\ni 7}$.

Cost. The cost of the pseudocodeword graph is the sum of the costs γ_i of the variable nodes in the graph; if there are multiple copies of a variable node, then each contributes to the cost of the pseudocodeword. The cost of the pseudocodeword graph is equal to the cost of the pseudocodeword from which it was derived. Therefore, Theorem 5.6 holds for pseudocodeword graphs as well.

5.3 Fractional Distance

Recall that the fractional distance of a proper $\mathcal{C}$-symmetric polytope for the code $\mathcal{C}$ is the minimum weight of any non-zero vertex of the polytope. All codewords are non-zero vertices of the polytope, so the fractional distance is a lower bound on the true distance.

For a point f in $\dot{\mathcal{Q}}$, recall that the weight of f is defined as $\sum_i f_i$. Let $\mathcal{V}^-(\dot{\mathcal{Q}})$ be the set of non-zero vertices of $\dot{\mathcal{Q}}$. Since $\dot{\mathcal{Q}}$ is $\mathcal{C}$-symmetric, the fractional distance of $\dot{\mathcal{Q}}$ is equal to the minimum weight of any vertex in $\mathcal{V}^-(\dot{\mathcal{Q}})$, by Theorem 4.7. The fractional distance d_{frac} is always a lower bound on the classical distance of the code, since every non-zero codeword is contained in $\mathcal{V}^-(\dot{\mathcal{Q}})$. Moreover, using Theorem 4.2, we have that LP decoding can correct up to $\lceil d_{frac}/2 \rceil - 1$ errors in the BSC.

5.3.1 Computing the Fractional Distance

In order to run the procedure outlined in Section 4.4.3, we use the small explicit representation Ω of $\dot{\mathcal{Q}}$ given by Theorem 5.1. The number of constraints in Ω has an exponential dependence on the check degree of the code. For an LDPC code, the number of constraints will be linear in n, so that we can compute the exact fractional distance efficiently. For arbitrary linear codes, we can still compute the minimum weight non-zero vertex of the polytope $\mathcal{R}$ (covered in Section 5.5), which provides a (possibly weaker) lower bound on the fractional distance. However, this polytope introduces many auxiliary variables, and may have many "false" vertices with low weight.

5.3.2 Experiments

Figure 5-5 gives the average fractional distance of a randomly chosen LDPC factor graph, computed using the algorithm described in Section 4.4.3. The graph has left degree 3, right degree 4, and is randomly chosen from an ensemble described by Sipser and Spielman [SS96]. This data is insufficient to extrapolate the growth rate of the fractional distance; however it certainly grows non-trivially with the block length. We conjecture that this growth rate is linear in the block length.

Figure 5-6 gives the fractional distance of the "normal realizations" of the Reed-Muller($n-1$, n) codes [For01][1]. These codes, well-defined for lengths n equal to a power of 2, have a classical distance of exactly $n/2$. The curve in the figure suggests that the fractional distance of these graphs is roughly $\frac{5}{14}n^{0.7}$. Note that for both these code families, there may be alternate realizations (factor graphs) with better fractional distance.

Although we have shown that the fractional distance is related to the worst-case error event for LP decoding, it would be interesting to see if the fractional distance is a good overall predictor of performance, especially since the fractional distance can be computed. In Figure 5-7, we show that the fractional distance does predict somewhat the performance of LP decoding. However, our data seem to show that the fractional distance is not a good predictor of the performance of the standard message-passing sum-product decoder.

[1]We thank G. David Forney for suggesting the normal realizations of the Reed-Muller codes.

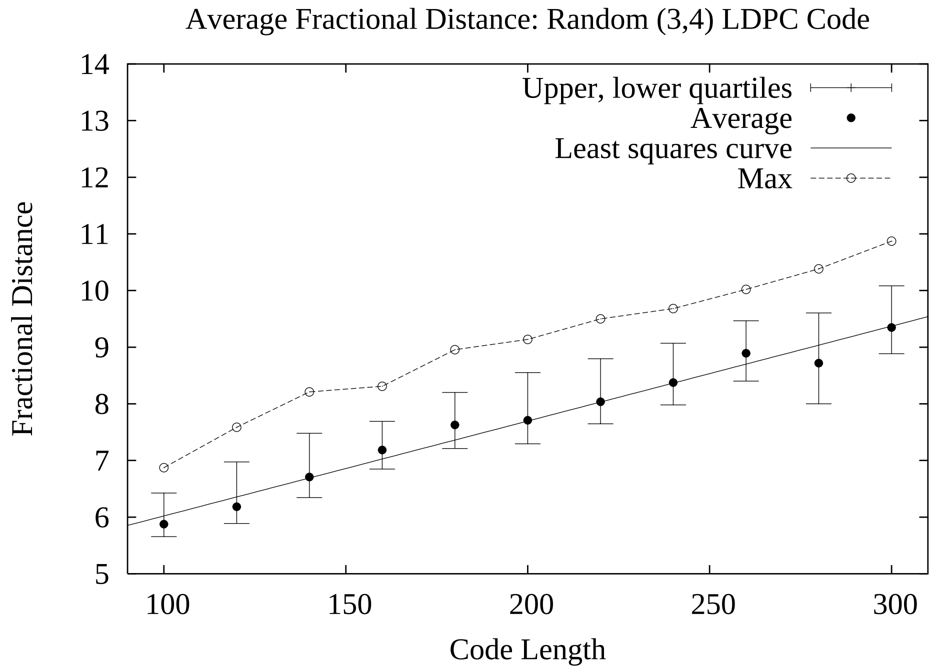


Figure 5-5: The average fractional distance d_{frac} as a function of length for a randomly generated LDPC code, with left degree 3, right degree 4. Also shown are the max, and the upper and lower quartiles, for 100 samples per block length.

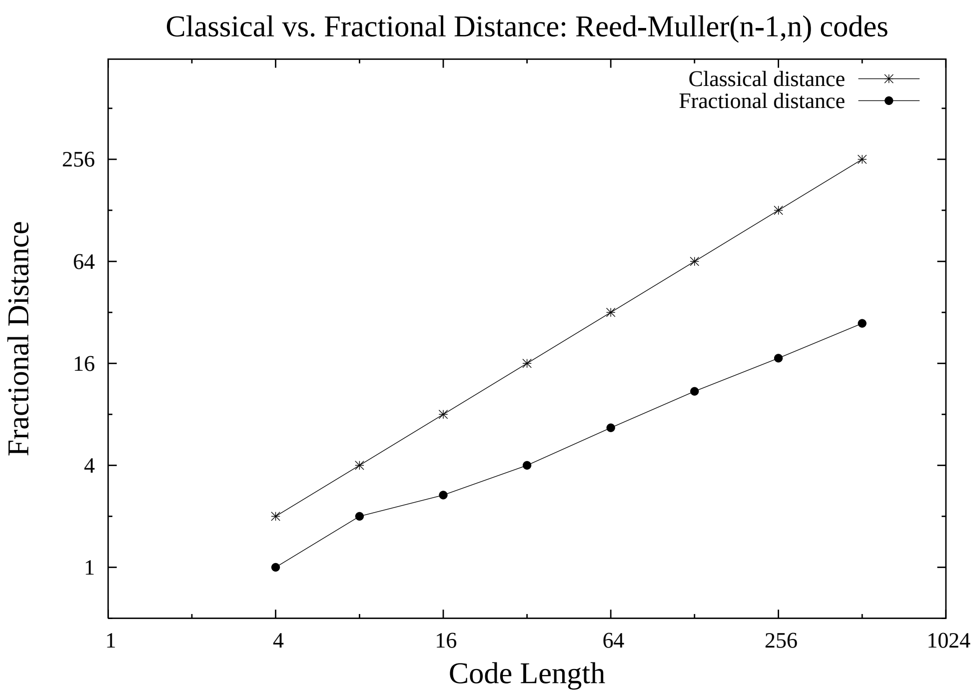


Figure 5-6: The classical vs. fractional distance of the "normal realizations" of the Reed-Muller($n-1$, n) codes [For01]. The classical distance of these codes is exactly $n/2$. The upper part of the fractional distance curve follows roughly $\frac{5}{14}n^{0.7}$.

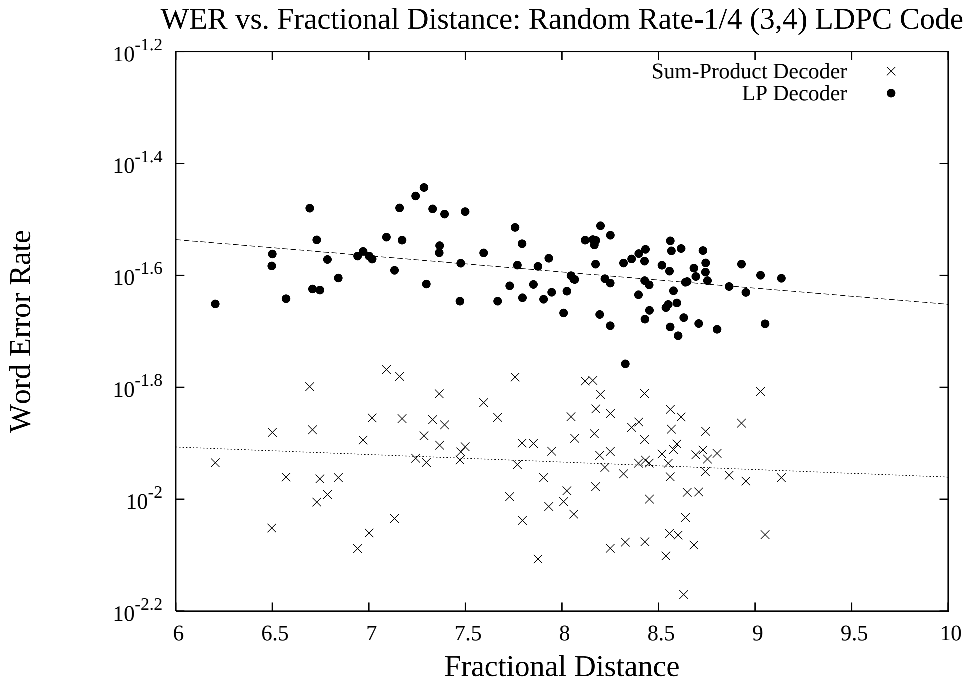


Figure 5-7: The word error rate (WER) as a function of the fractional distance of a random LDPC code, under both LP and sum-product decoding. Each of the 100 data points for each decoder represents a randomly chosen rate-1/4 (3,4) LDPC code of length 200. Three instances are not shown, nor considered in the curve fit (these had fractional distances of 2, 4 and 4, and very poor WER). The curves are fit to minimize the sum of the squared differences between the curve and the data points.

5.3.3 A Lower Bound Using the Girth

The following theorem asserts that the max-fractional distance of polytope $\dot{\mathcal{Q}}$ is exponential in the girth of G. It is analogous to an earlier result of Tanner [Tan81], which provides a similar bound on the classical distance of a code in terms of the girth of the associated factor graph.

Theorem 5.7 *Let G be a factor graph with $deg^-_\ell \geq 3$ and $deg^-_r \geq 2$. Let g be the girth of G, $g > 4$. Then the max-fractional distance of $\dot{\mathcal{Q}}$ is at least $d^{\max}_{frac} \geq (deg^-_\ell - 1)^{\lceil g/4 \rceil - 1}$.*

We prove this theorem shortly, making heavy use of the combinatorial properties of pseudocodewords. One consequence of Theorem 5.7 is that the max-fractional distance is at least $\Omega(n^{1-\epsilon})$ for some constant ϵ, for any graph G with girth $\Omega(\log n)$. Note that there are many known constructions of such graphs (e.g., [RV00]).

Theorem 5.7 demonstrates that LP decoding can correct $\Omega(n^{1-\epsilon})$ errors for any code defined by a graph with logarithmic girth. However, we need a linear lower bound on the fractional distance to yield a non-trivial bound on the WER. For example, consider the BSC with a constant crossover probability p. In expectation, we have a linear number of bits flipped by the channel; therefore a sub-linear bound on the fractional distance says nothing about the WER. On the other hand, even if the fractional distance is sub-linear, the code may still perform quite well, just as a code with low classical distance may still perform well. In RA(2) codes for example, even though the classical distance (and the fractional distance) are both logarithmic, the decoder still succeeds with high probability at low noise levels.

We note that for the polytope $\dot{\mathcal{Q}}$, we have $d_{frac}^{\max} \leq d_{frac}(deg_r^+/2)$ (proven below). It follows that for LDPC codes, where deg_r^+ is constant, we have that d_{frac} and $d_{frac}^{\max}$ for the polytope $\dot{\mathcal{Q}}$ are the same up to a constant factor.

Theorem 5.8 *If d_{frac} is the fractional distance of $\dot{\mathcal{Q}}$, and $d_{frac}^{\max}$ is the max-fractional distance of $\dot{\mathcal{Q}}$, then $d_{frac}^{\max} \leq d_{frac}(deg_r^+/2)$.*

Proof: Suppose not. Then, $d_{frac}^{\max} > d_{frac}(deg_r^+/2)$. It follows that there exists some vertex $f \in \mathcal{V}(\dot{\mathcal{Q}})$ such that

$$\sum_i f_i < \frac{2}{deg_r^+} d_{frac}^{\max}.$$

By the definition of $d_{frac}^{\max}$, it follows that

$$\sum_i f_i < \frac{2}{deg_r^+} \cdot \frac{\sum_i f_i}{\max_i f_i},$$

and so

$$\max_i f_i < \frac{2}{deg_r^+}.$$

Since $deg_r^+ = \max_{j \in \mathcal{J}} |N(j)|$, it follows that:

$$\forall\ j \in \mathcal{J}, \qquad \sum_{i \in N(j)} f_i < 2. \tag{5.9}$$

Set the variables $\{w_{j,S}\}_{j \in \mathcal{J}, S \in E_j}$ such that $(f, w) \in \mathcal{Q}$. From the constraints (5.4), it follows that for all $j \in \mathcal{J}$,

$$\sum_{i \in N(j)} f_i = \sum_{S \in E_j} |S| w_{j,S} \geq \sum_{\substack{S \in E_j \\ S \neq \emptyset}} 2 w_{j,S}.$$

The last step follows from the fact that for all $S \in E_j$ where $S \neq \emptyset$, we have $|S| \geq 2$. From the above equation, and equation (5.9), we have that for all $j \in \mathcal{J}$,

$$\sum_{\substack{S \in E_j \\ S \neq \emptyset}} w_{j,S} < 1.$$

Since $\sum_{S \in E_j} w_{j,S} = 1$ (by the constraints (5.3) of $\mathcal{Q}$), it follows that $w_{j,\emptyset} > 0$ for all $j \in \mathcal{J}$. This allows us to define a positive scaling factor $\epsilon > 0$ that is small enough to stay within $\mathcal{Q}$:

$$\epsilon = \frac{\min_{j \in \mathcal{J}} w_{j,\emptyset}}{2^{deg_r^+}}.$$

Construct a new $(f', w') \in \mathcal{Q}$ as follows. Set $f'_i = (1+\epsilon) f_i$ for all $i \in \mathcal{I}$, and set $w'_{j,S} = (1+\epsilon) w_{j,S}$ for all $j \in \mathcal{J}$ and $S \in (E_j \setminus \emptyset)$. Note that from the way we defined ϵ, we have, for all $j \in \mathcal{J}$

$$\sum_{\substack{S \in E_j, \\ S \neq \emptyset}} w'_{j,S} < 1.$$

Accordingly, we set

$$w'_{j,\emptyset} = 1 - \sum_{\substack{S \in E_j \\ S \neq \emptyset}} w'_{j,S}.$$

It follows that $0 \leq w_{j,\emptyset} \leq 1$ and $\sum_{S \in E_j} w'_{j,S} = 1$ for all $j \in \mathcal{J}$. Thus, the constraints (5.3) are met by (f', w'). From the constraint (5.4) on (f, w), it follows that for all $i \in \mathcal{I}$ and $j \in N(i)$,

$$f'_i = (1+\epsilon) f_i = (1+\epsilon) \sum_{\substack{S \in E_j \\ S \ni i}} w_{j,S} = \sum_{\substack{S \in E_j \\ S \ni i}} w'_{j,S}.$$

and so (f', w') meets the constraints (5.4) of $\mathcal{Q}$. We conclude that $(f', w') \in \mathcal{Q}$, and so $f' \in \dot{\mathcal{Q}}$.

To complete the proof, note that

$$f = \left(\frac{1}{1+\epsilon}\right) f' + \left(1 - \frac{1}{1+\epsilon}\right) 0^n.$$

Since $0^n \in \dot{\mathcal{Q}}$, it follows that f is a convex combination of two other distinct points in $\dot{\mathcal{Q}}$. Therefore f is not a vertex, a contradiction. ■

Proving Theorem 5.7. Before proving Theorem 5.7, we will prove a few useful facts about pseudocodewords and pseudocodeword graphs. For all the theorems in this section, let G be a factor graph with all variable nodes having degree at least deg_ℓ^-, where $deg_\ell^- \geq 3$ and all check nodes having degree at least deg_r^-, where $deg_r^- \geq 2$. Let g be the girth of G, $g > 4$. Let H be the graph of some arbitrary pseudocodeword (h, u) of G where $h \neq 0^n$.

We define a *promenade* Ψ to be a path $\Psi = (\phi_1, \phi_2, \ldots, \phi_{|\Psi|})$ in H that may repeat nodes and edges, but takes no U-turns; i.e., for all i where $0 \leq i \leq |\Psi| - 2$, we have $\phi_i \neq \phi_{i+2}$. We will also use Ψ to represent the set of nodes on the path Ψ (the particular use will be clear from context). Note that each ϕ_i could be a variable or a check node. These paths are similar to the *irreducible closed walk* of Wiberg [Wib96]. A *simple* path of a graph is one that does not repeat nodes.

Recall that Y_i consists of variable nodes in H that are all copies of the same variable node i in the factor graph G; similarly, the set $Z_{j,S}$ consists of check nodes in

H that are all copies of the same check node j in G. Accordingly, for some variable node ϕ in H, let $G(\phi)$ be the corresponding node in G; i.e., $(G(\phi) = i : \phi \in Y_i)$ if ϕ is a variable node, and $(G(\phi) = j : \phi \in Z_{j,S}$ for some $S \in E_j)$ if ϕ is a check node.

Lemma 5.9 *For all promenades $\Psi = (\phi_1, \phi_2, \ldots, \phi_{|\Psi|})$ where $|\Psi| < g$,*

- *Ψ is a simple path in H, and*
- *$G(\Psi) = (G(\phi_1), \ldots, G(\phi_{|\Psi|}))$ is a simple path in G.*

Proof: First note that $G(\Psi)$ is a valid path. This follows by construction, since if there is an edge (ϕ_i, ϕ_{i+1}) in H, there must be an edge $(G(\phi_i), G(\phi_{i+1}))$ in G. If the path $G(\Psi)$ is simple, then the path Ψ must be simple. So, it remains to show that $G(\Psi)$ is simple. This is true since the length of $G(\Psi)$ is less than the girth of the graph. ■

For the remainder of the section, suppose without loss of generality that $h_1 = \max_i h_i$. We have $Y_1 = ([1,1],[1,2],\ldots,[1,h_1])$ as the set of nodes in the pseudocodeword graph that are copies of the variable node 1. Note that g is even, since G is bipartite. For all $i \in \{1,\ldots,h_1\}$, let $\mathcal{T}_i$ be the set of nodes in H within distance $(g/2)-1$ of $[1,i]$; i.e., $\mathcal{T}_i$ is the set of nodes with a path in H of length at most $(g/2)-1$ from $[1,i]$.

Lemma 5.10 *The subgraph induced by the node set $\mathcal{T}_i$ is a tree.*

Proof: Suppose not. Then, for some node in ϕ in H, there are at least two different paths from $[1,i]$ to ϕ, each with length at most $(g/2)-1$. This implies a cycle in H of length less than g; a contradiction to Lemma 5.9. ■

Lemma 5.11 *The node subsets $(\mathcal{T}_1,\ldots,\mathcal{T}_{h_1})$ in H are all mutually disjoint.*

Proof: Suppose not; then, for some $i \neq i'$, $\mathcal{T}_i$ and $\mathcal{T}_{i'}$ share at least one vertex. Let ϕ be the vertex in $\mathcal{T}_i$ closest to the root $[1,i]$ that also appears in $\mathcal{T}_{i'}$. Now consider the path $\Psi = ([1,i],\ldots,\phi',\phi,\phi'',\ldots,[1,i'])$, where the subpath from $[1,i]$ to ϕ is the unique such path in the tree $\mathcal{T}_i$, and the subpath from ϕ to $[1,i']$ is the unique such path in the tree $\mathcal{T}_{i'}$.

To establish that the path Ψ is a promenade, we must show that Ψ has no U-turns. The subpaths $([1,i],\ldots,\phi',\phi)$ and $(\phi,\phi'',\ldots,[1,i'])$ are simple (since they are entirely within their respective trees), so the only possible U-turn is at the node ϕ; thus it remains to show that $\phi' \neq \phi''$. Since we chose ϕ to be the node in $\mathcal{T}_i$ closest to $[1,i]$ that also appears in $\mathcal{T}_{i'}$, the node ϕ' must not appear in $\mathcal{T}_{i'}$. From the fact that ϕ'' does appear in $\mathcal{T}_{i'}$, we conclude that $\phi' \neq \phi''$, and so Ψ is a promenade.

Since the trees all have depth $(g/2)-1$, the path Ψ must have length less that g. Therefore the path $G(\Psi)$ must be simple by Lemma 5.9. However, it is not, since node 1 appears twice in $G(\Psi)$, once at the beginning and once at the end. This is a contradiction. ■

Lemma 5.12 *The number of variable nodes in H is at least $h_1(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}$.*

Proof: Consider the node set $\mathcal{T}_i$. We will count the number of nodes on each "level" of the tree induced by $\mathcal{T}_i$. Each level ℓ consists of all the nodes at distance ℓ from $[1, i]$. Note that even levels contain variable nodes, and odd levels contain check nodes.

Consider a variable node ϕ on an even level. All variable nodes in H are incident to at least deg_ℓ^- other nodes, by the construction of H. Therefore, ϕ has at least $deg_\ell^- - 1$ children in the tree on the next level. Now consider a check node on an odd level; check nodes are each incident to at least two nodes, so this check node has at least one child on the next level.

Thus the tree expands by a factor of at least $deg_\ell^- - 1 \geq 2$ from an even to an odd level. From an odd to an even level, it may not expand, but it does not contract. The final level of the tree is level $(g/2) - 1$, and thus the final even level is level $2(\lceil g/4 \rceil - 1)$. By the expansion properties we just argued, this level (and therefore the tree $\mathcal{T}_i$) must contain at least $(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}$ variable nodes.

By Lemma 5.11, each tree is disjoint, so the number of variable nodes in H is at least $h_1(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}$. ■

We are now ready to prove Theorem 5.7. Recall that we assumed $h_1 = \max_i h_i$, and that $\mathcal{V}^-(\dot{\mathcal{Q}}) = \mathcal{V}(\dot{\mathcal{Q}}) \setminus \emptyset$.

Theorem 5.7 Let G be a factor graph with $deg_\ell^- \geq 3$ and $deg_r^- \geq 2$. Let g be the girth of G, $g > 4$. Then the max-fractional distance of $\dot{\mathcal{Q}}$ is at least $d_{frac}^{\max} \geq (deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}$.

Proof: Let f be an arbitrary vertex in $\mathcal{V}^-(\dot{\mathcal{Q}})$. Set the variables $\{w_{j,S}\}_{j \in \mathcal{J}, S \in E_j}$ such that $(f, w) \in \mathcal{Q}$. Construct a pseudocodeword (h, u) from (f, w) as in Lemma 5.6; i.e., let β be an integer such that βf_i is an integer for all bits i, and $\beta w_{j,S}$ is an integer for all for all checks j and sets $S \in E_j^-$. Such an integer exists because (f, w) is a vertex of $\mathcal{Q}$, and therefore rational [Sch87]. For all bits i, set $h_i = \beta f_i$; for all checks j and sets $S \in E_j^-$, set $u_{j,S} = \beta w_{j,S}$.

Let H be a graph of the pseudocodeword (h, u), as defined in Section 5.2.2. By Lemma 5.12, H has at least $(\max_i h_i)(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}$ variable nodes. Since the number of variable nodes is equal to $\sum_i h_i$, we have:

$$\sum_i h_i \geq (\max_i h_i)(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}. \tag{5.10}$$

Recall that $h_i = \beta f_i$. Substituting into equation (5.10), we have:

$$\beta \sum_i f_i \geq \beta(\max_i f_i)(deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}.$$

It follows that

$$\left(\frac{\sum_i f_i}{\max_i f_i} \right) \geq (deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}.$$

This argument holds for an arbitrary $f \in \mathcal{V}^-(\dot{\mathcal{Q}})$. Therefore

$$d_{frac}^{\max} = \min_{f \in \mathcal{V}^-(\dot{\mathcal{Q}})} \left(\frac{\sum_i f_i}{\max_i f_i} \right) \geq (deg_\ell^- - 1)^{\lceil g/4 \rceil - 1}.$$

■

5.4 Tighter relaxations

It is important to observe that LCLP has been defined with respect to a specific factor graph. Since a given code has many such representations, there are many possible LP relaxations, and some may perform better than others. The fractional distance of a code is also a function of the factor graph representation of the code. Fractional distance yields a lower bound on the true distance, and the quality of this bound could also be improved using different representations. In addition, there are various generic techniques for tightening LP relaxations that are of use in the decoding application. In this section we present possible strategies to tighten the LP relaxation, at the expense of a more complex polytope.

This area is largely unexplored; our goal in this section is to present some possible tightening strategies, and a few experimental results. Another unexplored area is to employ an adaptive strategy to decoding, using LP tightening methods. In other words, the decoder solves the LP; if a codeword is found, we are done. If not, the decoder adds an LP constraint to eliminate the the fractional vertex found, and repeats. We discuss this further in Chapter 9.

5.4.1 Redundant Parity Checks

Adding redundant parity checks to the factor graph, though not affecting the code, provides new constraints for the LP relaxation, and will in general strengthen it. For example, returning to the $(7, 4, 3)$ Hamming code of Figure 2-2, suppose we add a new check node whose neighborhood is $\{1, 3, 5, 6\}$. This parity check is redundant for the code, since it is simply the mod two sum of checks A and B. However, the linear constraints added by this check tighten the relaxation; in fact, they render our example pseudocodeword $f = [1, 1/2, 0, 1/2, 0, 0, 1/2]$ infeasible. Whereas redundant constraints may degrade the performance of BP decoding (due to the creation of small cycles), adding new constraints can only improve LP performance.

As an example, Figure 5-8 shows the performance improvement achieved by adding all "second-order" parity checks to a factor graph G. By second-order, we mean all parity checks that are the sum of two original parity checks. It turns out that the only effect of adding second-order parity checks is on 4-cycles in the original factor graph. (This is not hard to see.) This explains the meager performance gain in Figure 5-8.

It would be interesting to see a good strategy for adding redundant parity checks that would have a real effect on performance. A natural question is whether adding *all* redundant parity checks results in the codeword polytope poly$(\mathcal{C})$ (and thus an ML decoder). This would not imply $P = NP$, since there are an exponential number of unique redundant parity checks. However, this turns out not to be the case (Hadamard codes provide the counterexample).

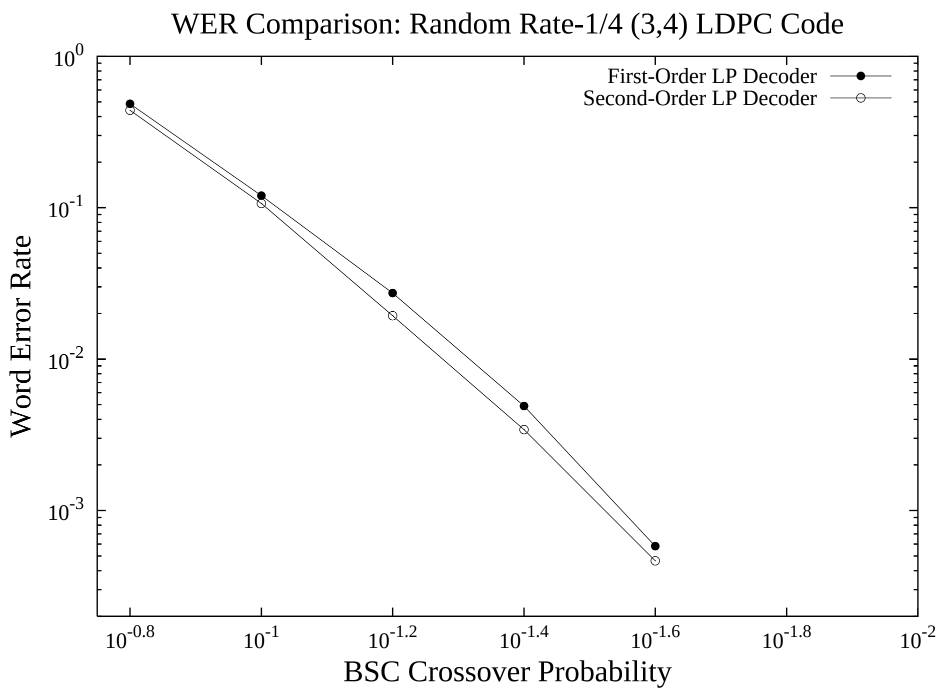


Figure 5-8: Error-correcting performance gained by adding a set of (redundant) parity checks to the factor graph. The code is a randomly selected regular LDPC code, with length 40, left degree 3 and right degree 4, from an ensemble of Gallager [Gal62]. The "First Order Decoder" is the LP decoder using the polytope $\mathcal{Q}$ defined on the original factor graph. The "Second Order Decoder" uses the polytope $\mathcal{Q}$ defined on the factor graph after adding a set of redundant parity checks; the set consists of all checks that are the sum (mod 2) of two original parity checks.

5.4.2 Lift and Project

In addition to redundant parity checks, there are various generic ways in which an LP relaxation can be strengthened (e.g., [LS91, SA90]). Such "lifting" techniques provide a nested sequence of relaxations increasing in both tightness and complexity, the last of which is equivalent to the convex hull of the codewords (albeit with exponential complexity). Therefore we obtain a sequence of decoders, increasing in both performance and complexity, the last of which is an (intractable) ML decoder. It would be interesting to analyze the rate of performance improvement along this sequence. Another interesting question is how complex a decoder is needed in order to surpass the performance of belief propagation.

Performing one round of "lift-and-project" [LS91] on the polytope $\mathcal{Q}$ results in a natural LP relaxation that can be explained independently. In this section we present this LP and give some experimental results illustrating the improvement gained over the "first-order" relaxation $\mathcal{Q}$.

The new relaxation will have n^2 variables f_{ij}, one for each $(i, j) \in \mathcal{I}^2$. The variable f_{ij} can be thought of as an indicator for setting *both* y_i and y_j equal to 1. Therefore, codewords y will correspond to integral solutions that have have $f_{ij} = y_i y_j$ for all $(i, j) \in \mathcal{I}^2$. (Note that $f_{ii} = y_i$ for all $i \in \mathcal{I}$.)

We can view a particular setting f of the variables as an $n \times n$ matrix F. For example, the following matrix represents the matrix F for the codeword `0111010` of the Hamming code in Figure 2-2:

$$\begin{pmatrix} \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & \mathbf{1} & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & \mathbf{1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{0} & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} \end{pmatrix}$$

Note that the codeword `0111010` sits on the diagonal of the matrix.

In the discussion below, we will argue that this matrix has certain properties. From these properties we will derive valid LP constraints that any integral solution must obey; since the codeword solutions obey these constraints, we may enforce them in our relaxation.

We describe the LP constraints in detail as follows. The variables are indicators, so we have

$$0 \leq f_{ij} \leq 1 \quad \text{for all} \quad (i, j) \in \mathcal{I}^2.$$

Note that in the matrix F, we have $F_{ij} = y_i y_j = F_{ji}$; so we may conclude that the matrix F is symmetric. Therefore we may enforce the constraints

$$f_{ij} = f_{ji} \quad \text{for all} \quad (i, j) \in \mathcal{I}^2.$$

The diagonal of F is the codeword y. Therefore, we may enforce the constraints of $\mathcal{Q}$ on the variables on the diagonal. We use the projected polytope $\dot{\mathcal{Q}}$ to simplify notation:

$$(f_{11}, f_{22}, \ldots, f_{nn}) \in \dot{\mathcal{Q}}$$

Now consider a row i of the matrix F. If $y_i = 0$, then this row will be all zeros. If $y_i = 1$, then this row will be exactly the codeword y. Since 0^n is a codeword, we may conclude that every row of F is a codeword. Thus, we may enforce the constraints of $\mathcal{Q}$ on each row:

$$(f_{1i}, f_{2i}, \ldots, f_{ni}) \in \dot{\mathcal{Q}} \quad \text{for all} \quad i \in \mathcal{I}$$

We also have that all columns of F are codewords; however from symmetry and the row constraints above, we already have the $\mathcal{Q}$ constraints enforced on each column.

Finally, consider the difference between the values on the diagonal of F and the values in a particular row i; i.e., the values $(F_{11} - F_{1i}, F_{22} - F_{2i}, \ldots, F_{nn} - F_{ni})$. If

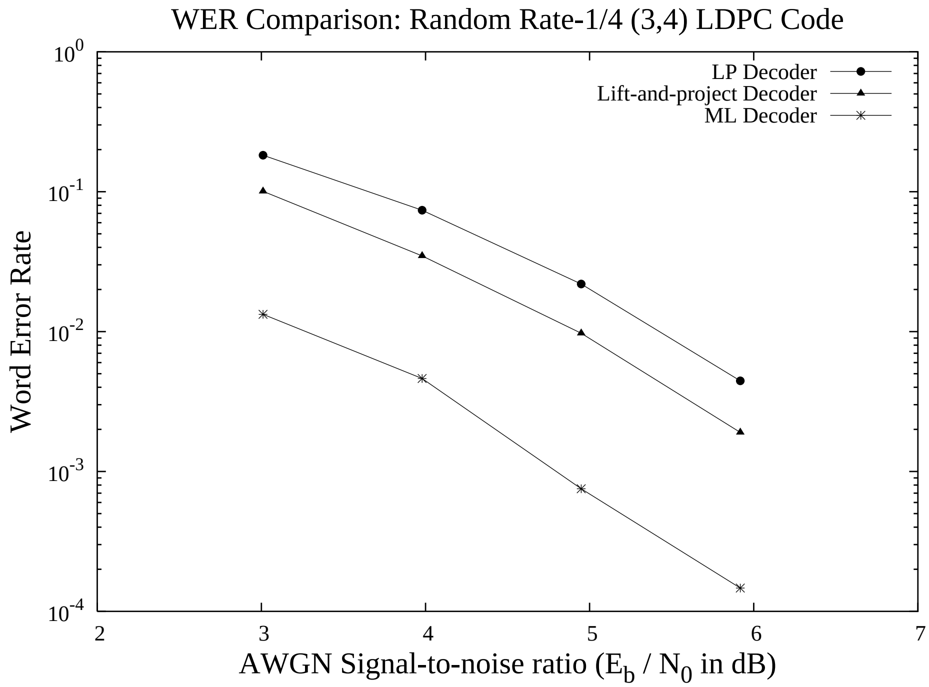


Figure 5-9: The word error rate of the lift-and-project relaxation compared with LP decoding and ML decoding.

$y_i = 0$, then row i is all zeros, and we have that this difference is equal to y. If $y_i = 1$, then row i is the codeword y, and the difference is equal to all zeros. Therefore, this difference vector is always a codeword. Thus, we may enforce the polytope constraints on this vector as well:

$$(f_{11} - f_{1i}, f_{22} - f_{2i}, \ldots, f_{nn} - f_{ni}) \in \dot{\mathcal{Q}} \quad \text{for all} \quad i \in \mathcal{I}$$

Again, the corresponding column constraints are implied. The cost function is applied to the diagonal, since it represents the codeword. Therefore, our objective function is of the form:

$$\text{minimize} \quad \sum_{i \in \mathcal{I}} \gamma_i f_{ii}$$

By the way we constructed this LP, we immediately have that all codewords are feasible solutions. Furthermore, an integral solution must have a codeword on the diagonal, since all integral points in $\dot{\mathcal{Q}}$ are codewords. We may therefore conclude that this LP has the ML certificate property. (Also, the projection of the defined polytope onto the $\{f_{ii}\}_{i \in \mathcal{I}}$ variables is a proper polytope.)

This new "lifted" polytope is a tighter relaxation to the ML decoding problem than the polytope $\mathcal{Q}$. This offers a better decoder, at the expense of increasing the complexity of the polytope from $\Theta(n)$ to $\Theta(n^2)$. Figure 5-9 shows the performance gained by using this tighter relaxation, using a random (3,4) LDPC code of length 36.

We note that for codewords, the matrix F is always positive semi-definite [LS91]. This constraint can be enforced in the relaxation, and the resulting problem would be a *semi-definite program*, also solvable in polynomial time. It would be interesting to interpret this constraint as we have done here, and analyze its affect on performance.

5.4.3 ML Decoding

Another interesting application of LP decoding is to use the polytope $\mathcal{Q}$ to perform ML decoding. Recall that an integer linear program (ILP) is an LP where variables are constrained to take on integer values. If we add the constraint $f_i \in \{0, 1\}$ to our linear program, then we get an exact formulation of ML decoding. In general, integer programming is NP-hard, but there are various methods for solving an IP that far outperform the naive exhaustive search routines. Using the program CPLEX (which uses a branch-and-bound algorithm) [ILO01], we were able to perform ML decoding on LDPC codes with moderate block lengths (up to about 100) in a "reasonable" amount of time. Figure 7-5 includes an error curve for ML decoding an LDPC code with a block length of 60. Each trial took no more than a few seconds (and often much faster) on a Pentium IV (2GHz) processor. Drawing this curve allows us to see the large gap between various suboptimal algorithms and the optimal ML decoder. This gap further motivates the search for tighter LP relaxations to approach ML decoding.

5.5 High-Density Code Polytope

The number of variables and constraints in the polytope $\mathcal{Q}$ has an exponential dependence on the degrees of the check nodes. For LDPC codes, the check nodes have constant degree, so $\mathcal{Q}$ has size linear in n, and this is not a problem. For high-density codes, however, the size of the LP may make it very complex to solve.

Recall that deg_r^+ is the maximum degree of any check node in the graph. The polytope $\mathcal{Q}$ has $O(n+m2^{deg_r^+})$ variables and constraints. For turbo and LDPC codes, this complexity is linear in n, since deg_r^+ is constant.

In this section we give a polytope $\mathcal{R}$ with $O(mn + m(deg_r^+)^2 + n deg_\ell^+ deg_r^+) = O(n^3)$ variables and constraints, for use with arbitrary (high-density) linear codes. Furthermore, the projection of this polytope $\mathcal{R}$ will be equivalent to $\dot{\mathcal{Q}}$; thus the polytopes $\mathcal{Q}$, Ω and $\mathcal{R}$ produce the same results when used for LP decoding.

To derive this polytope, we give a new polytope for each local codeword polytope whose size does not have an exponential dependence on the size of the check neighborhood. The local polytopes are based on a construction of Yannakakis [Yan91].

5.5.1 Definitions

For a check $j \in \mathcal{J}$, let $T_j = \{0, 2, 4, \dots, |N(j)|\}$ be the set of even numbers from 0 to $|N(j)|$. Our new polytope has three sets of variables:

- For all $i \in \mathcal{I}$, we have a variable f_i, where $0 \leq f_i \leq 1$. This variable represents the code bit y_i, as before.
- For all $j \in \mathcal{J}$, and $k \in T_j$, we have a variable $\alpha_{j,k}$, where $0 \leq \alpha_{j,k} \leq 1$. This variable indicates the contribution of weight-k local codewords.
- For all $j \in \mathcal{J}$, $k \in T_j$, and $i \in N(j)$, we have a variable $z_{i,j,k}$, where $0 \leq z_{i,j,k} \leq \alpha_{j,k}$. This variable indicates the portion of f_i locally assigned to local codewords of weight k.

Using these variables, we have the following constraint set:

$$\forall i \in \mathcal{I}, j \in N(i), \qquad f_i = \sum_{k \in T_j} z_{i,j,k} \tag{5.11}$$

$$\forall j \in \mathcal{J}, \qquad \sum_{k \in T_j} \alpha_{j,k} = 1 \tag{5.12}$$

$$\forall j \in \mathcal{J}, k \in T_j, \qquad \sum_{i \in N(j)} z_{i,j,k} = k \cdot \alpha_{j,k} \tag{5.13}$$

$$\forall i \in \mathcal{I}, \qquad 0 \leq f_i \leq 1 \tag{5.14}$$

$$\forall j \in \mathcal{J}, k \in T_j, \qquad 0 \leq \alpha_{j,k} \leq 1 \tag{5.15}$$

$$\forall i \in \mathcal{I}, j \in N(i), k \in T_j, \qquad 0 \leq z_{i,j,k} \leq \alpha_{j,k} \tag{5.16}$$

Let $\mathcal{R}$ be the set of points (f, α, z) such that the above constraints hold. This polytope $\mathcal{R}$ has only $O((deg_r^+)^2)$ variables per check node j, plus the $\{f\}$ variables, for a total of $O(n + m(deg_r^+)^2)$ variables. The number of constraints is at most $O(mn + n deg_\ell^+ deg_r^+)$. In total, this representation has at most $O(n^3)$ variables and constraints.

We must now show that optimizing over $\mathcal{R}$ is equivalent to optimizing over $\mathcal{Q}$. Let $\dot{\mathcal{R}}$ represent the projection of $\mathcal{R}$ onto the $\{f_i\}$ variables; i.e.,

$$\dot{\mathcal{R}} = \{f : (f, \alpha, z) \in \mathcal{R}\}.$$

Since the cost function only affects the $\{f\}$ variables, it suffices to show that $\dot{\mathcal{Q}} = \dot{\mathcal{R}}$. Before proving this, we need the following fact:

Lemma 5.13 *Let $X = \{x_1, \ldots, x_n\}$, $x_i \leq M$, and $\sum_i x_i = kM$, where k, n, M and all x_i are non-negative integers. Then, X can be expressed as the sum of sets of size k. Specifically, there exists a setting of the variables $\{w_S : S \subseteq \{1, \ldots, n\}, |S| = k\}$ to non-negative integers such that $\sum_S w_S = M$, and for all $i \in \{1, \ldots, n\}$, $x_i = \sum_{S \ni i} w_S$.*

Proof:[2] By induction on M. The base case ($M = 1$) is simple; all x_i are equal to either 0 or 1, and so exactly k of them are equal to 1. Set $w_{\{i : x_i = 1\}} = 1$.

[2]We thank Ryan O'Donnell for showing us this proof

For the induction step, we will greedily choose a set S consisting of the indices of the k largest values x_i. After increasing w_S by 1, what remains will be satisfied by induction.

Formally, we assume wlog that $x_1 \geq x_2 \geq \cdots \geq x_n$. Set $X' = (x'_1, \ldots, x'_n)$, where $x'_i = x_i - 1$ for $i \leq k$, and $x'_i = x_i$ otherwise. Since $\sum_i x_i = kM$ and $x_i \leq M$, it must be the case that the largest k values $x_1, \ldots, x_k$ are at least 1. Furthermore, we must have $x_i \leq M - 1$ for all $i > k$. Therefore $0 \leq x'_i \leq M - 1$ for all i. We also have $\sum_i x'_i = \sum_i x_i - k = (M-1)k$. Therefore, by induction, X' can be expressed as the sum of w'_S, where S has size k. Set $w = w'$, then increase $w_{\{1,\ldots,k\}}$ by 1. This setting of w expresses X. ■

Proposition 5.14 *The polytopes $\dot{\mathcal{R}}$ and $\dot{\mathcal{Q}}$ are equivalent. Therefore optimizing over $\mathcal{R}$ is equivalent to optimizing over $\mathcal{Q}$.*

Proof: Suppose $f \in \dot{\mathcal{Q}}$. Set the variables $\{w_{j,S}\}_{j \in \mathcal{J}, S \in E_j}$ such that $(f, w) \in \mathcal{Q}$. Now set

$$\alpha_{j,k} = \sum_{\substack{S \in E_j,\\ |S| = k}} w_{j,S} \qquad (\forall j \in \mathcal{J}, k \in T_j), \tag{5.17}$$

$$z_{i,j,k} = \sum_{\substack{S \in E_j,\\ |S| = k,\\ S \ni i}} w_{j,S} \qquad (\forall i \in \mathcal{I}, j \in N(i), k \in T_j). \tag{5.18}$$

It is clear that the constraints (5.14), (5.15) and (5.16) are satisfied by this setting. Constraint (5.11) is implied by (5.4) and (5.18). Constraint (5.12) is implied by (5.3) and (5.17). Finally, we have, for all $\forall j \in \mathcal{J}, k \in T_j$,

$$\begin{aligned}
\sum_{i \in N(j)} z_{i,j,k} &= \sum_{i \in N(j)} \sum_{\substack{S \in E_j,\\ |S| = k,\\ S \ni i}} w_{j,S} && \text{(by (5.18))} \\
&= \sum_{\substack{S \in E_j,\\ |S| = k}} |S| w_{j,S} \ = \ k \sum_{\substack{S \in E_j,\\ |S| = k}} w_{j,S} \ = \ k \cdot \alpha_{j,k} && \text{(by (5.17))},
\end{aligned}$$

giving constraint (5.13). We conclude that $(f, \alpha, z) \in \mathcal{R}$ and so $f \in \dot{\mathcal{R}}$. Since f was chosen arbitrarily, we have $\dot{\mathcal{Q}} \subseteq \dot{\mathcal{R}}$.

Now suppose f is a vertex of the polytope $\dot{\mathcal{R}}$. Set the variable sets α and z such that $(f, \alpha, z) \in \mathcal{R}$. For all $j \in \mathcal{J}$, $k \in T_j$, consider the set

$$X_1 = \left\{ \frac{z_{i,j,k}}{\alpha_{j,k}} : i \in N(j) \right\}.$$

By (5.16), all members of X_1 are between 0 and 1. Let $1/\beta$ be a common divisor of the numbers in X_1 such that β is an integer. Let

$$X_2 = \left\{ \beta \frac{z_{i,j,k}}{\alpha_{j,k}} : i \in N(j) \right\}.$$

The set X_2 consists of integers between 0 and β. By (5.13), we have that the sum of the elements in X_2 is equal to $k\beta$. So, by Lemma 5.13, the set X_2 can be expressed as the sum of sets S of size k. Set the variables $\{w_S : S \in N(j), |S| = k\}$ according to Lemma 5.13. Now set $w_{j,S} = \frac{\alpha_{j,k}}{\beta} w_S$, for all $S \in N(j), |S| = k$. We immediately satisfy (5.2). By Lemma 5.13 we get:

$$z_{i,j,k} = \sum_{\substack{S \in E_j, \\ S \ni i, \\ |S| = k}} w_{j,S}, \text{ and} \tag{5.19}$$

$$\alpha_{j,k} = \sum_{\substack{S \in E_j, \\ |S| = k}} w_{j,S}. \tag{5.20}$$

By (5.11), we have

$$\begin{aligned} f_i &= \sum_{k \in T_j} z_{i,j,k} = \sum_{k \in T_j} \sum_{\substack{S \in E_j, \\ S \ni i, \\ |S| = k}} w_{j,S} && \text{(by (5.19))} \\ &= \sum_{S \in E_j, S \ni i} w_{j,S}, \end{aligned}$$

giving (5.4). By (5.12), we have

$$\begin{aligned} 1 &= \sum_{k \in T_j} \alpha_{j,k} = \sum_{k \in T_j} \sum_{\substack{S \in E_j, \\ |S| = k}} w_{j,S} && \text{(by (5.20))} \\ &= \sum_{S \in E_j} w_{j,S}, \end{aligned}$$

giving (5.3). We conclude that $(f, w) \in \mathcal{Q}$, and so $f \in \dot{\mathcal{Q}}$. We have shown that all vertices $f \in \dot{\mathcal{R}}$ are contained in $\dot{\mathcal{Q}}$, and so $\dot{\mathcal{R}} \subseteq \dot{\mathcal{Q}}$. ■

5.6 The Parity Polytope

In this section we present the proof of Jeroslow, showing that the parity polytope is exactly the polytope described by the odd-set constraints for a particular check node in equation (5.6).

Recall that Ω_j is the set of points f such that $0 \leq f_i \leq 1$ for all $i \in \mathcal{I}$, and for all

$S \subseteq N(j)$, $|S|$ *odd*,

$$\sum_{i \in S} f_i + \sum_{i \in (N(j) \setminus S)} (1 - f_i) \leq |N(j)| - 1. \tag{5.21}$$

We must show the following:

Theorem 5.15 [Jer75] *The polytope* $\Omega_j = \dot{\mathcal{Q}}_j = \{f : \exists w \text{ s.t. } (f, w) \in \mathcal{Q}_j\}$.

Proof: For all $i \notin N(j)$, the variable f_i is unconstrained in both Ω_j and $\dot{\mathcal{Q}}_j$ (aside from the constraints $0 \leq f_i \leq 1$); thus, we may ignore those indices, and assume that $N(j) = \mathcal{I}$.

Let f be a point in $\dot{\mathcal{Q}}_j$. By the constraints (5.3) and (5.4), $\dot{\mathcal{Q}}_j$ is the convex hull of the incidence vectors of even-sized sets $S \in E_j$. Since all such vectors satisfy the constraints (5.21) for check node j, then f must also satisfy these constraints. Therefore $f \in \Omega_j$.

For the other direction, suppose some point $f' \in \Omega_j$ is not contained in $\dot{\mathcal{Q}}_j$. Then some facet F of $\dot{\mathcal{Q}}_j$ cuts f' (makes it infeasible). Since $0 \leq f'_i \leq 1$ for all $i \in \mathcal{I}$, it must be the case that F passes through the hypercube $[0,1]^n$, and so it must cut off some vertex of the hypercube; i.e., some $x \in \{0,1\}^n$. Since all incidence vectors of even-sized sets are feasible for $\dot{\mathcal{Q}}_j$, the vertex x must be the incidence vector for some odd-sized set $S \notin E_j$.

For a particular $f \in [0,1]^n$, let $[f]_i = 1 - f_i$ if $x_i = 1$, and $[f]_i = f_i$ if $x_i = 0$. We specify the facet F in terms of the variables $[f]$, using the equation

$$F: \qquad a_1[f]_1 + a_2[f]_2 + \cdots + a_n[f]_n \geq b.$$

Since x is infeasible for F, it must be that $\sum_i a_i[x]_i < b$. Since $[x]_i = 0$ for all $i \in \mathcal{I}$, we have $\sum_i a_i[x]_i = 0$, so we may conclude that $b > 0$.

For some $i' \in \mathcal{I}$, let $x^{\oplus i'}$ denote vector x with bit i' flipped; i.e., $x^{\oplus i'}_{i'} = 1 - x_{i'}$, and $x^{\oplus i'}_i = x_i$ for all $i \neq i'$. Since x has odd parity, we have that for all i', $x^{\oplus i'}$ has even parity, so $x^{\oplus i'} \in \dot{\mathcal{Q}}_j$, and $x^{\oplus i'}$ is not cut by F. This implies that for all $i' \in \mathcal{I}$,

$$\sum_i a_i[x^{\oplus i'}]_i \geq b.$$

Note that $[x^{\oplus i'}]_i = [x]_i = 0$ for all $i \neq i'$, and $[x^{\oplus i'}]_{i'} = 1 - [x]_{i'} = 1$. So, we may conclude $a_{i'} \geq b > 0$, for all $i' \in \mathcal{I}$.

The polytope $\dot{\mathcal{Q}}_j$ is full-dimensional (for a proof, see [Jer75]). Therefore F must pass through n vertices of $\dot{\mathcal{Q}}_j$; i.e., it must pass through at least n even-parity binary vectors. We claim that those n vectors must be the points $\{x^{\oplus i} : i \in \mathcal{I}\}$. Suppose not. Then some vertex x' of $\dot{\mathcal{Q}}_j$ is on the facet F, and differs from x in more than one place. Suppose wlog that $x'_1 \neq x_1$ and $x'_2 \neq x_2$, and so $[x']_1 = 1$, $[x']_2 = 1$. Since x' is on F, we have $\sum_i a_i[x']_i = b$. Therefore

$$a_1 + a_2 + \sum_{i=3}^{n} a_i[x']_i = b.$$

Since $a_i > 0 \ \ \forall i$, we have $\sum_{i=3}^{n} a_i[x']_i \geq 0$, and so

$$a_1 + a_2 \leq b.$$

This contradicts the fact that $a_1, a_2 \geq b > 0$.

Thus F passes through the vertices $\{x^{\oplus i} : i \in \mathcal{I}\}$. It is not hard to see that F is exactly the odd-set constraint (5.21) corresponding to the set S for which x is the incidence vector. Since F cuts f', and F is a facet of Ω_j, we have $f' \notin \Omega_j$, a contradiction. ■

Chapter 6

LP Decoding of Turbo Codes

The introduction of turbo codes [BGT93] revolutionized the field of coding theory by demonstrating error-correcting performance that was significantly better than that of any other code at the time. Since then, volumes of research has focused on design, implementation, and analysis of turbo codes and their variants and generalizations. For a survey, we refer the reader to a textbook written on the subject [VY00].

Turbo codes consist of a set of *convolutional codes*, concatenated in series or in parallel, with *interleavers* between them. A convolutional code (as a block code) is based on convolving the information bits with a fixed-length binary word, and an interleaver reorders the bits using a fixed permutation. Each individual convolutional code has a simple ML decoder; however, the system as a whole is much more complex to decode. The *sum-product* (also known as *belief propagation*) algorithm is the decoder most often used for turbo codes; this decoder iteratively passes messages about the likelihood of each bit between the decoders for the individual convolutional codes.

In this chapter we define a linear program (LP) to decode any turbo code. Our relaxation is very similar to the *min-cost flow problem* (see Chapter 3). We will motivate the general LP by first defining an LP for any convolutional code. In fact, we will give an LP for a more general class of codes: any code that can be defined by a finite state machine. Then, we use this LP as a component in a general LP for any turbo code. We discuss the success conditions of this decoder, which are related to the optimality conditions for the network flow problem.

We apply this LP to a family of codes known as *Repeat-Accumulate* (RA) codes, which are the simplest non-trivial example of a turbo code. For RA codes with rate 1/2, we give a specific code construction, and show that the word error rate under LP decoding can be bounded by an inverse polynomial in the block length of the code, as long as the noise is below a certain constant threshold.

The results in Section 6.3 on RA codes are joint work with David Karger, and have appeared elsewhere [FK02a, FK02b].

6.1 Trellis-Based Codes

In this section we define a linear program for codes represented by a *trellis*. This family of codes is quite general; in fact trellises in their most general form may be used to model any code (even non-binary codes). The trellis is a directed graph where paths in the graph represent codewords. The trellis representation is most useful when the code admits a small-sized trellis; in this case, ML decoding can be performed using the *Viterbi* algorithm [Vit67, For73, For74], which essentially finds the shortest path in the trellis.

The size of our LP will depend (linearly) on the size of the trellis. For simplicity (and because we lose no insight into the LP decoding technique), we will restrict our definition of a trellis-based code to binary codes that model a finite-state machine over time. We also assume that each code has rate $r = 1/R$, for positive integer R.

Section Outline. We begin in Section 6.1.1 by defining an encoding process based on finite state machines. We also define the trellis, which can be seen as a model of the actions of the encoder over time. Finally, we discuss how this encoding process can be made into a code, and how we can use the trellis to perform ML decoding efficiently. In Section 6.1.2, we review a commonly studied form of trellis-based code called a *convolutional code*. These codes traditionally form the building blocks for turbo codes. In Section 6.1.3, we discuss the method of *tailbiting* a trellis, which improves the quality of the code, and changes the structure of the trellis. Finally, in Section 6.1.4, we give an LP relaxation to decode any trellis-based code.

6.1.1 Finite State Machine Codes and the Trellis

Finite State Machines. Let M be a finite-state machine (FSM) over the input alphabet $\{0, 1\}$. An FSM M (Figure 6-1 for example) is simply a directed graph made up of states. Each state (node) in the graph has two outgoing edges, an *input*-`0` edge and an *input*-`1` edge. For a given edge e, we use $type(e) \in \{0, 1\}$ to denote the "type" of the edge. In diagrams of the FSM, we use solid lines to denote input-`0` edges, and dotted lines to denote input-`1` edges. Each edge has an associated *output label* made up of exactly R bits. For an edge e, we use $label(e) \in \{0, 1\}^R$ to denote the output label.

The FSM can be seen as an encoder as follows. When the machine receives a block of input bits, it examines each bit in order; if the bit is a zero, the machine follows the input-`0` edge from its current state to determine its new state, and outputs the output label associated with the edge it just traveled. If the bit is a one, it follows the input-`1` edge. The overall rate of this encoding process is $r = 1/R$.

For example, consider the FSM in figure 6-1. Suppose we start in state `00`, and would like to encode the information bits `1010`. Our first information bit is a `1`, so we follow an input-`1` edge (dotted line) to the new state `10`, and output the label `11`. The next information bit is `0`, so we follow the input-`0` edge (solid line) to the new state `01`, and output the label `10`. This continues for the next two information bits,

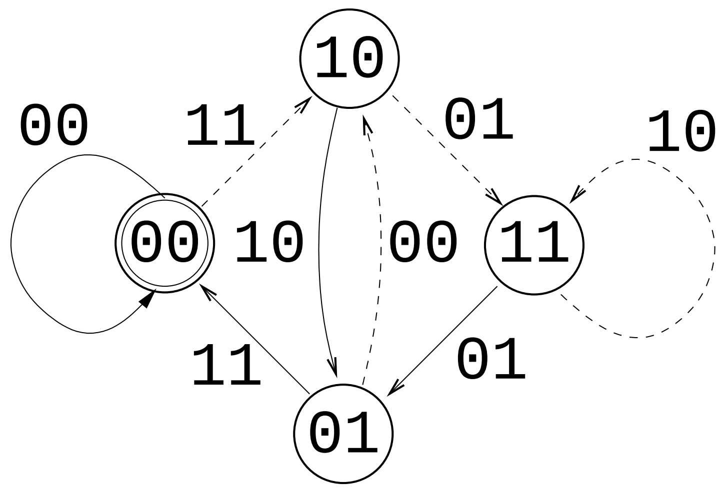


Figure 6-1: A state transition table for a rate-1/2 convolutional encoder with generating polynomials 5,7. The bit pairs on the transition edges represent encoded output bits.

and overall, we travel the path `00` $\rightarrow$ `10` $\rightarrow$ `01` $\rightarrow$ `10` $\rightarrow$ `01`, and output the code bits `11100010`.

The Trellis In order to simulate the operation of a single FSM M over some number k of input bits, we will define a *trellis* T. The trellis T is a graph with $k+1$ copies $(M_0, \ldots, M_k)$ of the states of M as nodes.

The trellis T has edges between every consecutive copy of M, representing the transitions of the encoder at each time step, as it encodes an information word of length k. So, for each edge (s, s') connecting state s to state s' in the FSM M, the trellis contains, for each time step t where $1 \leq t \leq k$, an edge from state s in M_{t-1} to state s' in M_t. This edge will inherit the same type and output label from edge (s, s') in M. Figure 6-2 gives the trellis for the FSM in Figure 6-1, where $k = 4$. The bold path in Figure 6-1 indicates the path taken by the encoder in the example given earlier, when encoding `1010`.

We can view the encoder as building a path through the trellis, as follows. Denote a "start-state" of the FSM. The trellis length k is equal to the length of the information word. The FSM encoder begins in the start state in M_0, uses the information word x as input, and outputs a codeword y. The length of this codeword will be $kR = k/r$ $= n$. Since every state has exactly two outgoing edges, there are exactly 2^k paths of length k from the start state, all ending in trellis layer M_k. So, the set of codewords is in one-to-one correspondence the set of paths of length k from the start state in M_0.

Before proceeding, we need to establish some notation we will use throughout our study of trellis-based codes and turbo codes. For each node s in a trellis, define $out(s)$ to be the set of outgoing edges from s, and $in(s)$ to be the set of incoming edges. For a set of nodes S, define $out(S)$ and $in(S)$ to be the set of outgoing and incoming edges from the node set S.

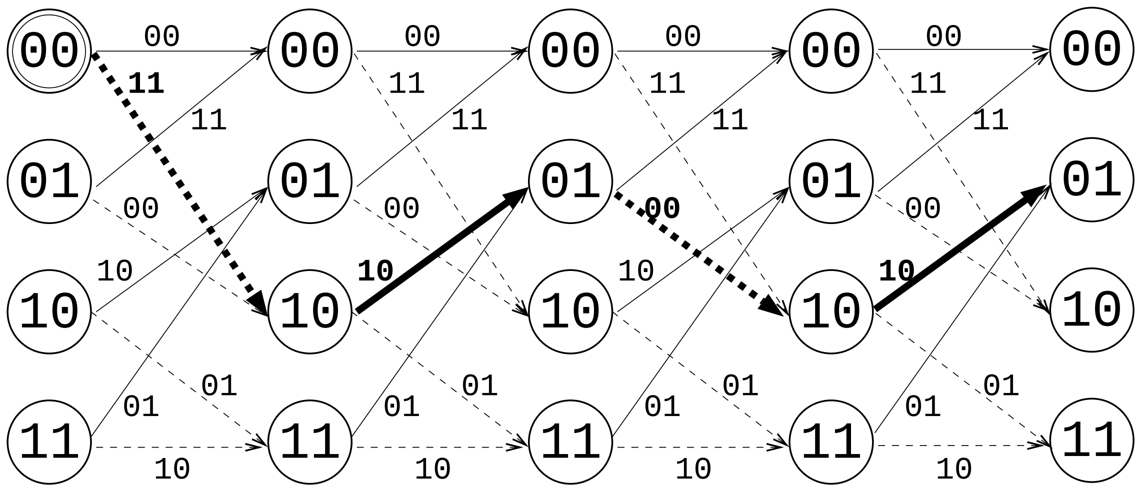


Figure 6-2: A trellis for the rate-1/2 FSM code in figure 6-1. The trellis has length $k = 4$, and five node layers $(M_0, M_1, M_2, M_3, M_4)$. As in the state transition table, the bit pairs on the transition edges represent encoded output bits. Each layer of the trellis represents one input bit. The bold path is the path taken by the encoder while encoding the information bits `1010`.

Let I_t be the set of "input-`1`" edges entering layer t. Formally,

$$I_t = \{e \in in(M_t) : type(e) = \mathtt{1}\}.$$

We also define edge sets O_i, where an edge is in O_i if it outputs a `1` for the i^{th} code bit. Formally,

$$O_i = \{e \in in(M_t) : label(e)_\ell = \mathtt{1}\},$$

where $t = \lfloor (i-1)/R \rfloor + 1$ and $\ell = i - R(t-1)$.

Decoding Trellis-Based Codes The Viterbi algorithm, developed by Viterbi [Vit67], and Forney [For73] [For74], is the one used by most decoders of trellis-based codes. The idea is to keep track (implicitly) of all possible codewords using dynamic programming, and output the one that is most likely. This can be done by assigning a cost to each edge of the trellis, and then finding the lowest-cost path of length k from the start state of the trellis.

For each edge e in T, define a cost γ_e. This is also referred to as the *branch metric*. This cost will be the sum of the costs γ_i of the code bits y_i in the output label (denoted $label(e)$) of the edge that are set to `1`. (Recall that the cost γ_i of a code bit is the log-likelihood ratio of the bit, as defined in Section 2.5.) More formally,

$$\gamma_e = \sum_{i : e \in O_i} \gamma_i.$$

In the BSC, the cost, or *branch metric* of an edge can also be set to the Hamming distance between the label on the edge and the bits of $\tilde{y}$ that were apparently transmitted at that time step. This is the same "cost-rescaling" we discussed in Section 2.5. For example, suppose we were using the code from Figure 6-2, and we have $\tilde{y}_1 = \mathtt{1}$ and $\tilde{y}_2 = \mathtt{0}$. The two code bits y_1 and y_2 were originally transmitted using the label of some edge from the first trellis layer (i.e., for some $e \in in(M_1)$). So, for each edge in that layer, we assign a cost equal to the Hamming distance between the label on the edge and the received sequence 10. For example, the edge e from 01 to 10 between layers M_0 and M_1 would have cost $\Delta(label(e), \mathtt{10}) = \Delta(\mathtt{00}, \mathtt{10}) = 1$.

The ML decoding problem is to find the minimum cost path of length k from the given start state. Since the graph is acyclic, the shortest path can be found by breadth-first search from the start state. This is exactly the Viterbi algorithm. There are many implementation issues surrounding the Viterbi algorithm that affect its performance; for details we refer the reader to textbooks that discuss convolutional codes [Wic95, JZ98].

6.1.2 Convolutional Codes

Convolutional codes can be seen as a particular class of FSM-based codes. Their simple encoder and decoder have made them a very popular class of codes. In this section we describe the basics of convolutional codes; for more details, we refer the reader to textbooks written on the subject [Wic95, JZ98].

The *state* of a convolutional encoder is described simply by the last $\kappa - 1$ bits fed into it, where κ is defined as the *constraint length* of the code. So, as the encoder processes each new information bit, it "remembers" the last $\kappa - 1$ information bits. The output (code bits) at each step is simply the sum (mod 2) of certain subsets of the last $\kappa - 1$ input bits. (We note that convolutional codes with *feedback* cannot be described this way, but can still be expressed as FSM codes.)

Convolutional encoders are often described in terms of a circuit diagram, such as the one in Figure 6-3. These diagrams also illustrate how simple these encoders are to build into hardware (or software). The circuit for a convolutional encoder with constraint length κ has $\kappa - 1$ *registers*, each holding one bit. In general, at time t of the encoding process, the i^{th} register holds the input bit seen at time $t - i$. In addition, there are R *output bits*, each one the mod-2 sum of a subset of the registers and possibly the input bit. The connections in the circuit indicate which registers (and/or the input bit) are included in this sum.

For example, in Figure 6-3, the constraint length is 3, and so there are 2 registers. The rate is 1/2, so there are two output bits. The bits $(x_1, x_2, \ldots, x_k)$ are fed into the circuit one at a time. Suppose the current "time" is t. The first output bit is the sum of the current input bit (at time t) and the contents of the second register: the bit seen at time $t-2$. The second output bit is the sum of the input bit, and the two registers: the bits seen at time $t-1$ and $t-2$.

The FSM and Trellis For Convolutional Codes We can also describe a convolutional code using a finite state machine. The state s of a convolutional encoder

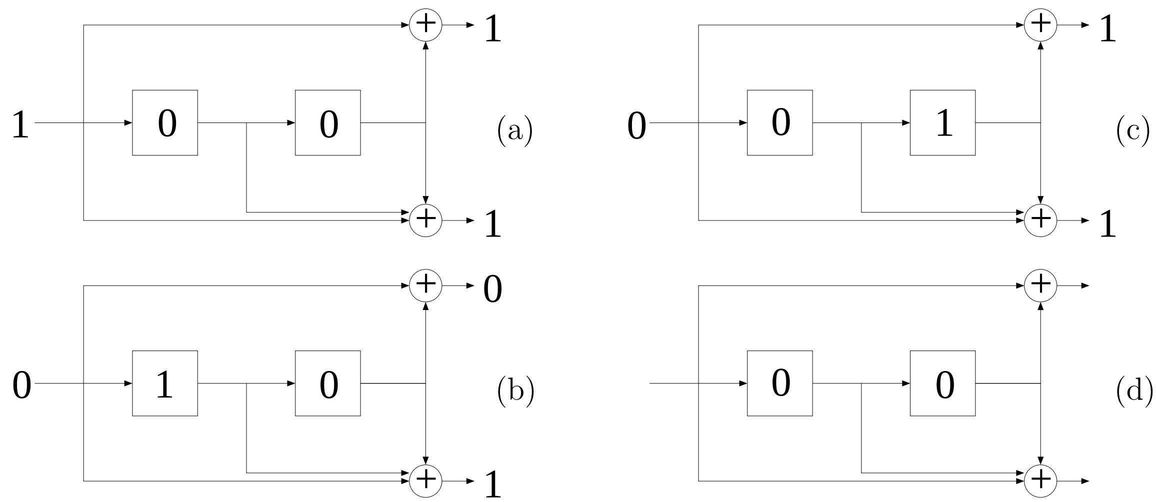


Figure 6-3: The actions of a convolutional encoder for a rate-1/2 convolutional code, from its initial state `00`, given an input stream of `100`. Diagram (a) shows the initial state of the encoder, with both memory elements set to `0`. The first input bit is a `1`. Upon seeing this bit, the encoder outputs $\texttt{1} + \texttt{0} = \texttt{1}$ for the first code bit, and $\texttt{1} + \texttt{0} + \texttt{0} = \texttt{1}$ for the second code bit. Diagram (b) shows the encoder after processing the first input bit; the memory elements have slid over one step, so the new state is `10`. The new input bit is `0`, and so the next output bits are $\texttt{0} + \texttt{0} = \texttt{0}$ and $\texttt{0} + \texttt{1} + \texttt{0} = \texttt{1}$. Diagram (c) shows the next state (`01`), the new input bit (`0`) and the next output bits (`1` and `1`). Finally, diagram (d) shows the final state `00` after processing all three input bits.

can be described by the contents of the registers in the circuit, so $s \in \{0,1\}^{\kappa-1}$.

The FSM for our running example is the same one shown in Figure 6-1. This is simply a graph with one node for each possible state $s \in \{0,1\}^{\kappa-1}$ of the circuit in Figure 6-3. For each node with state s, we draw two edges to other states, representing the two possible transitions from that state (one for an input bit of `1`, one for an input bit of `0`). As before, the edges are solid lines if the transition occurs for an input bit of `0` and dotted lines for an input bit of `1`.

The edges are labeled as before, with the bits output by the encoder circuit making this transition. For example, if we take the first step in Figure 6-3, the current state is `00`, the input bit is a `1`, the next state is `10`, and the output bits are `11`. So, in our FSM graph we make an edge from state `00`, written with a dotted line, into state `10`, and label the edge with code bits `11`. We derive the trellis for a convolutional code from the FSM, as we did in the previous section. The trellis for the code in Figure 6-3 is shown in Figure 6-2.

Feedback If a convolutional code has *feedback*, then its new state is not determined by the previous $\kappa-1$ information bits, but rather is determined by the previous $\kappa-1$ *feedback bits*. The feedback bit at time t is the sum (mod 2) of the information bit at time t and some subset of the previous $\kappa-1$ feedback bits. These codes still have a simple state transition table, and can be decoded with a trellis. We will see

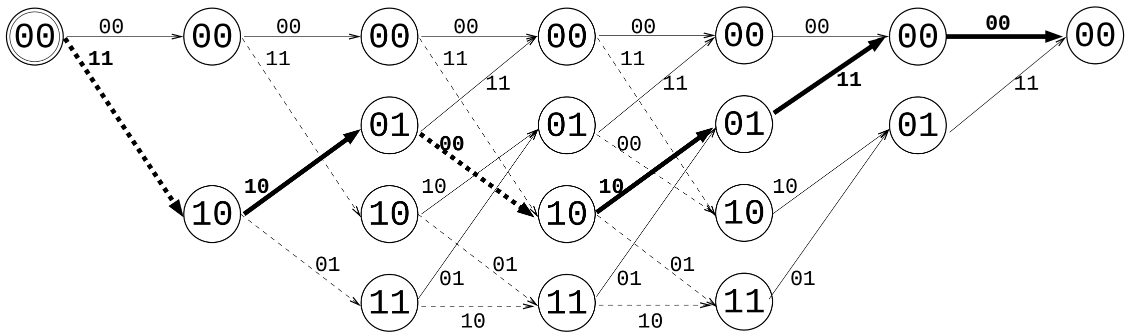


Figure 6-4: A terminated trellis for a rate-1/2 convolutional encoder. The bold path is the cycle taken by the encoder while encoding the information bits `1010 00`, which corresponds to the original information `1010` padded with two zeros.

an "accumulator" later in the thesis, which is a simple example of a convolutional code with feedback. For more details on feedback in convolutional codes, we refer the reader to [Wic95] or [JZ98].

6.1.3 Tailbiting

There are other ways to define codes based on this trellis that use "tailbiting." This is a method of code design such that codewords correspond to *cycles* in a cyclic trellis rather than paths in a conventional trellis. Depending on the structure of the FSM, tailbiting methods may remove edges and nodes of the trellis, redefine the encoding process, or assume a certain structure of the input sequence.

We can see the importance of tailbiting when we consider convolutional codes. In convolutional codes without tailbiting, the final few bits of the input stream are not as "protected" as the first bits; they do not participate in as many code bits. To counteract this problem, there are two common solutions.

1. One solution is to *terminate* the code by forcing it to end in the all-zero state. We can do this by padding the end of the information stream with $\kappa - 1$ `0`s. This way, the encoder always finishes in state $0^{\kappa-1}$ at time k. The drawback to padding with `0`s is the loss of rate; we must add $(\kappa - 1)(1/r)$ symbols to the codeword without gaining any information bits.

   If we use this scheme, then we can redraw the trellis with $\kappa - 1$ extra layers, as in figure 6-4. Note that in the figure we have removed some nodes and edges that are never used by the encoder. This new trellis can easily be made into a cyclic trellis if we merge the start state in M_0 with the final all-zeros state in node layer M_k (not shown in the figure). If we do this merge, the set of codewords is exactly the set of cycles in the trellis of length k.

2. A more elegant solution is to use the last $\kappa - 1$ bits of the information word, in reverse order, to determine the starting state at trellis layer 0. For example,

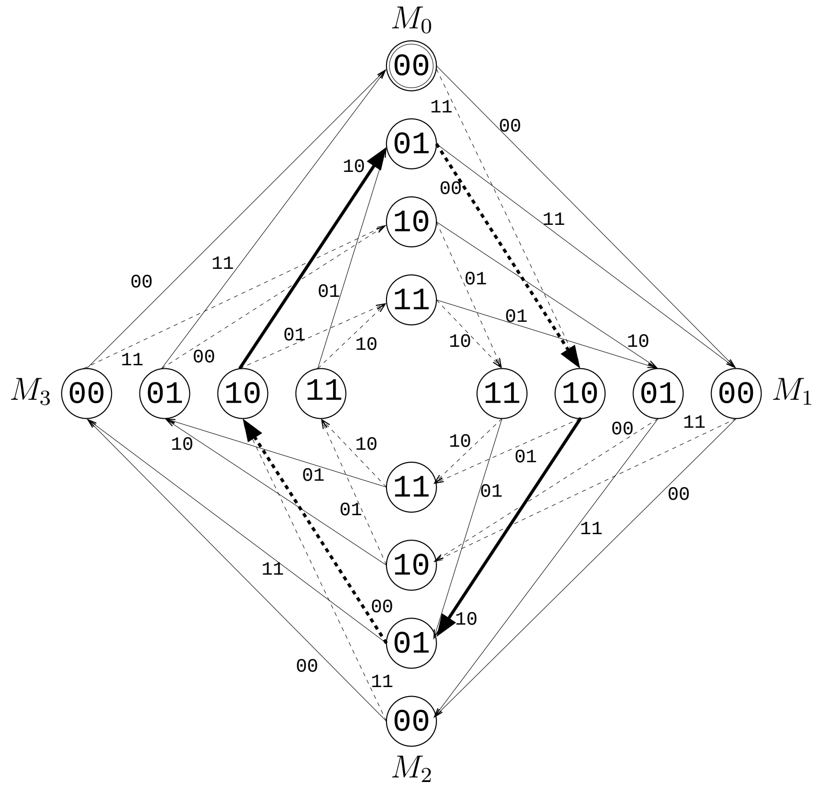


Figure 6-5: A cyclic trellis for a tailbiting rate-1/2 convolutional encoder. The trellis has length $k = 4$, and four node layers (M_0, M_1, M_2, M_3). The bold path is the path taken by the encoder while encoding the information bits `1010`.

> if $\kappa = 3$, and the last two bits were `01`, then the start state of the encoder is set to state `10`. Now encode as before, starting from this state rather than `00`. The encoder will always end in the state it began, since the state is exactly the previous $\kappa - 1$ bits seen. Therefore, we can merge the first and last node layer of the trellis, and the codewords correspond to cycles in the trellis, rather than paths.
>
> Figure 6-5 shows an appropriate trellis for this scheme, applied to our running example code. Now, if the information word is `1010`, we set the start state equal to `01`, then follow the cycle `01` $\to$ `10` $\to$ `01` $\to$ `10` $\to$ `01`, as seen in the figure.

From these two examples, we define a cyclic trellis abstractly as follows. A cyclic trellis has k layers of nodes $(M_0, \ldots, M_{k-1})$. All edges in the trellis go between consecutive layers of the trellis. (Note that M_{k-1} and M_0 are considered consecutive.)

As in the conventional trellis, every node has two outgoing edges, one of each "input type." In addition, every edge has an output label with exactly R bits. The definitions of I_t and O_i remain the same, where M_k is considered to be the same as M_0.

Given a cyclic trellis T, the set of codewords in the code it defines is in one-to-one correspondence with the set of *cycles* in T of length exactly k. Each such cycle contains exactly one node from each node layer $(M_0, \ldots, M_{k-1})$. The codeword itself can be read from the labels on the cycle, starting with the edge emanating from the node from M_0. Furthermore, if we examine the "types" of the edges in the cycle (either input-`0` or input-`1`) beginning with the node in layer M_0, then we have the information word that this codeword encodes.

The particular tailbiting scheme used depends on the code. Any non-cyclic trellis can be made cyclic by adding "dummy edges" from the last trellis node layer to the start state in the first layer. So, for the purposes of defining our LP decoder, we will assume tailbiting has been performed, and we will work with cyclic trellises.

6.1.4 LP Formulation of trellis decoding

Since every cycle in the trellis passes through M_0, the ML decoding problem on tail-biting trellises is easily solved by performing $|M_0|$ shortest path computations. However, we present a sub-optimal LP (min-cost flow) formulation of decoding in order to make it generalize more easily to turbo codes, where we have a set of trellises with dependencies between them.

We define a variable f_e for each edge e in the trellis T, where $0 \le f_e \le 1$. Our LP is simply a *min-cost circulation* LP [AMO93] applied to the trellis T, with costs γ_e. Specifically, our objective function is:

$$\text{Minimize} \quad \sum_{e \in T} \gamma_e f_e$$

We have flow conservation constraints on each node:

$$\sum_{e \in out(s)} f_e = \sum_{e \in in(s)} f_e \tag{6.1}$$

We also force the total flow around the trellis to be one. Since every cycle in T must pass between layers M_0 and M_1, it suffices to enforce:

$$\sum_{e \in out(M_0)} f_e = 1 \tag{6.2}$$

We define auxiliary LP variables $(x_1, \ldots, x_k)$ for each information bit, and LP variables $(y_1, \ldots, y_n)$ for each code bit. These variables indicate the value that the LP assigns to each bit. They are not necessary in the definition of the linear program, but make the connection to decoding much clearer. The value of an information bit x_t should indicate what "type" of edge is used at layer t of the trellis; if $x_i = 1$, this indicates that the encoder used an input-`1` edge at the i^{th} trellis layer. Accordingly, we set

$$x_t = \sum_{e \in I_t} f_e. \tag{6.3}$$

The value of y_i indicates the i^{th} bit output by the encoder. Therefore, if $y_i = 1$, then the edge taken by the encoder that outputs the i^{th} bit should have a `1` in the proper position of the edge label. To enforce this in the LP, we set

$$y_i = \sum_{e \in O_i} f_e. \tag{6.4}$$

Since all variables are indicators, we enforce the constraints

$$\begin{array}{lll} 0 \le f_e \le 1 & \text{for all} & e \in T, \\ 0 \le x_t \le 1 & \text{for all} & t \in \{1, \ldots, k\} \text{ and} \\ 0 \le y_i \le 1 & \text{for all} & i \in \{1, \ldots, n\}. \end{array} \tag{6.5}$$

We use the notation poly(T) to denote the polytope corresponding to these constraints; formally,

$$\text{poly}(T) = \{(f, x, y) : \text{equations (6.1)} - \text{(6.5) hold}\}.$$

Overall, our LP can be stated as:

$$\text{Minimize} \;\; \sum_{e \in T} \gamma_e f_e \;\; \text{s.t.} \;\; (f, x, y) \in \text{poly}(T).$$

Or, equivalently,

$$\text{Minimize} \;\; \sum_{i=1}^{n} \gamma_i y_i \;\; \text{s.t.} \;\; (f, x, y) \in \text{poly}(T).$$

The integral points of the polytope. The polytope poly(T) is a min-cost circulation polytope for the graph T, with the added requirement (6.2) that the total flow across the edges of the first layer of the trellis is exactly one. From simple flow-based arguments, we can see that the integral solutions are in one-to-one correspondence with codewords.

Before proceeding, consider the possible cycles in the trellis T. It is not hard to see that every cycle in T has length αk for some integer $\alpha \ge 1$, and has exactly α edges per layer of the trellis. Codewords correspond to cycles of length exactly k.

Theorem 6.1 *For any trellis T, the set of integral points in* poly(T) *is in one-to-one correspondence with the cycles of T of length k.*

Proof: Let y be an arbitrary codeword. Let f be the cycle flow that corresponds to the length-k cycle in the trellis taken by the encoder, with one unit of flow across the cycle. In other words, $f_e = 1$ if e is on the encoder cycle, and $f_e = 0$ otherwise. Let x_i be the information word that the codeword y encodes. The point (f, x, y) is in the polytope poly(T), since it satisfies every polytope constraint.

Let (f, x, y) be some integral point in poly(T). By the flow conservation constraints (6.1), the flow f is a circulation in the graph T. By the constraint (6.2), the flow f is non-zero. Since f is integral, it must have a decomposition into integral cycle flows [AMO93]. However, f must itself have only one unit of flow, since we have $f_e \leq 1$ for all edges e. Therefore, f can be decomposed into a collection of cycle flows, each with one unit of flow. Since $f_e \leq 1$ for all edges in the trellis, these cycle flows must be disjoint. By the constraint (6.1), at most one of these cycle flows passes through the first trellis layer; however, every cycle in the trellis must pass through the first trellis layer. Therefore, f is itself a single cycle of length k with one unit of flow. This is exactly the cycle corresponding to the codeword y. ■

Optimality Conditions The linear program will output a min-cost circulation that sends one unit of flow around the trellis. Every circulation has a cycle decomposition; in circulations with one unit of flow, this can be regarded as a convex combination of simple cycles. If all the flow was put onto the minimum-cost cycle in this decomposition, the resulting flow would have cost at most the cost of the circulation. Therefore we may assume that some simple cycle flow f^* obtains the LP optimum.

We have argued that if the circulation is integral, then it corresponds to a cycle of length k. Note also that the cost of an integral solution is exactly the cost of the cycle, which in turn is equal to the cost of the codeword. So if the LP solution is integral, it corresponds to the minimum-cost, or ML codeword, which is contained in the LP variables $\{y_i\}$. Thus in this case we may output this codeword, along with a proof that it is the ML codeword.

If the solution is fractional, then the cycle f^* has length αk for some integer $\alpha > 1$. As in our previous relaxations, in this case we simply output "error." Note that whenever we output a codeword, it is guaranteed to be the optimal ML codeword. Thus our decoder has the *ML certificate* property.

As before, the decoder will succeed if the output codeword is the one that was transmitted. Thus the decoder can fail in two ways: if it outputs "error," or if it outputs a codeword, but the most likely codeword was not the one transmitted. In the latter case, we can say that the code (rather than the decoder) has failed, since ML decoding would have failed as well.

6.2 A Linear Program for Turbo Codes

In general, a *turbo code* is any set of convolutional codes, concatenated in serial or parallel, with interleavers between them. These are often referred to in the literature as "turbo-like" codes [DJM98], because they are a generalization of the original turbo code [BGT93]. In this section we describe an LP to decode any turbo code. In fact, we will describe an LP for a more general class of code realizations, defined on trellises built from arbitrary finite-state machines, and associations between sets of edges in the trellises.

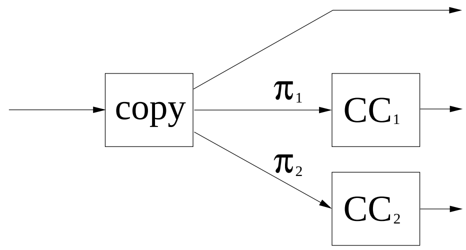


Figure 6-6: A circuit diagram for a classic rate-1/3 Turbo code. The "copy" FSM simply copies the input to the three outputs. The two component codes CC_1 and CC_2 are convolutional codes.

6.2.1 Turbo Codes

The original turbo code [BGT93] consists of two component rate-1 convolutional codes concatenated in parallel, with an interleaver in front of each of them. Additionally, the information bits themselves are added to the codeword. Figure 6-6 shows a tree representation of this code. The encoder for the turbo code in Figure 6-6 takes as input an information word x of length k. Two separate interleaves (fixed permutations) π_1 and π_2 are applied to the information word. Then, the length-k words $\pi_1(x)$ and $\pi_2(x)$ are sent to the encoders for rate-1 convolutional codes CC_1 and CC_2. The output of these encoders, along with a copy of the information word x, is output as the codeword.

In general we will define a turbo code by a directed out-tree $\mathcal{T}$, whose nodes correspond to trellises T. Each edge in the tree has an associated *interleaver* π; this is a permutation on k elements, where k is the length of the trellis whose corresponding node the edge enters.

Formally, let $\mathcal{T}$ be a directed out-tree, where the nodes $\{1, \ldots, |\mathcal{T}|\}$ correspond to trellises $\{T^1, \ldots, T^{|\mathcal{T}|}\}$, where each trellis is as described in Section 6.1. We assume that each trellis has tailbiting. By convention, we have node 1 in $\mathcal{T}$ as the root, corresponding to trellis T^1. Let k_m denote the length of trellis T^m, $1 \leq m \leq |\mathcal{T}|$, and let R_m denote the length of the output labels on edges of trellis T^m. For each edge from m to m' in the tree $\mathcal{T}$, there is an interleaver $\pi[m, m']$, a permutation on $k_{m'}$ elements. Let $L(\mathcal{T})$ denote the set of leaves of $\mathcal{T}$ (the nodes in $\mathcal{T}$ with no outgoing edges).

The encoder for a turbo code takes a block of k information bits, feeds it into the trellis at the root, and sends it through the tree. The codeword is output at the leaves of the tree. An individual trellis T^m of size k_m receives a block of bits of size k_m from its parent in $\mathcal{T}$, applies its encoding process to the block, and sends a copy of its output block of size $R_m k_m$ to each of its children in $\mathcal{T}$. Each edge applies its permutation to the bits sent across it. For the encoder to work properly, it must be the case that for every edge from m to m' in $\mathcal{T}$, we have $k_m R_m = k_{m'}$, so the number of output bits of trellis T^m is equal to the number of input bits for trellis $T^{m'}$.

The overall codeword is the concatenation of all the outputs of the leaf trellises of

$\mathcal{T}$. For a trellis T^m where $m \in L(\mathcal{T})$, we use y^m to denote the string of $R_m k_m$ code bits output by trellis T^m. The overall code length is then $n = \sum_{m \in L(\mathcal{T})} R_m k_m$, and the overall rate is k/n. This codeword is transmitted over the channel, and a corrupt codeword $\tilde{y}$ is received. We use $\tilde{y}^m$ to denote the corrupt bits corresponding to the code bits y^m, for some $m \in L(\mathcal{T})$.

6.2.2 TCLP: Turbo-Code Linear Program

In this section we define the *Turbo Code Linear Program* (TCLP), a generic LP that decodes any turbo code. We will first outline all the variables of TCLP, then give the objective function and the constraints. Basically, the LP will consist of the min-cost flow LPs associated with each trellis, as well as "agreeability constraints," tying together adjacent trellises in $\mathcal{T}$.

Variables. All variables in TCLP are indicator variables in $\{0, 1\}$, relaxed to be between 0 and 1. For each $m \in \mathcal{T}$, we have variables $x^m = (x^m_1, \ldots, x^m_{k_m})$ to denote the bits entering trellis T^m. The TCLP variables $(x^1_1, \ldots, x^1_{k_1=k})$ represent the information bits, since they are fed into the root trellis T^1. Additionally, we have variables $y^m = (y^m_1, \ldots, y^m_{R_m k_m})$ to denote the output bits of each trellis T^m. Finally, for each $m \in \mathcal{T}$, and edge e in trellis T^m, we have a flow variable f_e; we use the notation f^m to denote the vector of flow variables f_e for all edges e in trellis T^m.

Objective Function. The objective function will be to minimize the cost of the code bits. Since the code bits of the overall code are only those output by the leaves of the encoder, we only have costs associated with trellises that are leaves in $\mathcal{T}$. For each trellis T^m where $m \in L(\mathcal{T})$, we have a cost vector γ^m, with a cost γ^m_i for all $i \in \{1, \ldots, R_m k_m\}$. These costs are associated with the bits y^m_i output by the encoder for trellis T^m. The value of γ^m_i is the log-likelihood ratio for the bit y_i, given the received bit $\tilde{y}^m_i$, as described in Section 2.5. Thus our objective function is simply

$$\text{minimize} \quad \sum_{m \in L(\mathcal{T})} \sum_{i=1}^{R_m k_m} \gamma^m_i y^m_i.$$

Constraints. Recall that for a trellis T, the polytope $\text{poly}(T)$ is the set of unit circulations around the trellis T (defined in section 6.1.4). In TCLP, we have all the constraints of our original trellis polytope $\text{poly}(T^m)$ for each trellis T^m in the tree $\mathcal{T}$. In addition, we have equality constraints to enforce that the output of a trellis is consistent with the input to each of the child trellises. We define the LP constraints formally as follows:

- **Individual trellis constraints:** For all $m \in \mathcal{T}$,

  $$(f^m, x^m, y^m) \in \text{poly}(T^m).$$

  These constraints enforce that the values f^m are a unit circulation around the trellis T^m, and that x^m and y^m are consistent with that circulation.

- **Interleaver consistency:** For all edges $(m, m') \in \mathcal{T}$, for all $t \in \{1, \ldots, k_{m'}\}$,

$$y_t^m = x_{\pi[m,m'](t)}^{m'}.$$

  These constraints enforce consistency between trellises in the tree $\mathcal{T}$ that share an edge, using the interleaver associated with the edge.

A decoder based on TCLP has the ML certificate property for the following reasons. Every integral solution to TCLP corresponds to a single cycle in each trellis (this follows from Theorem 6.1 for trellis-based codes). Furthermore, the consistency constraints enforce a correspondence between adjacent trellises in $\mathcal{T}$, and the output bits $\{y^m : m \in L(\mathcal{T})\}$. We may conclude that every integral solution to this LP represents a valid set of encoding paths for an information word x, and the cost of the solution is exactly the cost of the codeword output by this encoding at the leaves. Thus this LP has the ML certificate property.

Success Conditions. We can use network flow theory to get a handle on when the LP decoder will succeed; i.e., when the encoder path is in fact the optimal solution to the LP. Formally, let $(\overline{f}, \overline{x}, \overline{y})$ represent the TCLP solution corresponding to the actions of the encoder. In other words, $\overline{x}$ represents the words input to each trellis by the encoder, $\overline{y}$ represents the words output by each trellis, and for every trellis T^m, the flow $\overline{f}^m$ is exactly the simple cycle of length k_m followed by the encoder. The LP decoder will succeed if and only if $(\overline{f}, \overline{x}, \overline{y})$ is the optimal solution to TCLP. (Note that we assume failure under multiple LP optima.)

Consider some other feasible solution (f, x, y) to TCLP, where $f \neq \overline{f}$. The flow difference $f - \overline{f}$ represents a circulation (with a net flow of zero across each trellis layer) in the residual graph $T^m_{\overline{f}}$ for each trellis T^m. Furthermore, this circulation "agrees" between trellis layers, i.e., the interleaver consistency constraints are satisfied. Define an *agreeable circulation* as any circulation $f - \overline{f}$ formed in this way. The cost of the circulation is the difference in cost between f and $\overline{f}$. We conclude that decoding with TCLP fails if and only if there is some negative-cost agreeable circulation in the residual graphs of the trellises $\{T^m_{\overline{f}}\}_{m \in \mathcal{T}}$. In the next section on repeat-accumulate codes, we will use this idea to derive specific combinatorial conditions for decoding failure, and use these conditions to derive codes and error bounds.

6.2.3 A Note on Turbo Codes as LDPC Codes.

Many trellis-based codes, including convolutional codes, can be represented by a factor graph; furthermore, the factor graph will have constant degree for a fixed-size FSM. This can be extended to turbo codes [Mac99] as well, and so turbo codes are technically a special case of LDPC codes.

However, if we were to write down a factor graph for a trellis-based code or a turbo code and plug in the polytope $\mathcal{Q}$ from Chapter 5, we would get a weaker relaxation in general. When the factor graph is written for a trellis, the $w_{j,S}$ variables of $\mathcal{Q}$ correspond to edge variables of the trellis. However, in $\mathcal{Q}$, the variables $w_{j,S}$

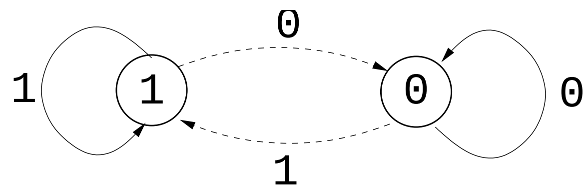


Figure 6-7: A state transition table for an accumulator.

are constrained only in their relation to single bit values f_i in the neighborhood $N(j)$, whereas in poly(T), the edge variables are constrained by flow conservation constraints on states. Each state can potentially represent a whole *sequence* of bit values.

Thus our turbo code relaxation can also be seen as a tightening of the polytope $\mathcal{Q}$ for the special case of turbo codes. We note that for repeat-accumulate codes, since the trellis is so simple, the polytopes turn out to be equivalent, and so the TCLP relaxation is *not* a tightening of the polytope $\mathcal{Q}$ in this case.

6.3 Repeat-Accumulate Codes

Repeat-Accumulate codes are perhaps the simplest non-trivial example of a turbo code. They were introduced by Divsalar and McEliece [DJM98] in order to show the first bounds for error probabilities under ML decoding. Their simple structure and highly efficient encoding scheme make them both practical and simpler to analyze than other more complex turbo codes. They have also been shown experimentally to have excellent error-correcting ability under iterative decoding [DJM98], competing with classic turbo codes.

The RA code is the concatenation (in series) of an outer *repetition code* and a particular inner convolutional code. A repetition-R code is simply a code whose encoder repeats each bit of the information word R times, and outputs the result. The encoder for an *accumulator* code simply scans through the input, and outputs the sum (mod 2) of the bits seen so far.

The accumulator is a convolutional code with feedback. For details on feedback in convolutional codes, we refer the reader to [Wic95]. For our purposes, it suffices to consider the state transition table for the accumulator, as shown in figure 6-7.

In this section we present the TCLP relaxation as it applies to RA codes. We give precise conditions for LP decoding error when using TCLP for RA codes. We also show how these conditions suggest a design for an interleaver for RA(2) codes, and prove an inverse polynomial upper bound on the word error probability of the LP decoder when we use this interleaver.

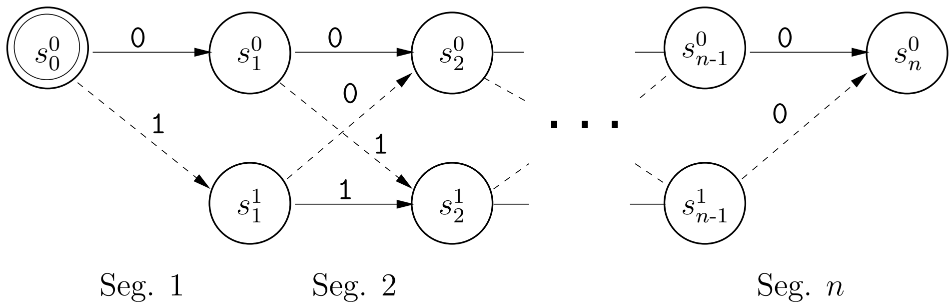


Figure 6-8: The trellis for an accumulator, used in RA codes. The dashed-line edge correspond to information bit 1, the solid-line edges to information bit 0. The edges are labeled with their associated output code bit.

6.3.1 Definitions and Notation

Formally, an RA(R) code is a rate-$1/R$ code of length $n = Rk$. The code has an associated interleaver (permutation) $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$. The encoder is given an information word $x = (x_1, \ldots, x_k)$ of length k. Let x' be the be the repeated and permuted information word, i.e., for all $i \in \{1, \ldots, n\}$, we have

$$x'_{\pi(i)} = x_{\lfloor (i-1)/R \rfloor + 1}.$$

The RA encoder outputs a codeword y of length n, where for all $i \in \{1, \ldots, n\}$,

$$y_i = \sum_{i'=1}^{i} x'_{i'} \mod 2.$$

For all $t \in \{1, \ldots, k\}$, let X_t be the set of R indices to which information bit x_t was repeated and permuted, i.e.,

$$X_t = \{\pi(R(t-1)+1), \pi(R(t-1)+2), \ldots, \pi(Rt)\}.$$

Let $\mathcal{X} = \{X_t : t \in \{1, \ldots, k\}\}$. As a tailbiting scheme, we assume that the input contains an even number of 1s. This can be achieved by padding the information word with an extra parity bit. Thus the rate of this code is $(k-1)/Rk$, or $1/R - o(1)$.

The Accumulator Trellis. Using the table in Figure 6-7, we can view the accumulator as a simple finite state machine, receiving the n-bit binary input string x' one bit at a time. The accumulator has two possible states per time step, depending on the parity of the input bits seen so far. We refer to the two states of the accumulator as 0 and 1; state 0 represents even parity, 1 represents odd parity.

We can construct a trellis for this inner accumulator in the same way as a conventional convolutional code. The trellis T for an accumulator is shown in Figure 6-8. The trellis has n layers, one for each code bit. Since the accumulator has only two states, each node layer M_i has two nodes, representing even and odd parity. We use $\{s_0^0, \ldots, s_n^0\}$ and $\{s_1^1, \ldots, s_{n-1}^1\}$ to refer to the sets of even and odd parity nodes, respectively.

An encoder using this trellis begins in state s_0^0. At each time step, if it receives a `1` as input, it follows the dashed-line transition to the next layer of nodes, switching states. If it receives a `0`, it follows the solid-line transition to the next layer of nodes, staying in the same state (see Figure 6-8). The labels on the transition edges represent the code bit output by the encoder taking that transition; label `0` for edges entering state 0, label `1` for edges entering state 1.

A path from s_0^0 across the trellis to s_n^0 corresponds to an entire n-bit input string. Since the accumulator begins and ends in state 0 (the input string has even parity by assumption), we do not need the states s_0^1 and s_n^1. Looking at the edge labels on the path, one can read off the codeword corresponding to that input string. Let $\overline{P}$ be the path taken by the encoder through the trellis T while encoding x'.

Recall that in a trellis, the set M_i refers to the nodes in the i^{th} segment of the trellis; in this case,

$$M_i = \{s_i^0, s_i^1\}.$$

Furthermore, recall that $in(s)$ refers to the edges entering a node s and that $in(S)$ refers to the edges entering nodes in a set S. Also, as before, $type(e)$ indicates the "type" of the edge (either input-1 or input-0). In this case, for all layers $1 \leq i \leq n$, we have

$$\begin{aligned} type(s_{i-1}^0, s_i^1) &= type(s_{i-1}^1, s_i^0) &= 1, \text{ and} \\ type(s_{i-1}^0, s_i^0) &= type(s_{i-1}^1, s_i^1) &= 0. \end{aligned}$$

Finally, recall the definitions of I_i and O_i. In this case, we have

$$\begin{aligned} I_i &= \{ e \in in(M_i) : type(e) = \mathtt{1} \} \\ &= \{ (s_{i-1}^0, s_i^1), (s_{i-1}^1, s_i^0) \} \end{aligned}$$

and

$$\begin{aligned} O_i &= \{ e \in in(M_i) : label(e) = \mathtt{1} \} \\ &= \{ (s_{i-1}^0, s_i^1), (s_{i-1}^1, s_i^1) \}. \end{aligned}$$

The *cost* γ_e of an edge e in the trellis at segment t is as defined for trellis codes. In this case, we have $\gamma_e = \gamma_i$ for all $e \in O_i$, and $\gamma_e = 0$ otherwise, where γ_i is the log-likelihood ratio for the bit, as defined in Section 2.5. In the BSC, the cost of an edge in segment i can be set to the Hamming distance between the label of the edge and the received bit $\tilde{y}_i$. The cost of a path is the sum of the costs of the edges on the path.

Note that we are not regarding this trellis in a cyclic manner; We could say that $s_0^0 = s_n^0$, and thus binary words of length n would correspond to cycles in the trellis. However, we separate these two nodes in this section for clarity.

Decoding with the Trellis. Assume for the moment that the accumulator was the entire encoder (i.e., the information bits are fed directly into the accumulator without repeating and permuting). Then, all paths through the trellis from s_0^0 to s_n^0 represent valid information words, and the labels along the path represent valid codewords. Furthermore, the cost of the path is equal to the cost of the associated codeword. Thus, a simple shortest-path computation (the Viterbi algorithm) yields the ML information word.

However, if we tried to apply the Viterbi algorithm to RA codes, we would run into problems. For example, suppose $R = 2$, and let $x_t = 1$ be some arbitrary information bit, where $X_t = \{i, i'\}$. Since information bit x_t is input into the accumulator at time i *and* time i', any path through the trellis T that represents a valid encoding would use an input-`1` at trellis layer i, *and* at trellis layer i'. In general, any path representing a valid encoding would use the same type of edge at every time step $i \in X_t$. We say a path is *agreeable* for x_t if it has this property for x_t. An *agreeable path* is a path that is agreeable for all x_t. Any path that is *not* agreeable does not represent a valid encoding, and thus finding the lowest cost path is not guaranteed to return a valid encoder path.

The ML codeword corresponds to the lowest cost *agreeable* path from s_0^0 to s_n^0. Using TCLP, we instead find the lowest-cost agreeable *flow* from s_0^0 to s_n^0.

6.3.2 RALP: Repeat-Accumulate Linear Program.

Repeat-accumulate codes fall under our definition of a turbo code by regarding the code as a repetition code concatenated with an accumulator. So in a sense we have two trellises: a "repeater" and an accumulator. In the following, we have simplified the LP by eliminating the variables associated with the "repeater" trellis. In addition, we do not write down the variables representing the code bits, and adjust the cost function to affect the edge variables instead.

The resulting integer program contains variables $f_e \in \{0, 1\}$ for every edge e in the trellis, and free variables x_i for every information bit, $i \in \{1, \ldots, k\}$. The relaxation RALP of the integer program simply relaxes the flow variables such that $0 \le f_e \le 1$. RALP is defined as follows:

$$\text{RALP}: \quad \text{minimize} \quad \sum_{e \in T} \gamma_e f_e \qquad \text{s.t.}$$

$$\sum_{e \in out(s_0^0)} f_e = 1 \tag{6.6}$$

$$\sum_{e \in out(s)} f_e = \sum_{e \in in(s)} f_e \qquad \forall\, s \in T \setminus \{s_0^0, s_n^0\} \tag{6.7}$$

$$x_t = \sum_{e \in I_i} f_e \qquad \forall\, i \in X_t, \quad X_t \in \mathcal{X} \tag{6.8}$$

$$0 \le f_e \le 1 \qquad \forall\, e \in T$$

The relaxation RALP is very close to being a simple min-cost flow LP, as in the case of a simple unconstrained trellis. Equation (6.6) enforces that exactly one unit of flow is sent across the trellis. Equation (6.7) is a flow conservation constraint at each node. Unique to RALP are the *agreeability constraints* (6.8). These constraints say that a feasible flow must have, for all $X_t \in \mathcal{X}$, the same amount x_t of total flow on input-$\mathtt{1}$ edges at every segment $i \in X_t$. Note that these constraints also imply a total flow of $1 - x_i$ on input-$\mathtt{0}$ edges at every segment $i \in X_t$. We will refer to the flow values f of a feasible solution (f, x) to RALP as an *agreeable flow*. The free variables x_i do not play a role in the objective function, but rather enforce constraints among the flow values, and represent the information word input to the encoder.

Using RALP as a decoder. A decoding algorithm based on RALP is as follows. Run an LP-solver to find the optimal solution (f^*, x^*) to RALP, setting the costs on the edges according to the received word $\tilde{y}$. If f^* is integral, output x^* as the decoded information word. If not, output "error." We will refer to this algorithm as the *RALP decoder*.

All integral solutions to RALP represent agreeable paths, and thus valid encodings of some information word. This implies that if the optimal solution (f^*, x^*) to RALP is in fact integral, then f^* is the lowest cost agreeable path, and represents the ML codeword. Thus the RALP decoder has the *ML certificate* property: whenever it finds a codeword, it is guaranteed to be the ML codeword.

6.3.3 An Error Bound for RA(2) Codes

Let $\overline{f}$ represent the path flow where one unit of flow is sent along the path $\overline{P}$ taken by the encoder. As in the TCLP relaxation from Section 6.2.2, the RALP decoder fails (does not return the original information word) if and only if the residual graph $T_{\overline{f}}$ contains a negative-cost circulation. In this section, we apply this statement more precisely to the trellis for RA(2) codes. We define an auxiliary graph for the purposes of analysis, where certain subgraphs called *promenades* correspond to circulations in the residual graph $T_{\overline{f}}$. The graph has a structure that depends on the interleaver π and weights that depend on the errors made by the channel.

The promenades in the auxiliary graph have a combinatorial structure that suggests a design for an interleaver. We use a graph construction of Erdös and Sachs [ES63] and Sauer [Sau67] (see also [Big98]) to construct an interleaver, and show that the LP decoder has an inverse-polynomial error rate when this interleaver is used.

Notation. For this section we deal exclusively with RA(2) codes. This means that each set $X_t \in \mathcal{X}$ has two elements. We also use the notation $f(s, s')$ to denote the flow $f_{(s,s')}$ along edge (s, s'). Note that in RA(2) codes, the agreeability constraints (6.8) say that for every $X_t = \{i, i'\}$, the total flow on input-$\mathtt{1}$ edges across trellis segment i is equal to the total flow on input-$\mathtt{1}$ edges across trellis segment i'; i.e., we have that $\forall X_t \in \mathcal{X}$ where $X_t = \{i, i'\}$,

$$x_i = f(s^{\mathtt{0}}_{i-1}, s^{\mathtt{1}}_i) + f(s^{\mathtt{1}}_{i-1}, s^{\mathtt{0}}_i) = f(s^{\mathtt{0}}_{i'-1}, s^{\mathtt{1}}_{i'}) + f(s^{\mathtt{1}}_{i'-1}, s^{\mathtt{0}}_{i'}).$$

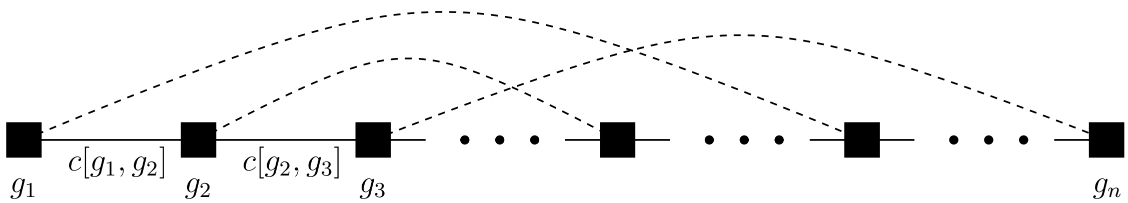


Figure 6-9: The auxiliary graph Θ for RA(2) codes. The graph Θ is defined as a Hamiltonian line $(g_1, \ldots, g_n)$, plus a matching edge $(g_i, g_{i'})$ for each $\{i, i'\} \in \mathcal{X}$.

The Auxiliary Graph and the Promenade. Let Θ be a weighted undirected graph with n nodes $(g_1, \ldots, g_n)$ connected in a line, where the cost $c[g_i, g_{i+1}]$ of edge (g_i, g_{i+1}) is equal to the cost added by decoding code bit i to the opposite value of the transmitted codeword. Formally, we have

$$c[g_i, g_{i+1}] = \gamma_i(1 - y_i) - \gamma_i y_i, \tag{6.9}$$

where y is the transmitted codeword, and γ_i is the log-likelihood ratio of code bit i, as defined in Section 2.5. In the BSC, we have $c[g_i, g_{i+1}] = -1$ if the i^{th} bit of the transmitted codeword is flipped by the channel ($\tilde{y}_i \neq y_i$), and $+1$ otherwise. Note that these costs are not known to the decoder, since they depend on the transmitted codeword. Call these edges the Hamiltonian edges, since they make a Hamiltonian path. Note that we do not have an edge representing the n^{th} code bit. This is because the n^{th} code bit is always equal to `0`, and need not be transmitted. For each $\{i, i'\} \in \mathcal{X}$, we also add an edge to Θ between node g_i and node $g_{i'}$ with cost $c[g_i, g_{i'}] = 0$. We refer to these edges as the matching edges, since they form a perfect matching on the nodes of Θ.

Note that Θ is a Hamiltonian line plus a matching; if the edge (g_n, g_1) were added, then Θ would be a 3-regular Hamiltonian graph (a cycle plus a matching). This graph Θ is illustrated in Figure 6-9. We comment that the graph Θ is essentially the factor graph for the code; in fact, if we regard the existing nodes of Θ as check nodes, and place a variable node on each edge of the graph Θ, we obtain the factor graph for the code.

Define a *promenade* to be a path in Θ that begins and ends in the same node, and may repeat edges as long as it does not travel along the same edge twice in a row. Let $|\Theta|$ denote the length of the path. The cost of a promenade is the total cost of the edges in the path, including repeats (i.e., repeats are not free). Formally, a promenade is a path $\Psi = (p_0, p_1, \ldots, p_{|\Psi|} = p_0)$ in Θ that begins and ends at the same node p_0, where for all $i \in \{0, \ldots, |\Psi| - 1\}$, $p_i \neq p_{i+2 \mod |\Psi|}$. The cost of a promenade Ψ is $c[\Psi] = \sum_{i=0}^{|\Psi|-1} c[p_i, p_{i+1}]$. We are now ready to state our main structural theorem.

Theorem 6.2 *The RALP decoder succeeds if all promenades in Θ have positive cost. The RALP decoder fails if there is a promenade in Θ with negative cost.*

When there is a zero-cost promenade, the RALP decoder may or may not decode correctly (this is the degenerate case where the LP has multiple optima). We will prove Theorem 6.2 in Section 6.3.4, but first we show what it suggests about interleaver design, and how it can used to prove a bound on the probability of error of our algorithm.

For most reasonable channels (e.g., the BSC with crossover probability less than 1/2), it will be the case that if a transmitted bit $y_i = 0$, then the cost γ_i is more likely to be positive than negative; if $y_i = 1$, then γ_i is more likely to be negative than positive. Since the cost of a Hamiltonian edge $c[g_i, g_{i+1}]$ in Θ is equal to $\gamma_i(1-y_i)-\gamma_i y_i$, we have that in both cases, Hamiltonian edges are more likely to have positive cost than negative cost. Thus, promenades with more edges are less likely to have a total negative cost than promenades with fewer edges (at least every other edge of a promenade is Hamiltonian). The *girth* of a graph is the length of its shortest simple cycle. Certainly every promenade contains at least one simple cycle, and so graphs with high girth will only have promenades with many edges. This suggests that what we want out of an interleaver, if we are to use the RALP decoder, is one that produces a graph Θ with high girth.

We use a result of Erdös and Sachs [ES63] and Sauer [Sau67] (see also [Big98]) to make a graph that is a line plus a matching, and has high girth. Their construction allows us, in cubic time, to start with an n-node cycle and build a 3-regular graph Θ with girth $\log n$ that contains the original cycle (assume for simplicity that n is a power of two). We remove an edge from the original cycle to obtain a line plus a matching with girth at least as high. To derive the interleaver, we simply examine the edges added by the construction. We will refer to this interleaver as π_E. This is the same construction used by Bazzi *et al.* [BMMS01] to show a $1/2 \log n$ lower bound on the minimum distance of an RA code using this interleaver.

Theorem 6.3 **[BMMS01]** *The rate 1/2 - $o(1)$ RA code with block length n, using π_E as an interleaver, has minimum distance of at least $\frac{1}{2}\log n$.*

Proof: To show that the code has minimum distance more than $\frac{1}{2}\log n$, we will show that the RALP decoder succeeds as long as fewer than $\frac{1}{4}\log n$ bits are flipped by the binary symmetric channel. This immediately implies the theorem, since if the minimum distance was less than $\frac{1}{2}\log n$, then the decoder would not be able to have this property.

If fewer than $\frac{1}{4}\log n$ bits are flipped by the channel, then all simple paths in Θ containing $\frac{1}{2}\log n$ Hamiltonian edges have less than half of their edges with cost -1, and thus have more than half with cost $+1$. Thus all such paths have positive cost, and by Lemma 6.4, every promenade has positive cost. By Theorem 6.2, the RALP decoder succeeds. ■

Error Bound. We would like to bound the probability that Θ contains a promenade with cost less than or equal to zero, thus bounding the probability that the RALP decoder fails. However, there are many promenades in Θ, so even a tight bound on the probability that a particular one is negative is insufficient. Furthermore, the fact that promenades may repeat edges creates dependencies that interfere with using standard Chernoff bound analysis. Consequently, we need to find a simpler structure that is present when there is a promenade with cost less than or equal to zero. In the following, a simple path or cycle means a path or cycle that does not repeat edges. For clarity (to avoid floors and ceilings), we will assume n is a power of 16, though our arguments do not rely on this assumption.

Lemma 6.4 *Assume Θ has girth at least $\log n$. If $n \geq 2^4$, and there exists a promenade Ψ in Θ where $c[\Psi] \leq 0$, then there exists a simple path or cycle Y in Θ that contains $\frac{1}{2}\log n$ Hamiltonian edges, and has cost $c[Y] \leq 0$.*

Proof: Let $\Psi = (p_0, p_1, \ldots, p_{|\Psi|-1}, p_{|\Psi|} = p_0)$ be a promenade in Θ where $c[\Psi] \leq 0$. Since Ψ contains a cycle, we have $|\Psi| \geq \log n$. Contract every matching edge in Ψ to obtain a closed path $H = (h_0, h_1, \ldots, h_{|H|} = h_0)$.

No two matching edges share an endpoint, so at most every other edge of H is a matching edge. Thus,

$$|H| \geq \frac{1}{2}|\Psi| \geq \frac{1}{2}\log n.$$

Let the cost of H be the sum of the costs of the edges on H, where repeated edges count every time they appear in H, i.e.,

$$c[H] = \sum_{i=0}^{|H|-1} c[h_i, h_{i+1}].$$

We know that $h_i \neq h_{i+2}$ for all i where $0 \leq i < |H|$ (operations on indices of $h \in H$ are performed mod $|H|$). If this was not the case, then there would be a cycle in Θ of length 3; this in turn would violate our assumption that $n \geq 2^4$, since the girth of Θ is at least $\log n$.

Let $H_i = (h_i, \ldots, h_{i+\frac{1}{2}\log n})$, a subsequence of H containing $\frac{1}{2}\log n$ edges, and let

$$c[H_i] = \sum_{j=i}^{i+\frac{1}{2}\log n-1} c[h_j, h_{j+1}].$$

We know that H_i has no repeated edges (i.e., it is a simple path or a simple cycle); if it were not, then if we added the matching edges back into H we would get a path in Θ with at most $\log n$ edges that repeated an edge, which would imply a cycle in Θ of length less than $\log n$.

Since all matching edges have zero cost, $c[\Psi] = c[H]$. Note that

$$c[H] = \left(\frac{1}{\frac{1}{2}\log n}\right) \sum_{i=0}^{|H|-1} c[H_i],$$

since every edge of H is counted in the sum exactly $(\frac{1}{2}\log n)$ times. Since $c[H] = c[\Psi] \leq 0$, at least one simple path or cycle H_{i^*} must have $c[H_{i^*}] \leq 0$. Set Y to be the simple path or cycle in Θ obtained by expanding the zero-cost matching edges in the path or cycle H_{i^*}. ■

Theorem 6.2, along with the above construction of Θ and a union bound over the paths of length $\frac{1}{2}\log n$ of Θ, gives us an analytical bound on the probability of error when decoding with the RALP decoder under the BSC:

Theorem 6.5 *For any $\epsilon > 0$, the rate $1/2 - o(1)$ RA code with block length n using π_E as an interleaver decoded with the RALP decoder under the binary symmetric channel with crossover probability $p < 2^{-4(\epsilon+(\log 24)/2)}$ has a word error probability* WER *of at most $n^{-\epsilon}$.*

Proof: By Theorem 6.2, and Lemma 6.4, the RALP decoder succeeds if all simple paths or cycles in Θ with $\frac{1}{2}\log n$ Hamiltonian edges have positive cost. We claim that there are at most $n \cdot 3^{\frac{1}{2}\log n}$ different simple paths that have $\frac{1}{2}\log n$ Hamiltonian edges. To see this, consider building a path by choosing a node, and building a simple path from that node, starting with a Hamiltonian edge. On the first step, there are two choices for the Hamiltonian edge. On each subsequent step, if one has just traversed a Hamiltonian edge to some node g, one can either continue along the Hamiltonian path, or traverse a matching edge. If a matching edge is traversed to some node g', then one of the two Hamiltonian edges incident to g' is the next edge traversed. We conclude that after traversing a Hamiltonian edge, there are at most three choices for the next Hamiltonian edge; thus the total number of paths with $\frac{1}{2}\log n$ Hamiltonian edges starting at a particular node is at most $3^{\frac{1}{2}\log n}$, and the total number of paths in the graph with $\frac{1}{2}\log n$ Hamiltonian edges is at most $n \cdot 3^{\frac{1}{2}\log n}$.

The remainder of the proof is a union bound over the paths of Θ with $\frac{1}{2}\log n$ Hamiltonian edges. Let Y be a particular path with $\frac{1}{2}\log n$ Hamiltonian edges. Each Hamiltonian edge has cost -1 or $+1$, so at least half of the Hamiltonian edges must have cost -1 in order for $c[Y] \leq 0$. Each Hamiltonian edge had cost -1 with probability p, so the probability that $c[Y] \leq 0$ is at most $\binom{\frac{1}{2}\log n}{\frac{1}{4}\log n} p^{\frac{1}{4}\log n}$. By the union bound,

$$\begin{aligned}
\text{WER} &\leq n 3^{\frac{1}{2}\log n} \binom{\frac{1}{2}\log n}{\frac{1}{4}\log n} p^{\frac{1}{4}\log n} \\
&\leq n^{1+\frac{1}{2}\log 3} n^{\frac{1}{2}} 2^{-(\epsilon + \frac{1}{2}\log 24)\log n} \\
&\leq n^{1+\frac{1}{2}\log 3 + \frac{1}{2} - (\epsilon + \frac{1}{2}\log 24)} \\
&\leq n^{-\epsilon}
\end{aligned}$$

■

A Bound for the AWGN Channel. We can derive a WER bound for the AWGN channel, also using a union bound, stated in the following theorem.

Theorem 6.6 *For any $\epsilon > 0$, the rate $1/2 - o(1)$ RA code with block length n using π_E as an interleaver decoded with the RALP decoder under the AWGN channel with variance $\sigma^2 \leq \frac{\log e}{4+2\log 3+4\epsilon}$ has a word error probability* WER *of at most $\frac{\sigma}{\sqrt{2\pi \log n}} n^{-\epsilon}$.*

Proof: As in the proof for Theorem 6.5, we have a union bound over the over the simple paths with $\frac{1}{2}\log n$ Hamiltonian edges. The cost of a Hamiltonian edge in the AWGN channel is $1 + z_i$, where $z_i \in \mathcal{N}(0, \sigma^2)$ is a Gaussian random variable representing the additive noise on code bit i with zero mean and variance σ^2.

Consider a particular path P of Θ with $\frac{1}{2}\log n$ Hamiltonian edges. We also use P to denote the set of code bits i such that the Hamiltonian edge (g_i, g_{i+1}) is on the path P. Thus the probability that the path has negative cost is at most:

$$\begin{aligned}
\Pr[c[P] \leq 0] &= \Pr\left[\sum_{i \in P}(1 + z_i) \leq 0\right] \\
&= \Pr\left[\sum_{i \in P} z_i \leq -|P|\right]
\end{aligned}$$

All z_i are independent Gaussian random variables with zero mean. The sum of independent Gaussian variables with the same mean is itself a Gaussian variable whose variance is the sum of the variances of the individual variables. So, if we let $\mathcal{Z} = \sum_{i \in P} z_i$, we have that $\mathcal{Z} \in \mathcal{N}\left(0, \frac{1}{2}\sigma^2 \log n\right)$ is a Gaussian with zero mean and variance $\frac{1}{2}\sigma^2 \log n$. Furthermore, we have

$$\begin{aligned}
\Pr[c[P] \leq 0] &\leq \Pr\left[\mathcal{Z} \leq -\frac{1}{2}\log n\right] \\
&= \Pr\left[\mathcal{Z} \geq \frac{1}{2}\log n\right].
\end{aligned}$$

For a Gaussian $z \in \mathcal{N}(0, s^2)$ with mean 0 and variance s^2, we have [Fel68], for all $x > 0$,

$$\Pr[z \geq x] \leq \frac{s}{x\sqrt{2\pi}} e^{-\frac{x^2}{2s^2}},$$

and so

$$\begin{aligned}
\Pr[c[P] \leq 0] &\leq \frac{\sigma}{\sqrt{2\pi \log n}} e^{-\frac{\log n}{4\sigma^2}} \\
&= \frac{\sigma}{\sqrt{2\pi \log n}} n^{-\frac{\log e}{4\sigma^2}}.
\end{aligned}$$

As argued in Theorem 6.5, the number of paths with $\frac{1}{2}\log n$ Hamiltonian edges is at most $n 3^{\frac{1}{2}\log n}$. Thus by the union bound,

$$\begin{aligned}
\text{WER} &\leq n 3^{\frac{1}{2}\log n} \Pr[c[P] \leq 0] \\
&= n^{1 + \frac{1}{2}\log 3} \Pr[c[P] \leq 0] \\
&\leq \frac{\sigma}{\sqrt{2\pi \log n}} n^{1 + \frac{1}{2}\log 3 - \frac{\log e}{4\sigma^2}}
\end{aligned}$$

By assumption, $\sigma^2 \leq \frac{\log e}{4+2\log 3+4\epsilon}$, and so $-\epsilon \geq 1 + \frac{1}{2}\log 3 - \frac{\log e}{4\sigma^2}$. This implies

$$\text{WER} \;\; \leq \;\; \frac{\sigma}{\sqrt{2\pi \log n}} n^{-\epsilon}.$$

■

We note that as $\epsilon \to 0$, the threshold on p stated in Theorem 6.5 approaches $\approx 2^{-9.17}$, and the threshold on σ^2 stated in Theorem 6.6 approaches $\approx 1/5$.

6.3.4 Proof of Theorem 6.2

In this section we prove Theorem 6.2. We will use some basic analytical tools used for the *min-cost flow* problem, specifically the notions of a *residual graph*, a *circulation* and a *path decomposition* [AMO93]. (See Section 3.2 for background.)

Theorem 6.2 *The RALP decoder succeeds if all promenades in Θ have positive cost. The RALP decoder fails if there is a promenade in Θ with negative cost.*

Proof: Let $\overline{f}$ be the unit flow along the path $\overline{P}$ taken by the encoder through the trellis. The RALP decoder will succeed if $\overline{f}$ is the unique optimal solution to RALP. The RALP decoder will fail if $\overline{f}$ is not an optimal solution to RALP. To prove the theorem, we will first establish conditions under which $\overline{f}$ is optimal that are very similar to the conditions under which a normal flow is optimal; specifically, we will show that $\overline{f}$ is optimal if and only if the residual graph $T_{\overline{f}}$ does not contain a certain negative-cost subgraph. We will then show a cost-preserving correspondence between these subgraphs in $T_{\overline{f}}$ and promenades in Θ, the Hamiltonian line graph on which Theorem 6.2 is based.

Agreeable Circulations. Suppose that f is some feasible flow for a normal min-cost flow problem (without agreeability constraints) on an arbitrary graph G. Let G_f be the residual graph after sending flow according to f. A fundamental fact in network flows is that a solution f is optimal iff G_f does not contain a negative-cost circulation [AMO93].

Now consider the agreeable flow problem (RALP) on the trellis T, and the solution $\overline{f}$ to RALP. For an arbitrary flow f, we denote the cost of f as $c[f]$, where

$$c[f] = \sum_{e \in T} \gamma_e f_e.$$

Define the residual graph $T_{\overline{f}}$ in the same manner as in normal min-cost flow. The residual graph $T_{\overline{f}}$ has the following structure. Since $\overline{f}$ is a unit flow across a single path the trellis, and each edge has unit capacity, the graph $T_{\overline{f}}$ contains all the edges on $\overline{P}$ in the reverse direction (toward s_0^0), with cost $-\gamma_e$. All other edges of the trellis remain the same, and have cost γ_e. Figure 6-10 shows an example of the residual graph $T_{\overline{f}}$.

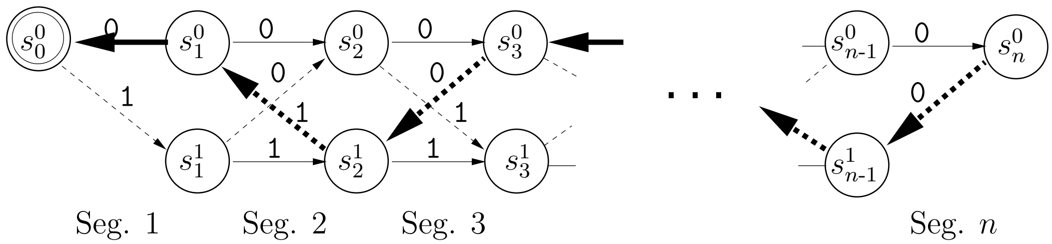


Figure 6-10: The residual graph $T_{\overline{f}}$. The bold path is the path $\overline{P}$ taken by the encoder, in reverse. All edges e in $T_{\overline{f}}$ have the same cost γ_e as they did in the original trellis, except the reversed edges of $\overline{P}$, which have cost $-\gamma_e$.

Let f' be some feasible agreeable flow in T, where $f' \neq \overline{f}$. Consider the circulation $f' - \overline{f}$ in $T_{\overline{f}}$. This circulation sends one unit of flow from s_0^0 to s_n^0 along the flow f', then sends it back from s_n^0 to s_0^0 along the path $\overline{P}$. The cost of the circulation is $c[f'] - c[\overline{f}]$. Since both $\overline{f}$ and f' obey the agreeability constraints, the circulation $f' - f$ also obeys the agreeability constraints. We call such a circulation an *agreeable circulation.*

Lemma 6.7 *The RALP decoder succeeds if all agreeable circulations in $T_{\overline{f}}$ have positive cost.*

Proof: If the RALP decoder fails, then $\overline{f}$ is not the unique optimal solution to RALP. Let $f^* \neq \overline{f}$ be some optimal solution to RALP. Consider the circulation $f' = f^* - \overline{f}$. Since $c[f'] = c[f^*] - c[\overline{f}]$, and $c[f^*] \leq c[\overline{f}]$, we know that $c[f'] \leq 0$. It is clear that the sum or difference of two agreeable flows is agreeable, so f' is agreeable. ■

Lemma 6.8 *The RALP decoder fails if there exists an agreeable circulation in $T_{\overline{f}}$ with negative cost.*

Proof: Let f' be a circulation in $T_{\overline{f}}$ with negative cost. Let $f^* = \overline{f} + f'$. Since $c[f'] < 0$, we have $c[f^*] < c[\overline{f}]$. The flow f^* is the sum of two agreeable flows, and so it is also an agreeable flow. Since f^* is a feasible solution to RALP with cost less than $\overline{f}$, the flow $\overline{f}$ is not optimal, thus the RALP decoder fails. ■

Subpaths of Θ and cycles in $T_{\overline{f}}$. In the remainder of the proof, we show a correspondence between agreeable circulations in $T_{\overline{f}}$ and promenades in Θ. We first define a special class of cycles in $T_{\overline{f}}$ and show that the cost of a cycle in this class is the same as that of a corresponding subpath in Θ. This is depicted in Figure 6-11. Then we show that every simple cycle in $T_{\overline{f}}$ is from this class. We conclude the proof by arguing that agreeable circulations always contain sets of these cycles that correspond to promenades in Θ with the same cost.

We define our "special" paths and cycles as follows:

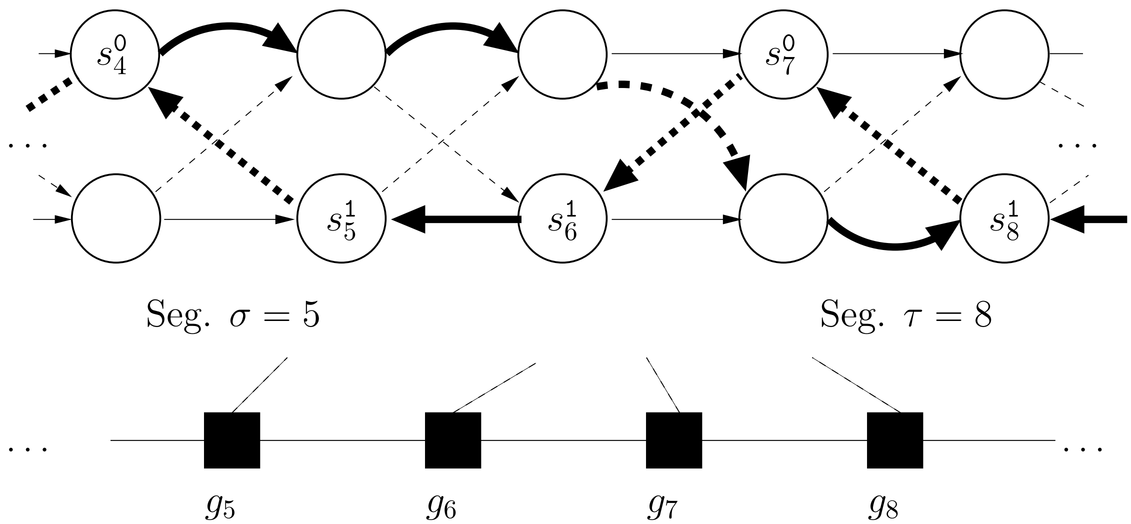


Figure 6-11: An example of a cycle $\nu(\sigma = 5, \tau = 8)$ in the residual graph $T_{\overline{f}}$, and the corresponding path $\mu(\sigma = 5, \tau = 8)$ in Θ. The cycle $\nu(\sigma = 5, \tau = 8)$ (above in bold) contains a subpath (straight bold lines) backward along residual edges of $\overline{P}$, where $\overline{P} = (\dots, s_4^0, s_5^1, s_6^1, s_7^0, s_8^1, \dots)$, and the complimentary forward path from s_4^0 to s_8^1 (curved bold lines). The subpath $\mu(\sigma = 5, \tau = 8)$ in Θ is the path from g_5 to g_8 along the Hamiltonian edges of Θ.

- Let $\mu(\sigma, \tau)$ (where $\sigma < \tau$) be a certain path in Θ, as depicted in Figure 6-11. Specifically, $\mu(\sigma, \tau)$ denotes the path $(g_\sigma, g_{\sigma+1}, \dots, g_\tau)$ of Hamiltonian edges in Θ. The cost $c[\mu(\sigma, \tau)]$ of the path is equal to the sum of the costs of the edges on the path.

- Let $\nu(\sigma, \tau)$ (where $\sigma < \tau$) be a certain cycle in $T_{\overline{f}}$, as depicted in Figure 6-11. This cycle will correspond to rerouting flow away from $\overline{P}$ between trellis segments σ and τ. We define $\nu(\sigma, \tau)$ formally as follows. Let y be the transmitted codeword. For the purposes of this definition, we let $y_0 = 0$. (The bit y_0 is not part of the codeword; it just makes the notation simpler.) Recall that $\overline{P}$ is the path taken by the accumulator when encoding x. The path $\overline{P}$ can also be expressed as follows:
$$\overline{P} = (s_0^{y_0}, s_1^{y_1}, \dots, s_n^{y_n}).$$
Recall that in $T_{\overline{f}}$, all edges along $\overline{P}$ have a unit residual capacity going backward toward s_0^0, since $\overline{f}$ is the unit flow across path $\overline{P}$. The cycle $\nu(\sigma, \tau)$ in $T_{\overline{f}}$ consists of the backward subpath along $\overline{P}$:
$$(s_\tau^{y_\tau}, s_{\tau-1}^{y_{\tau-1}}, \dots, s_\sigma^{y_\sigma}, s_{\sigma-1}^{y_{\sigma-1}}),$$
and the forward subpath that is "opposite" to $\overline{P}$ between segments σ and τ:
$$(s_{\sigma-1}^{y_{\sigma-1}}, s_\sigma^{1-y_\sigma}, s_{\sigma+1}^{1-y_{\sigma+1}}, \dots, s_{\tau-2}^{1-y_{\tau-2}}, s_{\tau-1}^{1-y_{\tau-1}}, s_\tau^{y_\tau}).$$
The cost $c[\nu(\sigma, \tau)]$ of the cycle is the sum of the costs of the edges in $\nu(\sigma, \tau)$.

In Figure 6-11, we have $\sigma = 5$, $\tau = 8$; we also have $y_4 = 0$, $y_5 = 1$, $y_6 = 1$, $y_7 = 0$, $y_8 = 1$. The figure shows both the cycle $\nu(\sigma, \tau)$ and the corresponding path $\mu(\sigma, \tau)$.

Consider the flow $P' = \overline{P} + \nu(\sigma, \tau)$ in the original trellis T. By pushing flow around the cycle $\nu(\sigma, \tau)$, the resulting flow P' is a path flow that follows the subpath "opposite" to $\overline{P}$ between segments σ and τ; otherwise, P' is identical to $\overline{P}$. Note that when we "reroute" this flow, the types of the edges (either input-0 or input-1) change only for the segments σ and τ (and not the edges in-between). The labels of the edges, however, change for the segments $\sigma \ldots \tau - 1$. This idea of "rerouting" flow will be important as we continue with the proof.

The following claim shows that the cost of the flow $\mu(\sigma, \tau)$ in $T_{\overline{f}}$ is the same as the cost of the path $\mu(\sigma, \tau)$ in Θ.

Claim 6.9 *For all pairs of trellis segments σ and τ, where $1 \leq \sigma < \tau \leq n$, we have $c[\mu(\sigma, \tau)] = c[\nu(\sigma, \tau)]$.*

Proof: First note that the two edges of $\nu(\sigma, \tau)$ from trellis segment τ have opposite cost, and thus have a net contribution of zero to the cost of $\nu(\sigma, \tau)$. (In Figure 6-11, the two edges in consideration are (s_8^1, s_7^0) and (s_7^1, s_8^1).) To see this, recall that the cost of an edge is determined by the state that it enters; the forward edge of $\nu(\sigma, \tau)$ from trellis segment τ enters state $s_\tau^{y_\tau}$, and thus has a cost of $\gamma_i y_i$ in $T_{\overline{f}}$. The backward edge leaves state $s_\tau^{y_\tau}$; therefore the edge has cost $\gamma_i y_i$ in T, and cost $-\gamma_i y_i$ in $T_{\overline{f}}$.

For the remainder of the proof, we will show that for all $i \in \{\sigma, \ldots, \tau - 1\}$, the cost of the edge (g_i, g_{i+1}) is equal to the sum of the cost of the two edges of $\nu(\sigma, \tau)$ in trellis segment i.

At each trellis segment i where $\sigma \leq i < \tau$, the cycle $\nu(\sigma, \tau)$ in $T_{\overline{f}}$ consists of a backward edge leaving $s_i^{y_i}$, and a forward edge entering $s_i^{1-y_i}$. The cost of the backward edge in $T_{\overline{f}}$ is $-\gamma_i y_i$ and the cost of the forward edge is $\gamma_i(1 - y_i)$, for a total cost of $\gamma_i(1 - y_i) - \gamma_i y_i$. This is the same as the cost of edge (g_i, g_{i+1}) in $\mu(\sigma, \tau)$, by equation (6.9). ■

In order to deal with an arbitrary circulation in $T_{\overline{f}}$, we will decompose it into simple cycles. It turns out that the cycles of the form $\nu(\sigma, \tau)$ represent every possible simple cycle in $T_{\overline{f}}$, as shown in the following claim.

Claim 6.10 *For every simple cycle C in $T_{\overline{f}}$, $C = \nu(\sigma, \tau)$ for some $1 \leq \sigma < \tau \leq n$.*

Proof: Let s_τ^α be the node on the cycle C that is the "closest" to the last node s_n^0 in the trellis, i.e., τ is maximum among all nodes on C (break ties arbitrarily). Now consider the outgoing edge from s_n^0 on the cycle C; it must not go forward in the trellis, since the node s_τ^α is the node on C that is closest to s_n^0. Therefore, this edge goes backward in trellis, and we may conclude that the node s_τ^α must be on $\overline{P}$ (the only backward edges are those on $\overline{P}$).

Follow the cycle C backward in the trellis along $\overline{P}$, "marking" edges along the way, until it diverges from $\overline{P}$ at some node $s_{\sigma-1}^\gamma$. Since C is a simple cycle, it must connect to s_τ^α via some path that does not traverse a marked edge. The only such path from $s_{\sigma-1}^\gamma$ back to s_τ^α that does not go past $\nu(\sigma, \tau)$ is the path forward in the trellis along the edges that complete the cycle $\nu(\sigma, \tau)$. ■

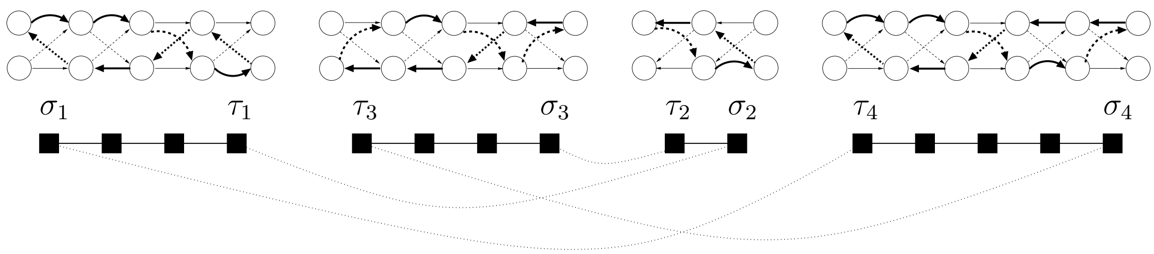


Figure 6-12: A promenade in Θ and its corresponding circulation in $T_{\overline{f}}$. The promenade consists of the Hamiltonian subpaths $\mathcal{U} = \{\mu(\sigma_1, \tau_1), \ldots, \mu(\sigma_4, \tau_4)\}$, and the circulation consists of the cycles $\mathcal{W} = \{\nu(\sigma_1, \tau_1), \ldots, \nu(\sigma_4, \tau_4)\}$.

Promenades in Θ and agreeable circulations in $T_{\overline{f}}$. We are now ready to complete the proof of Theorem 6.2. We use our correspondence between paths in Θ and cycles in $T_{\overline{f}}$ to show a correspondence between promenades in Θ and agreeable circulations in $T_{\overline{f}}$.

In the definitions of $\mu(\sigma, \tau)$ and $\nu(\sigma, \tau)$, we have assumed $\tau > \sigma$. For $\tau < \sigma$, we let $\mu(\sigma, \tau) = \mu(\tau, \sigma)$ and $\nu(\sigma, \tau) = \nu(\tau, \sigma)$. We will use the notation $\lambda\nu$ to represent a circulation in T_f that sends λ units of flow around a residual cycle ν. For a set $\mathcal{W}$ of cycles ν in $T_{\overline{f}}$, we say that $\lambda\mathcal{W}$ is the circulation $\sum_{\nu \in \mathcal{W}} \lambda\nu$.

Lemma 6.11 *If there is a promenade in Θ with negative cost, there is an agreeable circulation in $T_{\overline{f}}$ with negative cost.*

Proof: Let Ψ be a promenade in Θ with negative cost. Let $1/\lambda$ be the maximum number of occurrences of any single Hamiltonian edge in Ψ. If the matching edges along Ψ are removed, what remains is a multiset of subpaths of the Hamiltonian path of Θ, of the form $\mu(\sigma, \tau)$. Let $\mathcal{U} = \{\mu(\sigma_1, \tau_1), \ldots, \mu(\sigma_{|\mathcal{U}|}, \tau_{|\mathcal{U}|})\}$ be this set of subpaths, ordered by their occurrence during a traversal of Ψ. In other words, one may traverse Ψ by following the path $\mu(\sigma_1, \tau_1)$, matching edge $(g_{\tau_1}, g_{\sigma_2})$, path $\mu(\sigma_2, \tau_2)$, ..., path $\mu(\sigma_{|\mathcal{U}|}, \tau_{|\mathcal{U}|})$, and finally matching edge $(g_{\tau_{|\mathcal{U}|}}, g_{\sigma_1})$. An example is shown in Figure 6-12. Note that $\sigma_j > \tau_j$ is possible, as is the case for the paths in Figure 6-12. (Here we use the fact that $\mu(\sigma, \tau) = \mu(\tau, \sigma)$.)

Let $\mathcal{W} = \{\nu(\sigma_1, \tau_1), \ldots, \nu(\sigma_{|\mathcal{U}|}, \tau_{|\mathcal{U}|})\}$ be the set of corresponding cycles in $T_{\overline{f}}$. Figure 6-12 shows the set $\mathcal{W}$ corresponding to a particular promenade with subpaths $\mathcal{U}$. Consider the circulation $\lambda\mathcal{W}$ in $T_{\overline{f}}$, obtained by sending λ units of flow around every cycle in $\mathcal{W}$. No edge in $\lambda\mathcal{W}$ has more than one unit of flow sent across it by the definition of λ, so $\lambda\mathcal{W}$ is a feasible circulation in $T_{\overline{f}}$. (Note that we have not yet shown that $\lambda\mathcal{W}$ is agreeable, just that it is a circulation in conventional sense.)

Since all the matching edges of Θ have zero cost, and Ψ has negative cost, we have

$$c[\Psi] = \sum_{j=1}^{|\mathcal{U}|} c[\mu(\sigma_j, \tau_j)] < 0.$$

Therefore by Claim 6.9, we have

$$c[\mathcal{W}] = \sum_{j=0}^{|\mathcal{W}|} c[\nu(\sigma_j, \tau_j)] < 0.$$

It remains to show that $\lambda\mathcal{W}$ is agreeable. Consider the first cycle in $\mathcal{W}$, namely $\nu(\sigma_1, \tau_1)$. The only agreeability constraints violated by sending flow around this cycle are at trellis segments σ_1 and τ_1; all segments between σ_1 and τ_1 have the same amount of flow on input-`1` edges as they did before. This can be seen clearly in the example of Figure 6-11: At trellis segment 5, flow shifts from an input-`1` edge to an input-`0` edge; at segment 8, flow shifts from an input-`0` edge to an input-`1` edge; at segments 6 and 7, however, flow shifts among the same type of edge.

Consider building $\lambda\mathcal{W}$ by adding the flows $\lambda\nu(\sigma_1, \tau_1), \lambda\nu(\sigma_2, \tau_2), \ldots$ in order. As we argued above, after adding $\lambda\nu(\sigma_1, \tau_1)$ to $\lambda\mathcal{W}$, we have that agreeability constraints are violated at segments σ_1 and τ_1. When we add $\lambda\nu(\sigma_2, \tau_2)$, we "fix" the agreeability at segment τ_1, and violate the agreeability at τ_2; this is implied by the agreeability of $\overline{P}$, by the following argument. By the definition of Ψ, we have that $\{\tau_1, \sigma_2\} \in \mathcal{X}$. Since $\overline{P}$ is agreeable, the graph $T_{\overline{f}}$ has the same type (either input-`0` or input-`1`) of backward edge at segments τ_1 and σ_2. Therefore, routing λ units of flow around *both* $\nu(\sigma_1, \tau_1)$ and $\nu(\sigma_2, \tau_2)$ preserves the agreeability constraint for $\{\tau_1, \sigma_2\}$.

We can see an example of this in Figure 6-12. Consider the cycle $\nu(\sigma_1, \tau_1)$ in Figure 6-12. When we send λ units of flow around $\nu(\sigma_1, \tau_1)$, the total flow on input-`1` edges decreases by λ at segment τ_1. Consider the segment σ_2; since $\{\tau_1, \sigma_2\} \in \mathcal{X}$, it must be the case that $\overline{P}$ uses the same type of edge (in this case input-`1`) at segments τ_1 and σ_2. Therefore, the flow $\lambda\nu(\sigma_2, \tau_2)$ must also decrease the total flow on input-`1` edges by λ at segment σ_2.

By induction, we have that the flow $\lambda\{\nu(\sigma_1, \tau_1), \nu(\sigma_2, \tau_2), \ldots, \nu(\sigma_j, \tau_j)\}$ affects only the agreeability constraints at segments σ_1 and τ_j. Thus in the flow $\lambda\mathcal{W}$, we need only consider the segments σ_1 and $\tau_{|\mathcal{W}|}$. However, we have that $\{\sigma_1, \tau_{|\mathcal{W}|}\} \in \mathcal{X}$; thus by the previous argument, the agreeability constraint is maintained, and we have that $\lambda\mathcal{W}$ is agreeable. ■

Lemma 6.12 *If all promenades in Θ have positive cost, all agreeable circulations in $T_{\overline{f}}$ have positive cost.*

Proof: Suppose f' is an agreeable circulation in $T_{\overline{f}}$ where $c[f'] < 0$. We may assume wlog that $f' = f - \overline{f}$ for some vertex (f, x) of the polytope for RALP; otherwise there is some other circulation with less cost. This implies that all flow values in f' are rational, since all vertices of polytopes are rational. Therefore, there is a positive rational value $\lambda \leq 1$ such that every flow value is an integer multiple of λ. It follows that we may construct a cycle decomposition of f' where every cycle has λ units of flow around it. Let $\mathcal{W}$ be the collection of cycles in this decomposition, and so $f' = \lambda\mathcal{W}$. By Claim 6.10, every cycle in $T_{\overline{f}}$ is of the form $\nu(\sigma, \tau)$. So, $\mathcal{W}$ is a multiset of cycles of the form $\nu(\sigma, \tau)$.

Let B_i be the set of cycles in $\mathcal{W}$ that begin or end at segment i. Formally, for all $i \in \{1, \ldots, n\}$, we have

$$B_i = \{\nu \in \mathcal{W} : \nu = \nu(\sigma, \tau) \text{ where } \sigma = i \text{ or } \tau = i\}.$$

Recall that Each cycle $\nu(\sigma, \tau)$ only affects the agreeability constraints at segments σ and τ. Let x_t be the information bit such that $\sigma \in X_t$. If $x_t = \mathtt{0}$, then $T_{\overline{f}}$ has a backward input-0 edge at segment σ, and thus $\lambda\nu$ increases the flow on input-1 edges at segment σ by λ. If $x_t = \mathtt{1}$, then $\lambda\nu$ decreases the flow on input-1 edges at segment σ by λ. We conclude that at all segments i, where $i \in X_t$, the flow on input-1 edges changes by

$$\lambda|B_i| \ \text{ if } \ x_t = \mathtt{0}; \qquad -\lambda|B_i| \ \text{ if } \ x_t = \mathtt{1}. \tag{6.10}$$

Consider some $\{i, i'\} \in \mathcal{X}$. Since f' is agreeable, then the flow on input-1 edges at i and i' must change by the same amount when we send flow around f'. Therefore, equation (6.10) implies that for all $X_t \in \mathcal{X}$ where $X_t = \{i, i'\}$, we have $|B_i| = |B_{i'}|$.

For all $\{i, i'\} \in \mathcal{X}$, create a one-to-one correspondence between the members of B_i and $B_{i'}$ (we can do this because the sets are the same size). This correspondence will be used to define the matching edges of our promenade Ψ in Θ.

Create an auxiliary multigraph H with a node $v[\nu]$ for each $\nu \in \mathcal{W}$. Add edges according to the correspondence we just created between B_i and $B_{i'}$, for each $\{i, i'\} \in \mathcal{X}$. Each $\nu \in \mathcal{W}$ where $\nu = \nu(\sigma, \tau)$ is in exactly two sets: B_σ and B_τ; therefore each node $v[\nu]$ has exactly 2 incident edges. Note that if we have a cycle $\nu \in \mathcal{W}$ where $\nu = \nu(i, i')$ for some $\{i, i'\} \in \mathcal{X}$, then ν would be in both B_i and $B_{i'}$. In this case, the correspondence may assign $v[\nu]$ to itself; we represent this situation by a self-loop on $v[\nu]$.

Since H is 2-regular, it is a collection $\mathcal{Y}$ of simple undirected cycles (where a node with a self-loop is considered a cycle). For a cycle $Y \in \mathcal{Y}$, let $\mathcal{W}_Y$ be the set of (trellis) cycles that correspond to nodes in $\mathcal{Y}$; formally,

$$\mathcal{W}_Y = \{\nu \in \mathcal{W} : v[\nu] \in Y\}.$$

The set $\{\mathcal{W}_Y : Y \in \mathcal{Y}\}$ constitutes a partition of the cycles of $\mathcal{W}$ into subsets. The total cost of the flow $f' = \lambda\mathcal{W}$ is negative, so there must be at least one cycle $Y^* \in \mathcal{Y}$ such that the flow $\lambda\mathcal{W}_{Y^*}$ has negative cost.

We build a promenade Ψ in Θ by following the cycle Y^*. We use

$$(v[\nu(\sigma_1, \tau_1)], v[\nu(\sigma_2, \tau_2)], \ldots, v[\nu(\sigma_{|Y^*|}, \tau_{|Y^*|})])$$

to denote a traversal in Θ of the cycle Y^*. By definition of H, we have $\{\tau_i, \sigma_{i+1}\} \in \mathcal{X}$ for all $i \in \{1, \ldots, |Y^*| - 1\}$, and $\{\tau_{|Y^*|}, \sigma_1\} \in \mathcal{X}$. Therefore, the collection of paths

$$\mathcal{U} = (\mu(\sigma_1, \tau_1), \mu(\sigma_2, \tau_2), \ldots, \mu(\sigma_{|Y^*|}, \tau_{|Y^*|}))$$

forms a promenade Ψ by adding the matching edges between each consecutive subpath. Matching edges have zero cost, so

$$c[\Psi] = \sum_{i=1}^{|Y^*|} c[\mu(\sigma_i, \tau_i)].$$

Since the cost of $\lambda \mathcal{W}_{Y^*}$ is negative (by definition of Y^*), we have

$$\sum_{i=1}^{|Y^*|} c[\nu(\sigma_i, \tau_i)] < 0.$$

By Claim 6.9, we have

$$c[\Psi] = \sum_{i=1}^{|Y^*|} c[\mu(\sigma_i, \tau_i)] = \sum_{i=1}^{|Y^*|} c[\nu(\sigma_i, \tau_i)] < 0$$

Thus Ψ is a negative-cost promenade, and we have a contradiction. ■

Theorem 6.2 is implied by Lemmas 6.7, 6.8, 6.11 and 6.12. ■

6.3.5 Combinatorial Characterization for RA(R) Codes

In this section we give the generalization of Theorem 6.2 to the case of RA(R) codes, for any positive integer $R \geq 2$.

We will define an auxiliary graph Θ as in RA(2) codes, and claim that LP decoding succeeds if and only if this graph does not contain a certain negative-cost subgraph. However, in the general case, the graph Θ will be a *hypergraph*; i.e., it may include *hyperedges* that contain more than two nodes.

Let Θ be a line graph on n vertices $(g_1, \ldots, g_n)$, as in RA(2) codes, where the graph includes $n-1$ Hamiltonian edges between consecutive nodes in the line. Note that these edges are "normal" (non-hyper) edges. In RA(2) codes, we defined a matching edge for all $\{i, i'\} \in \mathcal{X}$. In the general RA($R$) case, we define an R-hyperedge in Θ for all $X_t \in \mathcal{X}$. This edge consists of exactly the nodes $\{g_i : i \in X_t\}$, where $X_t = \{i^1, i^2, \ldots, i^R\}$. We still refer to these edges as matching edges, since they form an R-dimensional perfect matching among the nodes.

The costs of the edges in the graph Θ are exactly the same as in the RA(2) case. We have $c[g_i, g_{i+1}] = \gamma_i(1 - y_i) - \gamma_i y_i$, where y is the transmitted codeword, and γ_i is the log-likelihood ratio of code bit i, as defined in Section 2.5. The matching edges all have zero cost.

We must define the generalization of a promenade in order to state the generalization to Theorem 6.2. We define a *hyperpromenade* Ψ as follows. A hyperpromenade Ψ is a set of subpaths of the form $\mu(\sigma, \tau)$ (as defined in Section 6.3.4), possibly with multiple copies of the same subpath in the set. We further require the the set Ψ satisfies a certain "agreeability" constraint. Formally, we define, for each segment i

in the trellis where $1 \leq i \leq n$, the following multiset B_i:

$$B_i = \{\mu \in \Psi : \mu = \mu(\sigma, \tau), \text{ where } i = \sigma \text{ or } i = \tau\}$$

Note that if multiple copies of some $\mu(\sigma, \tau)$ exist in Ψ, then B_i contains multiple copies as well. We comment that these sets are similar to the sets B_i defined in the proof for Lemma 6.12. We say that Ψ is a hyperpromenade if, for all $X_i \in \mathcal{X}$ where $X_i = \{i^1, i^2, \ldots, i^R\}$, we have

$$|B_{i^1}| = |B_{i^2}| = \cdots = |B_{i^R}|.$$

The cost of a subpath $\mu(\sigma, \tau)$ is as before; the sum of the costs of its edges. The cost of a hyperpromenade is equal to the sum of the costs of the subpaths it contains. Now, we may state the generalization to Theorem 6.2:

Theorem 6.13 *For an arbitrary RA(R) code, the RALP decoder succeeds if all hyperpromenades have positive cost. The RALP decoder fails if there is a hyperpromenade with negative cost.*

This theorem can be proved using the same arguments used for the proof of Theorem 6.2. The interesting question is whether we can construct graphs Θ to derive good interleavers for RA(R) codes. We would need to show that the graph Θ has a very small probability of having a negative-cost hyperpromenade.

Chapter 7

Comparison with Message-Passing Algorithms

Explaining the superb performance of message-passing decoders on turbo codes and LDPC codes has been the subject of intense study since the introduction of turbo codes in 1993 [BGT93]. In this chapter we define the general framework for message-passing decoders and give the details for two common decoders used in practice. We show how these algorithms apply to both turbo codes and LDPC codes. We cover the standard method used for decoding both turbo codes and LDPC codes: the *sum-product* algorithm (also known as *belief propagation* [MMC98]). The other message-passing algorithm we cover is the *min-sum* algorithm [Wib96], which has received some attention due to its simplicity (although it does not perform as well as sum-product). Message-passing algorithms can be regarded as generic procedures for inference on a graphical model [YFW02, Wai02], and have applications outside of coding theory; these applications include Bayesian networks [Pea88], computer vision [GG84] and statistical physics [EA75].

After reviewing these methods, we compare the performance of LP decoding to these decoders, both analytically and experimentally. We show that LP decoding is equivalent to message-passing decoding in several cases. For the case of tailbiting trellises, we can show that the success conditions of LP decoding are equivalent to those introduced by Forney et al. [FKMT01]. In the binary erasure channel (BEC), belief propagation fails if and only if a "stopping set" exists in the factor graph among the bits erased by the channel. In this chapter we show that the pseudocodewords we defined for LDPC codes are exactly stopping sets. Thus, the performance of the LP decoder is equivalent to belief propagation on the BEC. Our notion of a pseudocodeword also unifies other known results for particular cases of codes and channels. In cycle codes, our pseudocodewords are identical to the "irreducible closed walks" of Wiberg [Wib96]. Also, when applied to the analysis of computation trees for min-sum decoding, pseudocodewords have a connection to the *deviation sets* defined by Wiberg [Wib96], and refined by Forney et. al [FKKR01].

The sum-product algorithm is known to perform very well in practice [DJM98]. However, it will not always converge; additionally, when the algorithm finds a codeword, there is no theoretical guarantee that it has found the ML codeword, i.e., it does not have the ML certificate property enjoyed by the LP decoder. It should be acknowledged, however, that in practice (for LDPC codes with large block length), when the sum-product decoder converges to a codeword, it is extremely rare for it not to be the ML codeword [For03]. In Chapter 8 we will see how to derive new message-passing algorithms that are guaranteed to converge to the LP optimum. If one of these algorithms converges to a codeword, it is guaranteed to be the ML codeword.

We note that Gallager's original decoding algorithm [Gal62] and Sipser and Spielman's algorithm for decoding expander codes [SS96] also fall under the category of message-passing decoders, but we do not discuss them here. It would be interesting to see how these algorithms compare to LP decoding.

The results in this chapter are joint work with David Karger and Martin Wainwright, and most of them have been appeared previously, or have been submitted for publication [FWK03a, FWK03b, FK02a, FK02b].

Notes on computational complexity. Comparing the computational complexity of LP decoding and message-passing decoding is still very much an open issue, both in theory and practice. The primary goal in this study of LP decoding is not necessarily to derive an algorithm that will compete computationally with message-passing decoders in practice, but rather to derive an efficient (polynomial time) algorithm whose performance can be understood analytically. Many interesting issues remain open in this area, both in theory and in practice.

Linear programming is considered one of the most complex natural problems that is still solvable in polynomial time (the efficiency of the ellipsoid algorithm grows as $\Theta(n^3)$, with a large constant). So in theory, the LP decoder is far less efficient than the message-passing decoders, most of which run in linear time (for a fixed number of iterations). Intuitively, it would also seem that one would have to "pay" computationally for the ML certificate property [Yed03]. However, using recent work on "smoothed analysis" [ST01], one may be able show that the simplex algorithm has better performance guarantees for this application. Furthermore, there may be more efficient methods for solving the coding LPs than using a general-purpose LP solver. For example, the relaxation for RA(2) codes can be solved using a single instance of min-cost flow. Also, our "pseudocodeword unification" results in this chapter can be seen as showing that message-passing decoders represent efficient methods for solving the coding LP for certain special cases. It would be interesting to see more special cases that yield efficient algorithms, or (ideally) a general-purpose combinatorial algorithm for solving the turbo code and LDPC code relaxations we give in this thesis.

The author has not yet performed an experimental study comparing the efficiency of the two methods, nor is aware of any such work. Due to the variety of implementations of message-passing decoders (and of LP solvers), we hesitate to generalize about the relative efficiency (in practice) of the two methods. It would be very interesting to see a comprehensive study done on this issue.

7.1 Message-Passing Decoders for LDPC Codes

Message-passing decoders for LDPC codes are built on a factor graph for the code. Every edge (i, j) in the factor graph maintains two messages, one going in each direction. We denote the message from variable node i to check node j by m_{ij}. Similarly we use m_{ji} to denote the message from check j to variable i. The messages (in both directions) represent a "belief" regarding the value of the code bit y_i.

The algorithm goes through a series of *updates* on the messages. The order in which messages are updated (and whether in series or parallel) depends on the particular heuristic being used. A message m_{ij} from a variable to check node is updated using some function of the local cost γ_i at variable node i and the messages $\{m_{ji} : j \in N(i)\}$ incoming to i. The messages from check to variable nodes are updated using a function of the messages $\{m_{ij} : i \in N(j)\}$ incoming to j.

The messages are all initialized to some neutral value (typically zero), and updating usually begins with the messages m_{ij} from variables to checks. The iterative message update process continues until some sort of stopping condition is reached, which differs between algorithms and implementations. Then, a final decision is made for each bit. This final decision for bit y_i is a function of the local cost γ_i and the incoming messages $\{m_{ji} : j \in N(i)\}$ to the variable node i.

In the remainder of this section, we give the details of two specific message-passing algorithms, *min-sum* and *sum-product*, as they apply to decoding on a factor graph for a binary linear code.

7.1.1 Min-Sum Decoding

The *min-sum* algorithm is a particular message-passing decoder for LDPC codes. We describe this algorithm in a way that is specific to binary decoding; for its application in a more general setting, we refer the reader to Wiberg's thesis [Wib96]. The min-sum algorithm can be thought of as a dynamic program that *would* compute the ML codeword if the graph were a tree. The graph is usually not a tree, but the this decoder proceeds as if it was.

In this algorithm the messages m_{ij} and m_{ji} on an edge (i, j) represent the cost of setting $y_i = 1$. So, a positive message represents "evidence" that the bit y_i should be set to `0`, and a negative message represents evidence that the bit y_i should be set to `1`. The messages are all initialized to zero. All variable-to-check messages are then updated simultaneously, followed by all check-to-variable messages updated simultaneously. This alternation continues, either for a fixed number of iterations, or until the hard decision procedure yields a codeword.

The message from a variable i to a check j is simply the sum of the local cost γ_i and the messages coming into i (other than the one from j):

$$m_{ij} \;=\; \gamma_i + \sum_{j' \in N(i), j' \neq j} m_{j'i}.$$

For a check-to-variable message update, we use the incoming messages to a check

j as costs, and compute the cost of each configuration ($S \in E_j$). Then, for each bit i in the neighborhood of j, the message to i is the difference between the minimum cost configuration S where $i \in S$ (i.e., bit y_i is set to 1) and the minimum cost configuration S where $i \notin S$ (i.e., bit y_i is set to 0). Thus the sign of the message determines whether the check "believes" that the bit is either a 0 or a 1. Specifically:

$$m_{ji} = \min_{S \in E_j, S \ni i} \left(\sum_{i' \in S, i' \neq i} m_{i'j} \right) - \min_{S \in E_j, S \not\ni i} \left(\sum_{i' \in S} m_{i'j} \right)$$

Note that when we update a message to a node (either variable or check), the message *from* the node that will be receiving the new message is not factored into the calculation. Intuitively, this is done because the node already "knows" this information.

The hard decision for a bit y_i is made by summing its incoming messages along with its cost, and taking the sign. Formally, if we set $\alpha_i = \gamma_i + \sum_{j \in N(i)} m_{ji}$, then the hard decision for each bit y_i is made by setting $y_i = 0$ if $\alpha_i > 0$, and $y_i = 1$ otherwise.

7.1.2 Sum-Product (Belief Propagation)

The *sum-product*, or *belief propagation* algorithm can also be thought of as a dynamic program in the case where the factor graph is a tree. Here, the goal is to compute the exact *marginal probabilities* $\Pr[\ \tilde{y} \mid y_i = 0\ ]$ and $\Pr[\ \tilde{y} \mid y_i = 1\ ]$ for each bit y_i. We begin with the local probabilities $\Pr[\ \tilde{y}_i \mid y_i = 0\ ]$ and $\Pr[\ \tilde{y}_i \mid y_i = 1\ ]$. Note that for a particular bit y_i, the local probabilities are based only on the received symbol $\tilde{y}_i$ and the channel model being used. For example, in the BSC, we have

$$\Pr[\ \tilde{y}_i \mid y_i = 0\ ] = \begin{cases} p & \text{if } \tilde{y}_i = 1 \\ 1 - p & \text{if } \tilde{y}_i = 0 \end{cases}.$$

The messages to and from variable node i are pairs (m^0, m^1) that represent an estimate of the distribution on the marginal probabilities $\Pr[\ \tilde{y} \mid y_i\ ]$ over settings of y_i. The variable-to-check message m^0_{ij} is computed by multiplying the incoming messages $m^0_{j'i}$ (for all $j' \in N(i)$, where $j' \neq j$) together with the local probability $\Pr[\ \tilde{y}_i \mid y_i = 0\ ]$. Message m^1_{ij} is updated similarly.

The update on the check-to-variable message (m^0_{ji}, m^1_{ji}) is performed by first computing an estimate (based on the incoming messages) of the probability distribution on local configurations. Then, for all configurations where $y_i = 0$, these probabilities are added together to obtain m^0_{ji}. (A similar update is performed for m^1_{ji}.)

We define the message update rules formally as follows. All messages m_{ij} and m_{ji} are initialized to $(1, 1)$. To perform an update, variable node i multiplies all its incoming messages (besides the one from j) together with the local probability, and sends the result to j:

$$m^0_{ij} = \Pr[\ \tilde{y}_i \mid y_i = 0\ ] \prod_{j' \in N(i), j' \neq j} m^0_{ji} \qquad m^1_{ij} = \Pr[\ \tilde{y}_i \mid y_i = 1\ ] \prod_{j' \in N(i), j' \neq j} m^1_{ji}$$

When a check node j sends a message to a variable node i, it sums up all the probabilities of its local configurations for each setting of the bit i, based on the incoming messages other than i. The updates are as follows:

$$m_{ji}^0 = \sum_{\substack{S \in E_j, \\ S \not\ni i}} \left(\prod_{\substack{i' \in S, \\ i' \neq i}} m_{i'j}^1 \right) \left(\prod_{\substack{i' \notin S, \\ i' \neq i}} m_{i'j}^0 \right) \qquad m_{ji}^1 = \sum_{\substack{S \in E_j, \\ S \ni i}} \left(\prod_{\substack{i' \in S, \\ i' \neq i}} m_{i'j}^1 \right) \left(\prod_{\substack{i' \notin S, \\ i' \neq i}} m_{i'j}^0 \right)$$

A hard decision is made by multiplying all the incoming messages to a variable node along with the local probability. Formally, we set

$$\alpha_i^0 = \Pr[\, \tilde{y}_i \mid y_i = 0 \,] \prod_{j \in N(i)} m_{ij}^0, \text{ and} \qquad \alpha_i^1 = \Pr[\, \tilde{y}_i \mid y_i = 1 \,] \prod_{j \in N(i)} m_{ij}^1.$$

Then, if $\alpha_i^0 > \alpha_i^1$, we set $y_i = 0$; otherwise we set $y_i = 1$.

If the sum-product algorithm is run on a tree, the intermediate messages represent solutions to marginal probability subproblems on subtrees. If the graph is not a tree, then a particular local probability can contribute to the value of an intermediate message more than once, since the messages could have traversed a cycle. However, the algorithm still seems to perform well in practice, even in this case. Explaining exactly what the sum-product algorithm computes is a topic of great interest [Wib96, YFW02, Wai02].

7.2 Message-Passing Decoders for Turbo Codes

Turbo codes consist of a group of concatenated convolutional codes, each of which can be easily decoded with the Viterbi algorithm. So, the first algorithm that comes to mind for decoding Turbo codes is the following. First, decode each constituent convolutional code separately; if the results agree, then output the resulting codeword. However, in many cases the results will not agree; in fact, the real power of turbo codes comes from this case. (Otherwise, turbo codes would be no different than convolutional codes.)

Message-passing decoders recover from this case by passing messages between the trellises, and adjusting edge costs based on these messages. The messages indicate the "evidence" each trellis has for a bit being set to a particular value.

There are many different types of message-passing schemes; for a full review, we refer the reader to a textbook written on the subject [VY00]. Here we review two common approaches, the *min-sum* and *sum-product* algorithm, as they apply to rate-1/2 repeat-accumulate (RA(2)) codes.

Recall that in RA(2) codes, for an information bit x_t, we have $X_t = \{i, i'\}$, where i and i' are the two layers of the trellis that use information bit t to determine the next transition. The message-passing algorithm maintains a "likelihood ratio" LR_i for each trellis segment i, where $i \in X_t$ for some information bit x_t. This ratio is an *estimate* of the following quantity:

$$\frac{\Pr[x_t = 1]}{\Pr[x_t = 0]}.$$

In other words, the algorithm maintains two separate estimates of the likelihood of x_t; the trick will be to get them to agree. Note that if $\mathrm{LR}_i > 1$ then x_t is more likely to be a one; if $\mathrm{LR}_i < 1$, then x_t is more likely to be a zero.

For each edge e in the trellis at segment i, we have a weight ω_e. This weight will be equal to the local probability $\Pr[\tilde{y}_i|y_i]$, where y_i is the code bit on the label of e. For example, in the BSC, suppose we have an edge e at trellis segment t that enters state `0`, and so it has a label of `0`. If we receive $\tilde{y}_i = 1$, then we set $\omega = p$; If we receive $\tilde{y}_i = 0$, we set $\omega = 1 - p$, where p is the crossover probability in the channel.

The message passing scheme will repeatedly update the edge weights by maintaining a message m_i to each trellis segment i. The effective weight of an "input-1" edge will be equal to $\hat{\omega}_e = m_i \omega_e$, where i is segment of the trellis in which edge e is contained. The effective weight of an "input-0" edge is equal to its original weight. Let the effective weight $\hat{\omega}_P$ of a path P through the trellis be equal to the product of the effective weights of its edges. The messages are initialized all to 1, so they do not affect the weight.

The high-level algorithm for a single iteration is the following:

- Let $\mathcal{Z}_i$ be the set of paths that use an "input-1" edge at trellis segment i. We update LR_i as follows:

$$\mathrm{LR}_i = \frac{\sum_{P \in \mathcal{Z}_i} \hat{\omega}_P}{\sum_{P \notin \mathcal{Z}_i} \hat{\omega}_P} \tag{7.1}$$

  Note that these LRs can be computed by dynamic programming (also known as the "forward-backward" Viterbi algorithm), without explicitly computing $\hat{\omega}_P$ for all paths P.

- Now update the messages as follows. For each $\{i, i'\} \in \mathcal{X}$, we make new messages m'_i and $m'_{i'}$, where

$$m'_i = \frac{\mathrm{LR}_{i'}}{m_{i'}}, \qquad m'_{i'} = \frac{\mathrm{LR}_i}{m_i} \tag{7.2}$$

- Finally, update the effective weights of each edge. For each "input-`1`" edge e at trellis segment i, let

$$\hat{\omega}_e = m_i \omega_e.$$

Note that these weights are in a multiplicative domain, rather than the additive domain we have been working with for LP decoding. There is a way to perform this algorithm in an additive (log) domain using the log-likelihood ratio, but it is not as natural to present. Also, in the log domain, sum-product can become slow; to perform the additions necessary for equation (7.1), the algorithm needs to exponentiate, add, then take logs. In the *min-sum* algorithm, the LR is approximated using the following quantity:

$$\mathrm{MPLR}_i = \frac{\max_{P \in \mathcal{Z}_i} \hat{\omega}_P}{\max_{P \notin \mathcal{Z}_i} \hat{\omega}_P}$$

This approximation is significantly easier to compute, since in the log domain, the max operator remains, and multiplication turns into addition.

The message-passing algorithm continues either for a fixed number of iterations, or until it has "found" a codeword; i.e., if for all $\{i, i'\} \in \mathcal{X}$, we have agreement between LR_i and $\mathrm{LR}_{i'}$. By agreement, we mean that either LR_i and $\mathrm{LR}_{i'}$ are both greater than 1, or LR_i and $\mathrm{LR}_{i'}$ are both less than 1.

Upon termination, a hard decision is made by multiplying the two LRs together that correspond to each information bit. Formally, for all $X_t \in \mathcal{X}$ where $X_t = \{i, i'\}$, we set the decoded information word as follows; for all t, set

$$x'_t = \begin{cases} 0 & \text{if } \mathrm{LR}_i \mathrm{LR}_{i'} \leq 1 \\ 1 & \text{if } \mathrm{LR}_i \mathrm{LR}_{i'} > 1 \end{cases} .$$

7.3 Success Conditions for Message-Passing Decoders and their Relation to LP Decoding

A natural question to ask is how the pseudocodewords for LP decoding relate to the success conditions for the message-passing algorithms used in practice. Perhaps understanding this relationship can help us understand why these message-passing decoders are so successful. In this section we answer this question for several different specific codes and channel models.

7.3.1 Tailbiting Trellises

Recall that on tailbiting trellises (Section 6.1.4), the LP decoder always finds the minimum-cost circulation in the trellis with unit flow across each layer of the trellis. Each such circulation is a convex combination of simple cycles. Therefore, the LP decoder always finds the minimum-cost simple cycle in the trellis (or a convex combination, in the degenerate case of many simple cycles with the minimum cost).

Suppose the trellis has length k. Then, if the circulation found by the LP decoder is a simple cycle of length k, it corresponds to a codeword; otherwise, it is a fractional solution that contains some cycle of length αk, where α is an integer greater than one. We can think of the cycles in the trellis as "pseudocodewords" in the same way that we did in LDPC codes; the cycles are a superset of the codewords, and the decoder always finds a minimum-cost cycle. The cost of a cycle Y is the total cost of its edges, multiplied by the amount of flow on each edge (which is the same for each edge due to flow conservation). If the cycle has length αk, where $\alpha \geq 1$, it must pass through the first trellis layer exactly α times. Since the LP demands one unit of flow across the first trellis layer, it must be that every edge has flow $1/\alpha$. Thus the cost $c[Y]$ of the cycle Y is exactly

$$c[Y] = \frac{1}{\alpha} \sum_{e \in Y} \gamma_e. \tag{7.3}$$

In their work on tailbiting trellises, Forney et al. [FKMT01] have precisely the same characterization of a pseudocodeword; i.e., cycles through the trellis of some length αk. They show that min-sum decoding will find the pseudocodeword with minimum "weight-per-symbol," which is simply the average cost of the paths through the trellis that make up the cycle. Since there are exactly α paths, their cost is exactly the cost in equation (7.3). We may conclude that on tailbiting trellises, these two algorithms have identical performance.

7.3.2 The Binary Erasure Channel

In the binary erasure channel (BEC), bits are not flipped but rather erased. Consequently, for each bit, the decoder receives either `0`, `1`, or an erasure. If either symbol `0` or `1` is received, then it must be correct. On the other hand, if an erasure (which we denote by `x`) is received, there is no information about that bit. It is well-known [DPR+02] that in the BEC, the message-passing belief propagation (BP) decoder fails if and only if a *stopping set* exists among the erased bits. The main result of this section is that stopping sets are the special case of pseudocodewords on the BEC, and so LP decoding exhibits the same property.

Since the BEC is memoryless and symmetric, we can model the BEC with a cost function γ. As in the BSC, $\gamma_i = -1$ if the received bit $\tilde{y}_i = 1$, and $\gamma_i = +1$ if $\tilde{y}_i = 0$. If $\tilde{y}_i = \mathtt{x}$, we set $\gamma_i = 0$, since we have no information about that bit. Note that under the all-zeros assumption, all the costs are non-negative, since no bits are flipped. Therefore, Theorem 5.6 says that the LP decoder will fail only if there a non-zero pseudocodeword with zero cost.

Let $\mathcal{E}$ be the set of code bits erased by the channel. We define a subset $\mathcal{S} \subseteq \mathcal{E}$ as a *stopping set* if all the checks in the neighborhood $\cup_{i \in \mathcal{S}} N(i)$ of $\mathcal{S}$ have degree at least two with respect to $\mathcal{S}$ (see Figure 7-1). In the following statement, we have assumed that both the message-passing and the LP decoders fail when the answer is ambiguous. For the message-passing algorithm, this ambiguity corresponds to the existence of a stopping set; for the LP decoder, it corresponds to a non-zero pseudocodeword with zero-cost, and hence multiple optima for the LP.

Theorem 7.1 *In the BEC, there is a non-zero pseudocodeword with zero cost if and only if there is a stopping set. Therefore, the performance of LP and BP decoding are equivalent in the BEC.*

Proof: *If there is a zero-cost pseudocodeword, then there is a stopping set.* Let (h, u) be a pseudocodeword where $\sum_i \gamma_i h_i = 0$. Let $\mathcal{S} = \{i : h_i > 0\}$. Since all $\gamma_i \geq 0$, we must have $\gamma_i = 0$ for all $i \in \mathcal{S}$; therefore $\mathcal{S} \subseteq \mathcal{E}$.

Suppose $\mathcal{S}$ is not a stopping set; then $\exists j' \in (\cup_{i \in \mathcal{S}} N(i))$ where check node j' has only one neighbor i' in $\mathcal{S}$. By the definition of a pseudocodeword, we have $h_{i'} = \sum_{S \in E_{j'}, S \ni i'} u_{j',S}$. Since $h_{i'} > 0$ (by the definition of $\mathcal{S}$), there must be some

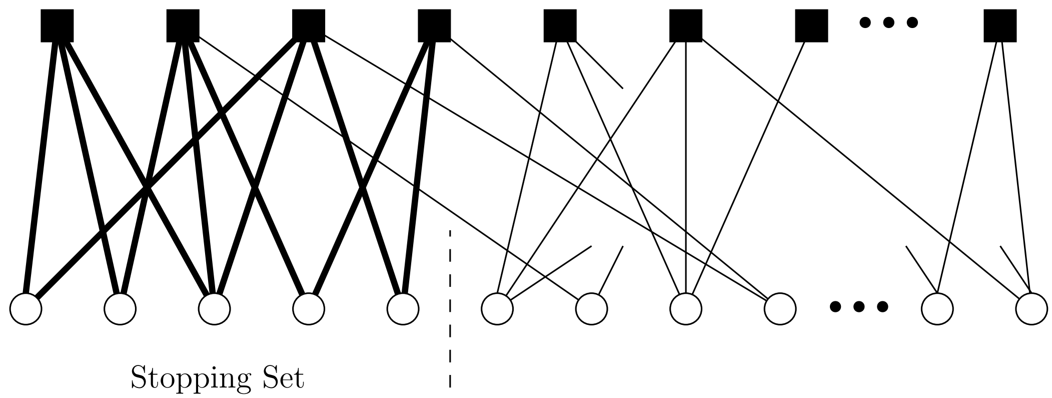


Figure 7-1: A *stopping set* for the sum-product decoder in the BEC. A set $\mathcal{S}$ of variable nodes is a stopping set if all their corresponding bits are erased by the channel, and the checks in the neighborhood of $\mathcal{S}$ all have degree at least two with respect to $\mathcal{S}$.

$S' \in E_{j'}, S' \ni i'$ such that $u_{j',S'} > 0$. Since S' has even cardinality, there must be at least one other code bit i'' in S', which is also a neighbor of check j'. We have $h_{i''} \geq u_{j',S'}$ by the definition of pseudocodeword, and so $h_{i''} > 0$, implying $i'' \in \mathcal{S}$. This contradicts the fact that j' has only one neighbor in $\mathcal{S}$.

If there is a stopping set, then there is a zero-cost pseudocodeword. Let $\mathcal{S}$ be a stopping set. Construct pseudocodeword (h, u) as follows. For all $i \in \mathcal{S}$, set $h_i = 2$; for all $i \notin \mathcal{S}$, set $h_i = 0$. Since $\mathcal{S} \subseteq \mathcal{E}$, we immediately have $\sum_i \gamma_i h_i = 0$.

For a check j, let $M(j) = N(j) \cap \mathcal{S}$. For all $j \in (\cup_{i \in \mathcal{S}} N(i))$ where $|M(j)|$ even, set $u_{j,M(j)} = 2$. By the definition of a stopping set, $M(j) \geq 2$, so if $|M(j)|$ is odd, then $M(j) \geq 3$. For all $j \in (\cup_{i \in \mathcal{S}} N(i))$ where $|M(j)|$ odd, let $I = \{i_1, i_2, i_3\}$ be an arbitrary size-3 subset of $M(j)$. If $|M(j)| > 3$, set $u_{j,M(j)\setminus I} = 2$. Set $u_{j,\{i_1,i_2\}} = u_{j,\{i_1,i_3\}} = u_{j,\{i_2,i_3\}} = 1$. Set all other $u_{j,S} = 0$ that we have not set in this process. We have $\sum_{S \in E_j, S \ni i} u_{j,S} = 2 = h_i$ for all $i \in \mathcal{S}, j \in N(i)$. Additionally, $\sum_{S \in E_j, S \ni i} u_{j,S} = 0 = h_i$ for all $i \notin \mathcal{S}, j \in N(i)$. Therefore (h, u) is a pseudocodeword. ■

7.3.3 Cycle Codes

A cycle code is a binary linear code described by a factor graph whose variable nodes all have degree 2. In this case, pseudocodewords consist of a collection of cycle-like structures that we call *promenades*. The connection with the RA(2) case is intentional; RA(2) codes are in fact cycle codes. A promenade is a closed walk through the factor graph that is allowed to repeat nodes, and even traverse edges in different directions, as long as it makes no "U-turns;" i.e., it does not use the same edge twice in a row. Wiberg [Wib96] calls these same structures *irreducible closed walks*.

From Theorem 5.6, we have that the LP decoder fails if and only if there is some pseudocodeword with cost less than or equal to zero. Wiberg proves exactly the same success conditions for irreducible closed walks under min-sum decoding. Thus we may conclude from this connection that min-sum and LP decoding have identical performance in the case of cycle codes.

Even though cycle codes are poor in general, they are an excellent example of when LP decoding can decode beyond the minimum distance. For cycle codes, the minimum distance is no better than logarithmic. However, in Section 6.3.3 we showed that there are cycle codes that give a WER of $n^{-\alpha}$ for any α, requiring only that the crossover probability is bounded by a certain function of the constant α (independent of n).

We note that the results of Section 6.3.3 may be obtained by using the factor graph representation of an RA(2) code, which turns out to be essentially the same as the auxiliary graph Θ. In fact, this turns out to be a simpler proof, since we may invoke Theorem 5.4. However, we feel that the techniques we used in Section 6.3.3 and in the proof of Theorem 6.2 are better suited for the study of more complex turbo codes. In RA(2) codes, the factor graph LP relaxation turns out to be equivalent to the trellis-based relaxation; this will not hold in general.

7.3.4 Min-Sum Decoding of LDPC Codes

The *deviation sets* defined by Wiberg [Wib96], and further refined by Forney et. al [FKKR01] can be compared to pseudocodeword graphs. The *computation tree* of the message-passing min-sum algorithm is a map of the computations that lead to the decoding of a single bit at the root of the tree. This bit will be decoded correctly (assuming the all-zeros word is sent) unless there is a negative-cost "locally-consistent minimal configuration" of the tree that sets this bit to `1`. Such a configuration is called a *deviation set*, or a pseudocodeword.

All deviation sets can be defined as acyclic graphs of the following form. The nodes of D are nodes from the factor graph, possibly with multiple copies of a node. Furthermore,

- All the leaves of D are variable nodes,
- each non-leaf variable node $i \in \mathcal{I}$ is connected to one copy of each check node in $N(i)$, and
- each check node has even degree.

As is clear from the definition, deviation sets are quite similar to pseudocodeword graphs; essentially the only difference is that deviation sets are acyclic. In fact, if you removed the "non-leaf" condition above, the two would be equivalent. In his thesis, Wiberg states:

> *Since the [factor graph] is finite, an infinite deviation cannot behave completely irregularly; it must repeat itself somehow. . . . It appears natural to look for repeatable, or 'closed' structures [in the graph]. . . , with the property that any deviation can be decomposed into such structures.*[Wib96]

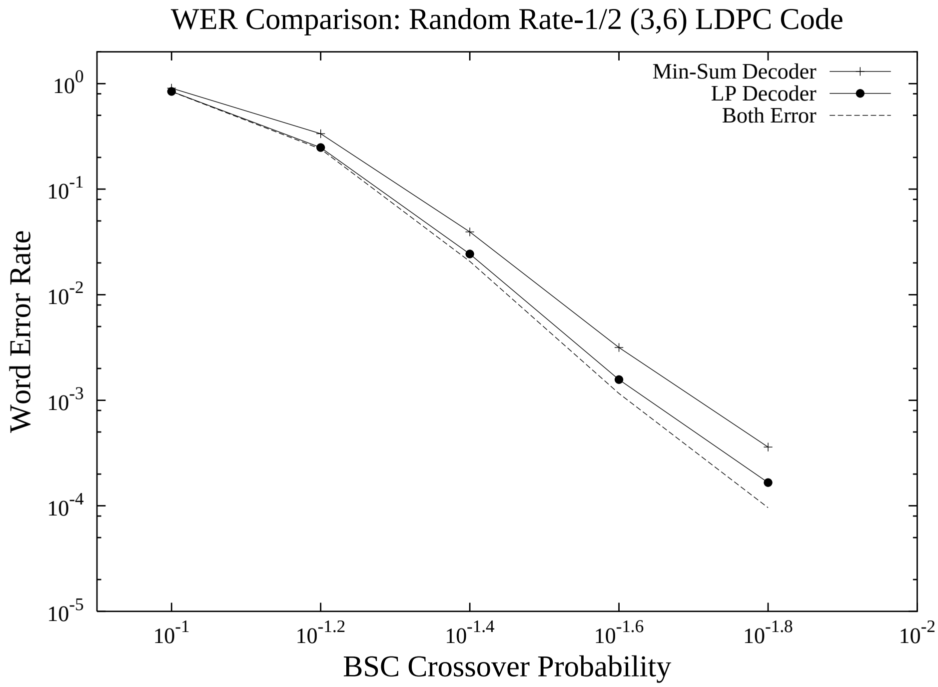


Figure 7-2: A waterfall-region comparison between the performance of LP decoding and min-sum decoding (with 100 iterations) under the BSC using the same random rate-1/2 LDPC code with length 200, left degree 3 and right degree 6. For each trial, both decoders were tested with the same channel output. The "Both Error" curve represents the trials where both decoders failed.

Our definition of a pseudocodeword is the natural "closed" structure within a deviation set. However, an arbitrary deviation set *cannot* be decomposed into pseudocodewords, since it may be irregular near the leaves. Furthermore, as Wiberg points out [Wib96], the cost of a deviation set is dominated by the cost near the leaves, since the number of nodes grows exponentially with the depth of the tree.

Thus strictly speaking, min-sum decoding and LP decoding are incomparable. However, experiments suggest that it is rare for min-sum decoding to succeed and LP decoding to fail (see Figure 7-2). We also conclude from our experiments that the irregular "unclosed" portions of the min-sum computation tree are not worth considering; they more often hurt the decoder than help it.

7.3.5 Tree-Reweighted Max-Product

We have explored the connection between this LP-based approach applied to turbo codes, and the *tree reweighted max-product* message-passing algorithm developed by Wainwright, Jaakkola, and Willsky [Wai02, WJW02]. By drawing a connection to the dual of our linear program, we showed that whenever this algorithm converges to a code word, it must be the ML code word. We give the details of this connection for RA(2) codes in Section 8.1.3.

7.4 Experimental Performance Comparison

While this is primarily a theoretical study of the method of LP decoding, we have done some experimental work to compare the performance of LP decoding with the standard message-passing techniques. For RA(2) codes, we show that the insight gained from the analysis of LP decoding leads to an affirmation of the well-established heuristic already used in code design: to make graphs with large girth. Specifically, we demonstrate that using a graph with high girth not only leads to better performance for LP decoding, but for message-passing decoding as well. This work also shows how the bound from Section 6.3.3 compares to the real observed word error rate.

For LDPC codes, we see that LP decoding on random codes seems to perform better than min-sum, and worse than sum-product (belief propagation). We also see that when compared with ML decoding, the three algorithms have quite similar performance.

7.4.1 Repeat-Accumulate Codes

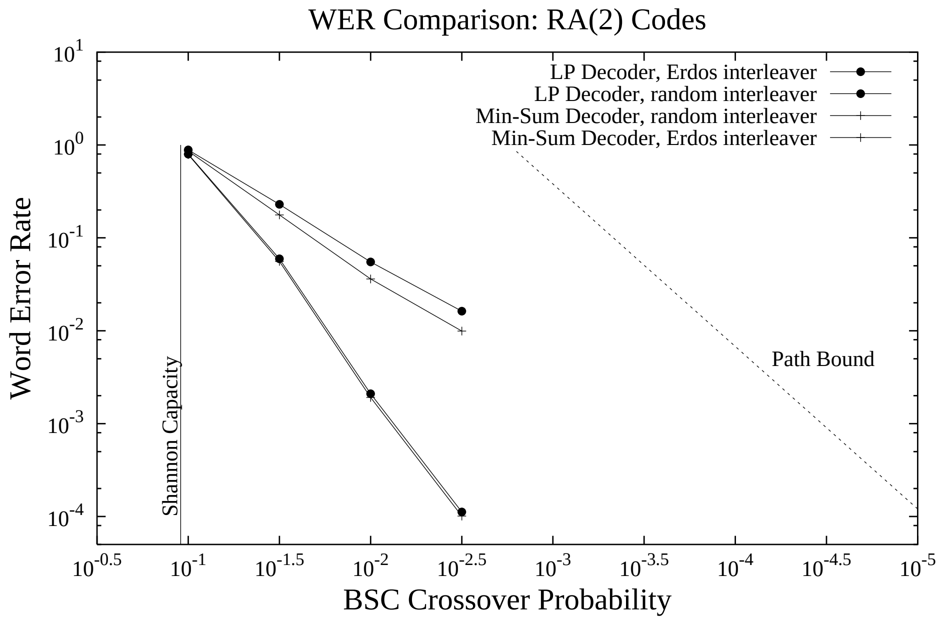
Our experiments on RA(2) codes show that the decoding error of both the sum-product and min-sum algorithms are strongly correlated to the performance of LP decoding under the BSC (see Figure 7-3). Specifically, at a crossover probability of at most $p \leq 10^{-1.5}$, conditioned on the LP decoder succeeding, the observed WER of both other algorithms, regardless of the interleaver, is always less than 1.43×10^{-5}. In contrast, when LP decoding fails, the WER always exceeds 0.6. Since the Erdös interleaver was designed to make negative-cost promenades rare, it is no surprise that the performance of the message-passing decoders improves markedly (see Figure 7-3) when the Erdös interleaver is used.

Our data also show the relationship between the upper bound from Section 6.3.3 and the observed probability of the existence of a negative-cost promenade (see Figure 7-3) for the BSC. The gap between the bound and the observed probability is due to the union bound used in the analysis. The "path bound" of Theorem 6.5 could potentially be improved by a deeper understanding of the distribution of promenades, but the slope of the bound seems quite accurate.

7.4.2 LDPC Codes

We have compared the performance of the LP decoder with the min-sum and sum-product decoders on the binary symmetric channel. We used a randomly generated rate-1/4 LDPC code with left degree 3 and right degree 4. Figure 7-4 shows an error-rate comparison in the waterfall region for a block length of 200. We see that LP decoding performs better than min-sum in this region, but not as well as sum-product.

However, when we compare all three algorithms to ML decoding, it seems that at least on random codes, all three have similar performance. This is shown in Figure 7-5. In fact, we see that LP decoding slightly outperforms sum-product at very low noise levels. It would be interesting to see whether this is a general phenomenon, and whether it can be explained analytically.



	$p = 10^{-1}$		$p = 10^{-1.5}$		$p = 10^{-2}$		$p = 10^{-2.5}$	
	err / *tot*	WER	*err* / *tot*	WER	*err* / *tot*	WER	*err* / *tot*	WER
NCP?	Min-Sum, Erdös interleaver							
Y	787506 / 796648	.989	55817 / 59519	.938	1914 / 2099	.912	101 / 112	.902
N	4 / 203352	2e-5	0 / 940481	0	0 / 997901	0	0 / 999888	0
	Sum-Product, Erdös interleaver							
Y	792176 / 796648	.994	55461 / 59519	.932	1918 / 2099	.914	98/ 112	.875
N	19258 / 203352	.0947	2 / 940481	2e-6	0 / 997901	0	0 / 999888	0
	Min-Sum, random interleaver							
Y	859856 / 886050	.970	176905 / 229738	.770	36153 / 55146	.656	9912 / 16234	.611
N	3 / 113950	3e-5	1 / 770251	1e-6	0 / 944854	0	0 / 983776	0
	Sum-Product, random interleaver							
Y	864673 / 886050	.976	170668 / 229738	.743	35225 / 55146	.639	9935 / 16234	.612
N	12042 / 113950	.106	11 / 770262	1e-5	0 / 944854	0	0 / 983766	0

err= number of errors *tot*= total trials WER = word error rate NCP = neg-cost promenade (incl. 0-cost)

Figure 7-3: Comparison of the WER of some standard message-passing algorithms to the frequency of a negative-cost promenade for both the Erdös interleaver and a random interleaver. The sum-product algorithm is not shown on the plot, since its rate would not be distinguishable from that of the min-sum algorithm. Also included in the plot for reference is the theoretical bound on WER for the Erdös interleaver. This *path bound* is the error bound of Theorem 6.5.
The data show the standard max-product and sum-product message-passing decoding algorithms, using the Erdös interleaver, and using a random interleaver (a new random interleaver is picked for each trial). For each of these four combinations, the data are separated into the case where there is a negative-cost promenade (Y), and when there is not (N). An RA(2) code with block length 128 is used, under the BSC with varying crossover probability p. The data show one million trials for each interleaver type.

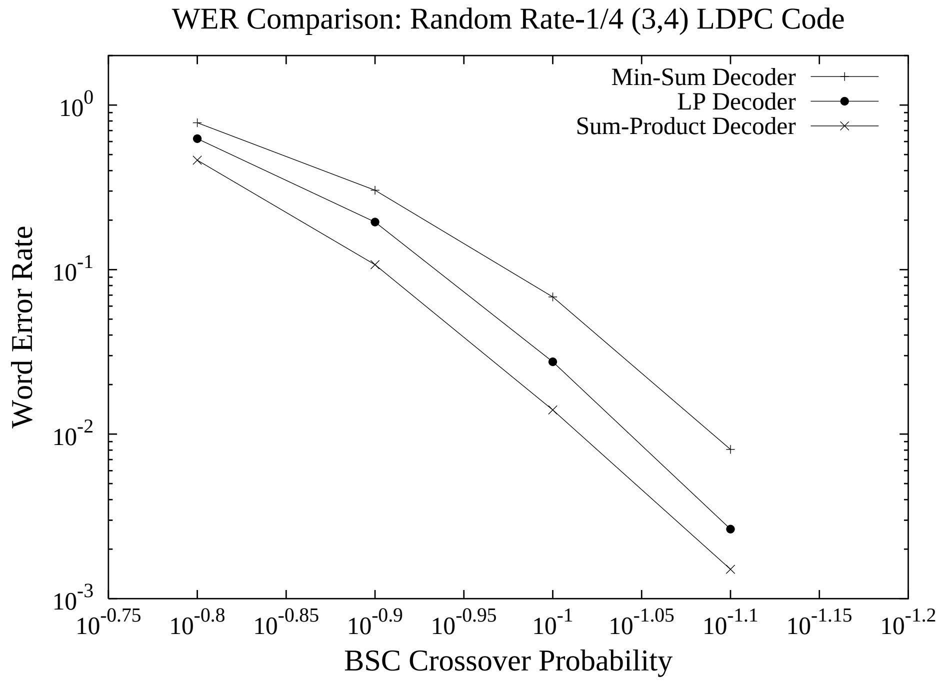


Figure 7-4: A comparison between the performance of LP decoding, min-sum decoding (100 iterations) and belief propagation (100 iterations) under the BSC using the same random rate-1/4 LDPC code with length 200, left degree 3 and right degree 4.

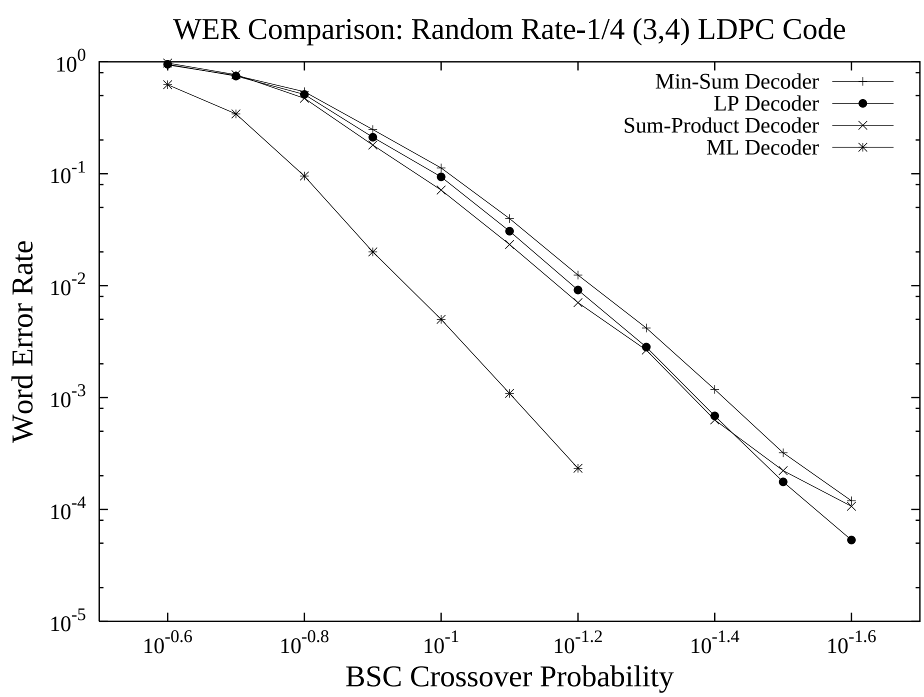


Figure 7-5: A comparison between the performance of ML decoding, LP decoding, min-sum decoding (100 iterations) and belief propagation (100 iterations) under the BSC using the same random rate-1/4 LDPC code with length 60, left degree 3 and right degree 4. The ML decoder is a mixed-integer programming decoder using the LP relaxation.

Chapter 8

New Message-Passing Algorithms Using LP Duality

None of the standard message-passing methods are currently able to give an ML certificate. In this chapter we use LP duality to give message-passing algorithms this ability. We show that if the messages satisfy certain conditions, then the output of the decoder must be the ML codeword. Furthermore, we define a new message-passing decoder to solve the dual of our linear program directly. Therefore all the analytical techniques and performance guarantees given for LP decoding apply to this new message-passing algorithm as well. We first cover rate-1/2 RA codes, then use similar techniques for the case of LDPC codes.

The conditions we impose on the messages will correspond to complementary slackness conditions of the LP. Upon termination, if the messages satisfy these conditions, then an LP optimum is found that corresponds to the ML codeword. This technique has connections to the *tree-reweighted max-product* (TRMP) algorithm of Wainwright et. al [WJW02], which we show to have the ML certificate property.

Note that our result does not guarantee that the algorithm will always converge to the ML codeword; such an algorithm would be unreasonable to expect, since ML decoding is NP-hard. However, our result does give iterative algorithms the power (in some cases) to prove that if they made a decoding error, then the optimal (ML) decoder would have made an error as well.

In addition, we use a partial Lagrangian dual form of our LP to define a new message-passing algorithm based on the *subgradient* method that is guaranteed to converge to a well-understood point: the LP optimum. Thus we get all the performance guarantees of LP decoding, but in an algorithm that more closely matches the ones used in practice. For LDPC codes, this form of the dual also provides a more general means by which *any* message-passing decoder can prove that their output is the ML codeword.

We note that while the subgradient method has theoretical convergence guarantees, it does not necessarily converge in polynomial time; for a polynomial-time guarantee of convergence, we can simply solve the linear program with the Ellipsoid algorithm. However, in practice message-passing algorithms can have a significant running-time advantage, especially in cases where there is little noise in the channel.

On the other hand, subgradient methods are sometimes slower to converge in practice than other methods, and are quite sensitive to a certain "step-size" parameter. It would be interesting to see whether more efficient (perhaps primal-dual) algorithms exist to solve our LPs.

The results in this chapter are joint work with David Karger and Martin Wainwright. The work on turbo codes has appeared in conference form [FKW02]. The work on LDPC codes has been submitted for conference publication [FKW03].

8.1 Rate-1/2 Repeat-Accumulate Codes

In this section, we provide iterative algorithms for solving RALP (Section 6.3.2). We begin by developing a particular Lagrangian dual formulation, the optimal value of which is equivalent to the optimal value of the RALP relaxation. This dual formulation suggests the use of an iterative subgradient optimization method [BT97]. We then consider the variant of the min-sum algorithm, known as tree-reweighted max-product (TRMP), proposed by Wainwright, Jaakkola and Willsky [WJW02] for maximum likelihood calculations on factor graphs. Like the standard min-sum and sum-product algorithms, the algorithm is based on simple message-passing updates, so it has the same complexity per iteration as those algorithms. Here we show that TRMP is also attempting to solve a dual formulation of RALP. In addition, we prove that when TRMP reaches a certain stopping condition, it has found an optimal integral point of RALP, and thus has found the ML code word.

8.1.1 Lagrangian Dual

Without the agreeability constraints, the RALP relaxation (Section 6.3.2) solves a standard shortest path problem. This observation motivates a partial Lagrangian dualization procedure, wherein we deal with the troublesome agreeability constraints using Lagrange multipliers. Recall that $\mathcal{Z}_i$ is the set of paths through the trellis that use an input-`1` edge at segment i. More specifically, for a particular path P, the "agreeability" of that path with respect to some $X_t \in \mathcal{X}$ where $X_t = \{i, i'\}$ can be expressed by the following function A:

$$A_t(P) = [P \in \mathcal{Z}_i] \; - \; [P \in \mathcal{Z}_{i'}] \tag{8.1}$$

Here $[P \in \mathcal{Z}_i]$ is an indicator function that takes the value one if path P is in $\mathcal{Z}_i$, and zero otherwise. Note that $A_t(P) = 0$ for all $X_t = \{i, i'\}$ if and only if P is agreeable. If $A_t(P) \neq 0$, then its sign determines *how* the two segments disagree.

We then consider the Lagrangian obtained by assigning a real-valued Lagrange multiplier λ_t to each agreeability constraint:

$$\mathcal{L}(P; \lambda) \;\; = \;\; \sum_{e \in P} \gamma_e \; + \; \sum_{X_t \in \mathcal{X}} \lambda_t \, A_t(P) \tag{8.2}$$

For a fixed vector $\lambda \in \mathbb{R}^k$, the corresponding value of the dual function $\mathcal{L}^*(\lambda)$ is obtained by minimizing the Lagrangian over all paths — that is,

$$\mathcal{L}^*(\lambda) = \min_{P \in \mathcal{P}} \mathcal{L}(P; \lambda),$$

where $\mathcal{P}$ denotes the set of all paths through the trellis. Since the dual is the minimum of a collection of functions that are linear in λ, it is concave. Moreover, for a fixed λ, calculating the dual value $\mathcal{L}^*(\lambda)$ corresponds to solving a shortest path problem on the trellis, where the input-`1` edges in each paired set of segments $\{i, i'\} = X_t$, have been reweighted by λ_t and $-\lambda_t$ respectively. Since there may not be a unique shortest path, we consider the set $SP(\lambda) = \{P : \mathcal{L}(P; \lambda) = \mathcal{L}^*(\lambda)\}$ of shortest paths under the weighting λ.

Note that the value of an agreeable path is unchanged by the reweighting λ, whereas any non-agreeable path may be affected by λ. This opens up the possibility of "exposing" the ML agreeable path by changing the weighting λ such that all non-agreeable paths have higher cost under λ than the ML agreeable path. Thus, the problem we would like to solve is to find a reweighting λ such that $\mathcal{L}^*(\lambda)$ is maximized. From linear programming duality [BT97], one can show that the optimal value $\max_{\lambda \in \mathbb{R}^k} \mathcal{L}^*(\lambda)$ of this Lagrangian dual problem is equal to the optimal value of the RALP relaxation, so that solving this problem is equivalent to solving RALP.

8.1.2 Iterative Subgradient Decoding

An iterative technique to compute $\max_\lambda \mathcal{L}^*(\lambda)$ is the *subgradient optimization algorithm* [BT97]. Classical gradient ascent/descent methods rely on the fact that the dual function is differentiable. Our dual function $\mathcal{L}^*$ is not differentiable, but it is concave. In this case the *subgradient* is a sufficient substitute for the gradient to ensure convergence. For a concave function $\mathcal{L}^*$, a *subgradient* at λ is a vector d such that $\mathcal{L}^*(\mu) \leq \mathcal{L}^*(\lambda) + d^T(\mu - \lambda)$ for all μ. For the particular form of $\mathcal{L}^*$ at hand, it can be shown [BT97] that the collection of all subgradients is given by the following convex hull:

$$\partial \mathcal{L}^*(\lambda) \quad = \quad \mathrm{CH}\{\ A(P) \mid P \in SP(\lambda)\ \} \tag{8.3}$$

The subgradient method is an iterative method that generates a sequence $\{\lambda^k\}$ of Lagrange multipliers. As previously described, for any pair $X_t = \{i, i'\}$, segment i is reweighted by λ_t^k, whereas segment i' is reweighted by $-\lambda_t^k$. At each iteration k, the algorithm entails choosing a subgradient $d(\lambda^k) \in \partial \mathcal{L}^*(\lambda^k)$, and updating the multipliers via

$$\lambda^{k+1} \quad = \quad \lambda^k + \alpha^k d(\lambda^k), \tag{8.4}$$

where α^k is a scalar representing the "step size" at iteration k. This iterative procedure is guaranteed to converge as long as [BT97]

$$\sum_{k=0}^{\infty} \alpha^k = \infty, \qquad \text{and} \qquad \lim_{k \to \infty} \alpha^k = 0. \tag{8.5}$$

An example [BT97] of such a function is $\alpha^k = 1/(k+1)$. In practice, however, the rate of convergence may be slow. A subgradient $d(\lambda^k)$ can be calculated by finding a shortest path in the reweighted trellis $\mathcal{L}(P, \lambda^k)$.

8.1.3 Iterative TRMP Decoding

In lieu of the subgradient updates, we now consider the TRMP updates [WJW02], which take a simple form for a turbo code with two constituent codes. The TRMP updates are similar to but distinct from standard min-sum updates. Like the subgradient updates of equation (8.4), the TRMP algorithm generates a sequence of Lagrange multipliers $\{\lambda^k\}$. At each iteration, it uses the new trellis weights $\mathcal{L}(P, \lambda^k)$ to compute two shortest paths for each segment i: the shortest path that uses an input-`1` edge in segment i, as well as a competing shortest path that uses an input-`0` edge. As with min-sum decoding, it then uses this information to form an estimate of the min-log likelihood ratio for each trellis segment:

$$\mathrm{LLR}(\lambda^k; i) \quad = \quad \min_{P \in \mathcal{Z}_i} \mathcal{L}(P, \lambda^k) - \min_{P \notin \mathcal{Z}_i} \mathcal{L}(P, \lambda^k) \tag{8.6}$$

The value $\mathrm{LLR}(\lambda^k; i)$ corresponds to the difference between the costs of the (estimated) most likely paths using an input-`1` or an input-`0` edge respectively on segment i. At iteration k, the LLR estimates can be computed efficiently using dynamic programming (also known as the "forward-backward Viterbi algorithm") on the trellis, where λ^k is used to reweight the edges appropriately.

The goal of the iterative decoder is to come to a decision for each information bit, i.e., to make the sign of $\mathrm{LLR}(\lambda^k; i)$ agree with the sign of $\mathrm{LLR}(\lambda^k; i')$ for all $X_t = \{i, i'\}$. With this goal in mind, the Lagrange multipliers are updated via the recursion

$$\lambda_t^{k+1} = \lambda_t^k + \alpha^k \big(\mathrm{LLR}(\lambda^k; i') - \mathrm{LLR}(\lambda^k; i) \big), \tag{8.7}$$

where $\alpha^k \in (0, 1]$ is the step size parameter. Note that, as with the subgradient updates (8.4), the sum of the reweighting factors on segments i and i' is zero at all iterations.

An often used heuristic for standard iterative decoding algorithms is to terminate once thresholding the LLRs yields a valid codeword. With TRMP, we can prove that a heuristic of this form is optimal; specifically, if we terminate when for each information bit, both of the LLRs have the same sign for that bit, then we have found the ML codeword. Formally, call a setting of λ an *agreement* if for all $\{i, i'\} \in \mathcal{X}$, $\mathrm{LLR}(\lambda; i) \cdot \mathrm{LLR}(\lambda; i') > 0$. In other words, $\mathrm{LLR}(\lambda; i)$ and $\mathrm{LLR}(\lambda; i')$ have the same sign, and neither is equal to zero.

Theorem 8.1 *If λ^* is an agreement, then $SP(\lambda^*)$ contains only one path P; furthermore, the path P corresponds to the ML code word.*

Proof: Since $\mathrm{LLR}(\lambda^*; i) \neq 0$ for all segments i, the type of edge (input-`0` or input-`1`) used at each segment by any path in $SP(\lambda^*)$ is determined. It follows that $SP(\lambda^*)$

contains only one path P. Since λ^* is an agreement, we know that for every $X_t \in \mathcal{X}$ where $X_t = \{i, i'\}$, the type of edge used by the shortest path P at segment i matches the type used by P at segment i', i.e., $A_t(P) = 0$. It follows that P is agreeable, and that $\mathcal{L}(P; \lambda^*) = c[P]$. Since $P \in SP(\lambda)$, we have $\mathcal{L}^*(\lambda^*) = \mathcal{L}(P; \lambda^*) = c[P]$.

Now consider the primal solution to RALP that sets $f_e = 1$ for all $e \in P$, and $f_e = 0$ for all $e \in P'$ where $P' \neq P$. The value of this primal solution is $c[P]$. Thus we have exhibited a primal solution P and a dual solution λ^* with the same value $c[P]$. By strong duality, they must both be optimal. Thus P is the ML code word. ■

Corollary 8.2 *If TRMP finds an agreement, then it has found the ML code word.*

We have not yet shown that TRMP always finds an agreement whenever RALP has an integral solution. Consequently, unlike the subgradient algorithm, we cannot assert that the TRMP result is always equivalent to RALP. However, we have observed identical behavior in experiments on RA codes, and further investigation should deepen the connection between these algorithms.

8.2 Low-Density Parity-Check Codes

In this section we define a new class of message-passing algorithms for LDPC codes that use the dual variables of our linear program LCLP. We show that if a message-passing decoder that fits into this class "finds" a codeword, then it must be the ML codeword. We will make the notion of "finding" a codeword precise, and show that upon termination, if a codeword is found, then the message-passing decoder has found a primal and dual optimal solution. Since the primal solution found is a codeword, the dual solution is a proof that this codeword is the ML codeword.

We then define an alternate form of the LP dual using Lagrangian relaxation, and use it to derive a new message-passing algorithm based on the subgradient algorithm [BT97]. We also use this dual form to provide a more general way for a message-passing algorithm to give an ML certificate.

8.2.1 The Dual of $\mathcal{Q}$ for LDPC Codes

Our linear program from Section 5.1, along with its dual, are written below:

$$\begin{array}{rll}
\underline{\text{Primal :}} & \text{minimize} & \sum_i \gamma_i f_i \quad \text{s.t.} \\
& \forall j \in \mathcal{J}, & \sum_{S \in E_j} w_{j,S} = 1 \\
& \forall \text{ edges } (i,j), & f_i = \sum_{S \in E_j, S \ni i} w_{j,S}, \\
& \forall S \in E_j, & w_{j,S} \geq 0, \\
& \forall i \in \mathcal{I}, & f_i \geq 0.
\end{array}
\qquad
\begin{array}{rll}
\underline{\text{Dual :}} & \text{maximize} & \sum_j c_j \quad \text{s.t.} \\
& \forall i \in \mathcal{I}, & \sum_{j \in N(i)} m_{ij} = \gamma_i \\
& \forall j \in \mathcal{J}, S \in E_j, & c_j \leq \sum_{i \in S} m_{ij}
\end{array}$$

Note that we do not need the restrictions $f_i \leq 1$ or $w_{j,S} \leq 1$ in the primal, as they are implied by the other constraints. In the dual, we have free variables m_{ij} for all edges (i,j) in G, and c_j for all $j \in \mathcal{J}$. The notation for the dual variables m_{ij} matches the notation for messages intentionally; these dual variables correspond to messages in the algorithms we derive.

In the above formulation of the dual, we have assumed without loss of generality that the constraint $\sum_{j \in N(i)} m_{ij} \leq \gamma_i$ is tight for all i; if some such constraint were not tight, then we could increase the value of some m_{ij} without affecting the feasibility or the cost of the solution.

8.2.2 Cost-Balanced Message-Passing Algorithms

In this section we derive a class of message-passing algorithms from the dual of our LP. We do not define message update functions; rather we place a small restriction on the outgoing messages from a variable node in order to enforce dual constraints. Thus any decoder that is modified to meet these restrictions fits into our class.

We will regard the dual variables m_{ij} as the messages from variable to check nodes. We make no restrictions on the messages from check to variable nodes. For a particular check j, we let the cost of a configuration S be equal to $\sum_{i \in S} m_{ij}$. Let S_j^- be the minimum-cost configuration $S \in E_j$, with ties broken arbitrarily. Then, we maintain the dual variable c_j equal to the cost of S_j^-. This enforces the dual constraint on c_j. To enforce the dual constraint on m_{ij}, we make the following restriction on the outgoing messages from a variable node i:

$$\sum_{j \in N(i)} m_{ij} = \gamma_i \tag{8.8}$$

This is the only restriction we place on our algorithm. In fact, this is easily achieved by the min-sum or belief propagation algorithm by scaling its outgoing messages. It would be interesting to see how this rescaling affects performance.

We place this restriction in order to say that the message-passing algorithm finds a dual solution; in fact, we need only enforce this restriction at the end of the execution of the algorithm. We call a message-passing algorithm *cost-balanced* if, upon termination of the algorithm, its messages $\{m_{ij}\}$ obey equation (8.8).

8.2.3 ML Certificate Stopping Condition

As a message-passing algorithm progresses, the minimum cost configuration S_j^- for some check j represents the local "best guess" for the settings of the neighbors of j, based on the messages received so far. Suppose, for some code bit i with both j and j' in $N(i)$, we have i in S_j^-, but i not in $S_{j'}^-$. This means that in some sense j and j' "disagree" on the proper setting for i. If there is no such disagreement anywhere in the graph, then the message-passing algorithm has "found" a codeword. The main result of this section is that if a cost-balanced message-passing algorithm is used, then this codeword is the ML codeword.

Formally, for a codeword $y \in \mathcal{C}$ and a configuration $S \in E_j$, we say that y *agrees* with S if, for all $i \in N(j)$, we have $y_i = 1$ if and only if $i \in S$. We say that set of configurations agrees with a codeword y if y agrees with every configuration in the set.

Theorem 8.3 *Suppose a cost-balanced message-passing algorithm is executed, and the messages $\{m_{ij} : i \in \mathcal{I}, j \in N(i)\}$ represent the final variable-to-check messages. If some codeword y agrees with the set $\{S_j^- : j \in \mathcal{J}\}$ of minimum cost configurations under m, then y must be the ML codeword.*

Proof: Consider the solution (y, w) to the primal, where $w_{j,S} = 1$ if S agrees with y, and $w_{j,S} = 0$ otherwise. This solution is clearly feasible, since y is a codeword. Its cost is equal to $\sum_i \gamma_i y_i$.

Now consider the following solution to the dual. For all edges (i, j), set m_{ij} equal to the current message from i to j. For all $j \in \mathcal{J}$, set $c_j = \min_{S \in E_j}(\sum_{i \in S} m_{ij})$. This dual solution is feasible, by condition (8.8) and the definition of c_j. We claim that this primal and dual solution obey complementary slackness. Note that for all $w_{j,S}$ we have $(w_{j,S} > 0) \implies (c_j = \sum_{i \in S} m_{ij})$. Additionally, for all y_i we have $\sum_{j \in N(i)} m_{ij} = \gamma_i$. Therefore (y, w) and (m, c) obey complementary slackness, and are both optimal. Thus, by Lemma 5.2, the codeword y is the minimum cost (ML) codeword. ■

Theorem 8.3 gives a generic stopping condition for a message-passing algorithm to give an ML certificate: if the messages are cost-balanced, and all the minimum configurations agree on a codeword, then output that codeword.

8.2.4 Lagrangian Dual

In this section we give an alternative form for the LP dual based on a partial Lagrangian relaxation. We use this form to derive a specific message-passing algorithm based on the subgradient algorithm [BT97] that is guaranteed to converge to the LP optimum. Finally, we use this dual form to provide a more general way for *any* message-passing algorithm to give an ML certificate.

We define a partial Lagrangian dual form of our LP by "dualizing" only the consistency constraints (Chapter 5, equation (5.4)) on each edge. Specifically, for a particular edge (i, j) in the graph, define the "consistency" of a setting of (f, w) as

$$A_{ij}(f, w) = \left(\sum_{S \in E_j, S \ni i} w_{j,S} \right) - f_i. \tag{8.9}$$

We define Lagrange multipliers m_{ij} for each edge, and obtain the following objective function:

$$\mathcal{L}(f, w; m) = \sum_i \gamma_i f_i + \sum_{i,j} m_{ij} A_{ij}(f, w) \tag{8.10}$$

Recall that $\mathcal{Q}$ is the LP polytope defined in Section 5.1. Note that $A_{ij}(f,w) = 0$ for all $(f,w) \in \mathcal{Q}$. It follows that $\mathcal{L}(f,w;m) = \sum_i \gamma_i f_i$ for all $(f,w) \in \mathcal{Q}$. Let $\mathcal{L}(\mathcal{Q}) \supseteq \mathcal{Q}$ be the polytope that remains after removing the constraints (5.4). Specifically, $\mathcal{L}(\mathcal{Q})$ is the set of points (f,w) such that:

$$\forall j \in \mathcal{J}, \qquad \sum_{S \in E_j} w_{j,S} = 1, \tag{8.11}$$

$$\forall i \in \mathcal{I}, \qquad 0 \le f_i \le 1, \text{ and} \tag{8.12}$$

$$\forall j \in \mathcal{J}, S \in E_j, \qquad w_{j,S} \ge 0. \tag{8.13}$$

The polytope $\mathcal{L}(\mathcal{Q})$ is quite relaxed, requiring only that a value between 0 and 1 is chosen for each bit, and a convex combination of configurations in E_j is chosen for each $j \in \mathcal{J}$. Note that in the polytope $\mathcal{L}(\mathcal{Q})$, the $\{f\}$ and $\{w\}$ variables are completely independent.

Suppose we fix m. Define

$$\mathcal{L}^*(m) = \min_{(f,w) \in \mathcal{L}(\mathcal{Q})} \mathcal{L}(f,w;m).$$

The minimization problem above is much simpler than optimizing over $\mathcal{Q}$, since f and w are independent. To see this, we use equations (8.10) and (8.9) to rewrite $\mathcal{L}(f,w;m)$ as

$$\mathcal{L}(f,w;m) = \sum_i f_i \left(\gamma_i - \sum_{j \in N(i)} m_{ij}\right) + \sum_j \sum_{S \in E_j} w_{j,S} \sum_{i \in S} m_{ij}. \tag{8.14}$$

Now we can find (f,w) such that $\mathcal{L}(f,w;m) = \mathcal{L}^*(m)$ as follows:

- For all $j \in \mathcal{J}$, let
$$S_j^- = \arg\min_{S \in E_j} \sum_{i \in S} m_{ij}. \tag{8.15}$$
Now set $w_{j,S_j^-} = 1$. Set $w_{j,S} = 0$ for all other $S \in E_j, S \neq S_j^-$. This setting of w minimizes the second term in equation (8.14).

- For all $i \in \mathcal{I}$, let
$$f_i = \begin{cases} 1 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) < 0 \\ 0 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) > 0 \end{cases} \tag{8.16}$$
If $\gamma_i = \sum_{j \in N(i)} m_{ij}$ then f_i may be set arbitrarily. This setting of f minimizes the first term in equation (8.14).

Since $\mathcal{L}(f,w;m)$ is independent of m for all $(f,w) \in \mathcal{Q}$, raising $\mathcal{L}^*(m)$ "exposes" the minimum-cost point in $\mathcal{Q}$. So, we would like to find the setting of m that maximizes $\mathcal{L}^*(m)$. In fact, using strong duality, we have the following:

$$\min_{(f,w) \in \mathcal{Q}} \sum_i \gamma_i f_i = \max_m \mathcal{L}^*(m)$$

8.2.5 The Subgradient Algorithm

To solve the problem $\max_m \mathcal{L}^*(m)$, we can use the subgradient algorithm [BT97], as we did for RA(2) codes. We generate a sequence $(m^0, m^1, m^2, \dots)$ of Lagrange multipliers, which we can think of as messages. For a particular message vector m^k, we compute m^{k+1} by adding a subgradient of the function $\mathcal{L}^*(\cdot)$ at the point m^k. The subgradient d is a vector with the same dimension as m; so, it consists of a value for each edge in the graph.

At each iteration, finding a subgradient is simple. We have already given a procedure (equations (8.15) and (8.16)) for finding (f, w) such that $\mathcal{L}^*(m) = \mathcal{L}(f, w; m)$. Using this setting of (f, w), we simply set $d_{ij} = A_{ij}(f, w)$ for all edges (i, j). It can be shown [BT97] that this will always be a subgradient. Note that since (f, w) is integral, we have $d \in \{-1, 0, +1\}^n$.

After finding the subgradient d, we set $m^{k+1} = m^k + \alpha^k d$, where α^k is the step-size parameter. As in RA(2) codes, this procedure is guaranteed to converge as long as the conditions in equation 8.5 hold.

To summarize, our message-passing algorithm proceeds as follows. Initially, all $m^0_{ij} = 0$. In the algorithm to follow, we also have messages m_{ji} from checks to variables in order to fit better into the paradigm; these messages will indicate membership in the minimum-cost configuration for a given check node. For an iteration k, the algorithm performs the following steps:

1. Update the check-to-variable messages. The messages from a check j are indicators of membership in the minimum-cost configuration S_j^-. If a variable node is in the min-cost configuration, then the check "believes" that it should be set to one. Formally, for all $j \in \mathcal{J}$:
   - Let $S_j^- = \arg\min_{S \in E_j} \sum_{i \in S} m_{ij}^{k-1}$. (Break ties arbitrarily).
   - For all $i \in N(j)$, if $i \in S_j^-$, set $m^k_{ji} = 1$; otherwise set $m^k_{ji} = 0$.

2. Update the variable-to-check messages. The messages from a variable i indicate whether the current setting of the bit y_i agrees with the belief of each check in the neighborhood of i. If there is agreement, then the message is unchanged from the previous iteration; otherwise, the message is either increased or decreased, in a way that will bias the next iteration toward agreement. Formally, for all $i \in \mathcal{I}$, we set
$$y_i = \begin{cases} 1 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) < 0 \\ 0 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) \geq 0 \end{cases}$$
   Then, for all $j \in N(i)$:
   - If $y_i = m^k_{ji}$, set $m^k_{ij} = m^{k-1}_{ij}$.
   - If $y_i = 0$ and $m^k_{ji} = 1$, set $m^k_{ij} = m^{k-1}_{ij} + \alpha^k$.
   - If $y_i = 1$ and $m^k_{ji} = 0$, set $m^k_{ij} = m^{k-1}_{ij} - \alpha^k$.

   The step-size α^k is any function of the form indicated in equation (8.5).

3. If none of the variable-to-check messages changed (i.e., if $y_i = m^k_{ji}$ for all edges (i, j)), then stop. Output y, which is the ML codeword.

This algorithm is guaranteed to converge to the LP optimum. If there is a unique integral optimum to the LP, then the algorithm converges to the ML codeword. However, if the LP optimum is not integral, then the algorithm will reach a state where the minimum configurations S^-_j oscillate between different configurations in E_j, and the cost difference between them goes to zero. A reasonable strategy to handle this is to terminate after a fixed number of iterations.

8.2.6 Another Message-Passing ML Certificate

We can also use the Lagrangian dual to derive an ML certificate stopping condition that applies to *any* message-passing algorithm. This generalizes Theorem 8.3.

Theorem 8.4 *Suppose a message-passing algorithm terminates with a codeword y and messages m_{ij} for each edge in G. Then y is the ML codeword, as long as*

- for all $j \in \mathcal{J}, S \in E_j$, $$\sum_{i \in N(j): y_i = 1} m_{ij} \leq \sum_{i \in S} m_{ij}, \quad \text{and} \tag{8.17}$$
- for all $i \in \mathcal{I}$, $$y_i = \begin{cases} 1 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) < 0 \\ 0 & \text{if } (\gamma_i - \sum_{j \in N(i)} m_{ij}) > 0 \end{cases}. \tag{8.18}$$

Proof: Suppose $(y,\ m)$ fits the assumptions in the statement of the theorem. We construct a point $(f, w) \in \mathcal{Q}$ as follows. Let $f = y$. For each $j \in \mathcal{J}$, let S^*_j be the unique configuration in E_j such that for all $i \in N(j)$, $y = 1$ if and only if $i \in S^*_j$. Now let $w_{j,S^*_j} = 1$ for all $j \in \mathcal{J}$, and set all other $w_{j,S} = 0$.

We claim that the point (f, w) is a minimum point in $\mathcal{L}(\mathcal{Q})$ under the cost function $\mathcal{L}(f, w; m)$; i.e., $\mathcal{L}(f, w; m) = \mathcal{L}^*(m)$. To see this, first note that equation (8.17) implies that $S^*_j = S^-_j$ for all $j \in \mathcal{J}$, where S^-_j is as defined in equation (8.15). Additionally, equation (8.18) implies that $f_i(\gamma_i - \sum_{j \in N(i)} m_{ij}) \leq f'_i(\gamma_i - \sum_{j \in N(i)} m_{ij})$ for all $f' \in [0, 1]^n$. Using equation (8.14), we may conclude that $\mathcal{L}(f, w; m) = \mathcal{L}^*(m)$.

Since $\mathcal{Q} \subseteq \mathcal{L}(\mathcal{Q})$, we have $\mathcal{L}^*(m) \leq \mathcal{L}(f', w'; m)$, for all $(f', w') \in \mathcal{Q}$, and so $\mathcal{L}(f, w; m) \leq \mathcal{L}(f', w'; m)$ for all $(f', w') \in \mathcal{Q}$.

However, for all $(f', w') \in \mathcal{Q}$, we have $\mathcal{L}(f', w'; m) = \sum_i \gamma_i f'_i$. Therefore, since $(f, w) \in \mathcal{Q}$, we have $\sum_i \gamma_i f_i \leq \sum_i \gamma_i f'_i$ for all $(f', w') \in \mathcal{Q}$. We may conclude that (f, w) is an optimal point in $\mathcal{Q}$. Since (f, w) is integral, $f = y$ is the ML codeword. ■

Note that Theorem 8.4 does not require that the message-passing algorithm be *cost-balanced*. This generalization occurs because the partial Lagrangian dual form of the LP puts the messages into the objective function, whereas the LP dual in Section 8.2 has the messages as dual variables with hard constraints.

Chapter 9

Conclusions and Future Work

The work in this thesis represents the first consideration of the application of linear programming relaxation to the problem of decoding an error-correcting code. We have been successful in applying the technique to the modern code families of turbo codes and LDPC codes, and have proved a number of results on error-correcting performance. However, there is much to understand even within these families, let alone in other code families we have yet to explore. In this final chapter we survey some of the major open questions in LP decoding of turbo codes and LPDC codes, and suggest a number of general ideas for future research in this area.

9.1 Turbo Codes

Improving the Running Time A drawback of the LP approach to turbo decoding is the complexity of solving a linear program. Even though the simplex algorithm runs quite fast in practice, most applications of error-correcting codes require a more efficient decoding algorithm. There are two possible solutions to this problem, and both have some important unanswered questions.

The first option is to try to solve the turbo code LPs combinatorially. For the case of RA(2) codes, the resulting agreeable flow problem can be reduced to an instance of normal min-cost flow, and thus yields a more efficient combinatorial algorithm. It is an interesting open question as to whether combinatorial solutions exist for RA(R) codes (where $R \geq 3$), or other codes.

The agreeable flow problem has a more general formulation that could apply to areas outside of coding theory. We define the *min-cost agreeable flow problem* as follows:

Min-Cost Agreeable Flow: Given a directed network $G = (V, E)$, a source $s \in V$ and a sink $t \in V$ with a demand α, capacities $u : E \rightarrow \mathbb{R}^+$ and costs $c : E \rightarrow \mathbb{R}$ on the edges, and a sequence of edge sets $\mathcal{A} = \{\{A_1, \hat{A}_1\}, \{A_2, \hat{A}_2\}, \ldots, \{A_m, \hat{A}_m\}\}$, find a minimum-cost α-unit flow f from s to t, where $\forall(A_i, \hat{A}_i)$, the total flow going through arcs in A_i is equal to the total flow going through arcs in $\hat{A}_i$.

Any LP with a constraint matrix made up of $\{+1, -1\}$ can be expressed using only the agreeability constraints of the above formulation, so we would not expect to be able to solve *Min-Cost Agreeable Flow* combinatorially in its full generality. However, the specialized structure of RALP or TCLP may allow a combinatorial solution. In general, it is an interesting question to determine how the min-cost agreeable flow problem must be restricted in order to make it solvable combinatorially.

Improving the Error Bounds. The RA(2) that we were able to analyze completely is not the best code experimentally in the literature. In fact, this code is a *cycle code*, as discussed in Section 7.3.3. Since cycle codes are considered poor, it is important to understand the combinatorics behind more complicated codes such as a rate 1/3 RA code, and the classic turbo code (parallel concatenated convolutional code). In order to provide better bounds for these codes, we need to prove that negative-cost "promenade"-like subgraphs are unlikely. Theorem 6.2 suggested a design for an interleaver for the RA(2) code. It would be interesting to see if other design suggestions can be derived for more complex turbo codes.

9.2 Low-Density Parity-Check Codes

In Chapter 5 we described an LP-based decoding method for LDPC codes, and proved a number of results on its error-correcting performance. Central to this characterization is the notion of a *pseudocodeword*, which corresponds to a rescaled solution of the LP relaxation. Our definition of pseudocodeword unifies previous work on message-passing decoding (e.g., [FKMT01, FKKR01, Wib96, DPR$^+$02]). We also introduced the *fractional distance* of the relaxation for a code, a quantity which shares the worst-case error-correcting guarantees with the classical notion, but with an efficient algorithm to realize those guarantees.

There are a number of open questions and future directions suggested by this work. It is likely that the fractional distance bound in Chapter 5 can be substantially strengthened by consideration of graph-theoretic properties other than the girth (e.g., expansion), or by looking at random codes. A linear lower bound on the fractional distance would yield a decoding algorithm with exponentially small error rate. This is particularly important to show that LP decoding can compete with algorithms that are known to correct a constant fraction of error (e.g., expander-based codes [SS96, GI01]).

However, even if this is not the case, the performance of LP decoding may still be very good. For RA codes, we were able to prove a bound on the error rate of LP decoding stronger than that implied by the minimum distance. It would be interesting to see the same result in the more general setting of LDPC codes or linear codes. Perhaps an analysis similar to that of Di et al. [DPR$^+$02], which was performed on stopping sets in the BEC, could be applied to pseudocodewords in other channel models.

Our experiments on LDPC codes leave a number of open questions as well. In Figure 7-5, it seems that LP decoding performs better than sum-product at very low noise. While this data is insufficient to draw any general conclusions, it may be that LP decoding has different behavior than sum-product in the "error-floor" region (very low noise). This difference in behavior is due to the fact that the "pseudocodewords" for LP decoding are different than those of sum-product decoding. An analytical understanding of this difference would be preferable, but an experimental study would be interesting as well.

9.3 Message-Passing Algorithms

We have used LP duality to give message-passing decoders the ability to prove that their output is the ML codeword. In addition, new message-passing decoders based on the subgradient algorithm were derived. Unlike the standard belief propagation algorithm, this message-passing decoder converges; furthermore, it converges to a well-defined point: the LP optimum. Using results from the previous chapters, we obtained exact characterizations of decoding success for these new decoders, for any discrete memoryless channel, and any block length, even in the presence of cycles in the factor graph; this is something we do not have for the standard message-passing algorithms.

There are many interesting open questions regarding the relationship between message-passing algorithms and LP decoding. The subgradient algorithms we give in Chapter 8 converge to the LP optimum, but to get a bound on the running time, we need to run an interior point algorithm to solve the LP. It would be interesting to see if there was a primal-dual or other combinatorial algorithm with a provably efficient running time. Additionally, while we do provide a way for a message-passing algorithm to show an ML certificate, it is not clear how well-established algorithms like belief propagation should be modified in order to meet our conditions. An interesting question is how modifying belief propagation to be "cost-balanced" affects performance.

9.4 Higher-Order Relaxations

Perhaps the most attractive aspect to the method of LP decoding is in its potential ability to improve. With message-passing decoders, it is unclear how to improve the algorithm to make it perform better. However, with LP decoding, we can employ a number of different known techniques for tightening an LP relaxation, and instantly get a better decoder.

We discussed this idea somewhat in Section 5.4; however, there is still much to understand. We have "interpreted" the first-order Lovász-Schrijver relaxation as it applies to LDPC codes; however, the higher-order relaxations remain a mystery. It would be interesting to understand how quickly this hierarchy of decoders approaches ML decoding, even for a class of codes specifically constructed for this purpose. Furthermore, there are other generic methods for improving 0-1 LP relaxations (e.g., Sherali and Adams [SA90], Lasserre [Las01]; surveyed in [Lau01]), and it would be interesting to explore the impact of these methods on LP decoding. Semi-definite programming is an important aspect of many of these techniques; perhaps there is a natural decoding relaxation similar to the Goemans and Williamson's semi-definite relaxation for max-cut [GW95].

Perhaps we can also use the idea of LP tightening to produce an adaptive algorithm that approaches an ML decoder. Since our LP decoders have the ML certificate property, they can stop when they find an integral optimum. In fact, in decoding, when the noise in the channel is low, it is very common for the "first-order" LPs (the ones we give in this thesis) to be correct. In the rare case when the LP returns a fractional solution, instead of outputting "error," we could try to come up with a cut of the polytope that would eliminate the fractional vertex we found, then re-optimize. It would be interesting to prove something about the relationships between running time, noise level and decoding success in this setting.

9.5 More Codes, Alphabets and Channels

In this thesis we study binary linear codes in memoryless symmetric channels. There is certainly a way to generalize the notion of LP decoding for more general codes and channels, and this opens up a whole new set of questions.

In many applications, we operate over larger alphabets than binary (for example in the transmission of internet packets). We could model this in an LP by having a variable range over this larger space, or by using several $0-1$ variables as indicators of a symbol taking on a particular value. Alternatively, we could map the code to a binary code, and use an LP relaxation for the binary code. It would be interesting to see if anything is gained by by representing the larger alphabet explicitly.

In practice, channels are generally not memoryless due to physical effects in the communication channel (see [Pro95] for details). Even coming up with a proper linear cost function for an LP to use in these channels in an interesting question. The notions of pseudocodeword and fractional distance would also need to be reconsidered for this setting.

9.6 A Final Remark

The work in this thesis represents the exploration of a computer science theorist into the world of error-correcting codes. We have discovered that many standard techniques in theoretical computer science can help shed light on the algorithmic issues in coding theory. The author therefore encourages communication between these two fields, and hopes that this work serves as an example of the gains that can result.

Bibliography

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[BGT93] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo-codes. *Proc. IEEE International Conference on Communication (ICC), Geneva, Switzerland*, pages 1064–1070, May 1993.

[Big98] N. Biggs. Constructions for cubic graphs with large girth. *Electronic Journal of Combinatorics*, 5(A1), 1998.

[Bla83] R. E. Blahut. *Theory and Practice of Error-Control Codes*. Addison-Wesley, 1983.

[BMMS01] L. Bazzi, M. Mahdian, S. Mitter, and D. Spielman. The minimum distance of turbo-like codes. *manuscript*, 2001.

[BT97] D. Bertsimas and J. Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, Belmont, MA, 1997.

[CFRU01] S.-Y. Chung, G. D. Forney, T. Richardson, and R. Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5(2):58–60, February 2001.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. M.I.T. Press, Cambridge, Massachusetts, U.S.A., 2001.

[DJM98] D. Divsalar, H. Jin, and R. McEliece. Coding theorems for 'turbo-like' codes. *Proc. 36th Annual Allerton Conference on Communication, Control, and Computing*, pages 201–210, September 1998.

[DPR$^+$02] C. Di, D. Proietti, T. Richardson, E. Telatar, and R. Urbanke. Finite length analysis of low-density parity check codes. *IEEE Transactions on Information Theory*, 48(6), 2002.

[EA75] S. F. Edwards and P. W. Anderson. *J. Phys. F.*, 5(965), 1975.

[ES63] P. Erdös and H. Sachs. Regulare graphen gegebene taillenweite mit minimaler knotenzahl. *Wiss. Z. Univ. Hall Martin Luther Univ. Halle–Wittenberg Math.– Natur.Reine*, 12:251–257, 1963.

[Fel68] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley & Sons, Inc., third edition, 1968.

[FK02a] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. *Proc. 43rd annual IEEE Symposium on Foundations of Computer Science (FOCS)*, November 2002.

[FK02b] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. Manuscript, submitted to: *Journal of Computer and System Sciences*, January 2002.

[FKKR01] G. D. Forney, R. Koetter, F. R. Kschischang, and A. Reznik. On the effective weights of pseudocodewords for codes defined on graphs with cycles. In *Codes, systems and graphical models*, pages 101–112. Springer, 2001.

[FKMT01] G. D. Forney, F. R. Kschischang, B. Marcus, and S. Tuncel. Iterative decoding of tail-biting trellises and connections with symbolic dynamics. In *Codes, systems and graphical models*, pages 239–241. Springer, 2001.

[FKW02] J. Feldman, D. R. Karger, and M. J. Wainwright. Linear programming-based decoding of turbo-like codes and its relation to iterative approaches. In *Proc. 40th Annual Allerton Conference on Communication, Control, and Computing*, October 2002.

[FKW03] J. Feldman, D. R. Karger, and M. J. Wainwright. Giving message-passing decoders a maximum-likelihood certificate. Manuscript, submitted to: *15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, April 2003.

[For73] G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, 1973.

[For74] G. D. Forney. Convolutional codes II: Maximum likelihood decoding. *Information Control*, 25:222–266, 1974.

[For01] G. D. Forney. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47, February 2001.

[For03] G. D. Forney. Personal communication, 2003.

[FWK03a] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode linear codes. *37th annual Conference on Information Sciences and Systems (CISS '03)*, 2003.

[FWK03b] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode linear codes. Manuscript, submitted to: *IEEE Transactions on Information Theory*, May 2003.

[Gal62] R. Gallager. Low-density parity-check codes. *IRE Trans. Inform. Theory*, IT-8:21–28, Jan. 1962.

[GG84] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions in Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984.

[GI01] V. Guruswami and P. Indyk. Expander-based constructions of efficiently decodable codes. *42nd Symposium on Foundations of Computer Science (FOCS)*, 2001.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[GLS81] M. Grotschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[Gur01] V. Guruswami. *List Decoding of Error-Correcting Codes*. PhD thesis, Massachusetts Institute of Technology, 2001.

[GW95] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42:1115–1145, 1995.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Tech. Journal*, XXIX(2):147–160, 1950.

[Hoc95] D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing, 1995.

[ILO01] ILOG, Inc. *User's Manual for ILOG CPLEX v. 7.1*, 2001.

[Jer75] R. G. Jeroslow. On defining sets of vertices of the hypercube by linear inequalities. *Discrete Mathematics*, 11:119–124, 1975.

[JZ98] R. Johannesson and K. Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Press, 1998.

[Las01] L. B. Lasserre. An explicit exact SDP relaxation for nonlinear $0-1$ programs. *K. Aardal and A.M.H. Gerards, eds.,* Lecture Notes in Computer Science, 2081:293–303, 2001.

[Lau01] M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver and Lasserre relaxations for $0-1$ programming. Technical Report PNA–R0108, Centrum voor Wiskunde en Informatica, CWI, Amsterdam, The Netherlands, 2001.

[LMSS98] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Improved low-density parity-check codes using irregular graphs and belief propagation. *Proc. 1998 IEEE International Symposium on Information Theory*, page 117, 1998.

[LS91] L. Lovász and A. Schrijver. Cones of matrices and set-functions and $0-1$ optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[Mac99] D. MacKay. Good error correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

[Mac03] D. MacKay. *Information Theory, Inference and Learning Algorithms.* Cambridge Press, 2003. *Scheduled for publication in 2007*. Avaiable electronically: `http://www.inference.phy.cam.ac.uk/mackay`.

[MMC98] R. McEliece, D. MacKay, and J. Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[MS81] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North-Holland, 1981.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[PH98] V. S. Pless and W. Huffman, editors. *Handbook of Coding Theory*. Elsevier Science, 1998.

[Pro95] J. G. Proakis. *Digital Communications*. McGraw-Hill, 1995.

[RU01] T. Richardson and R. Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[RV00] J. Rosenthal and P. O. Vontobel. Constructions of LDPC codes using Ramanujan graphs and ideas from Margulis. In *Proc. of the 38-th Annual Allerton Conference on Communication, Control, and Computing*, pages 248–257, 2000.

[SA90] H. D. Sherali and W. P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Optimization*, 3:411–430, 1990.

[Sau67] N. Sauer. Extremaleigenschaften regularer graphen gegebener taillenweite, i and ii. *Sitzungsberichte Osterreich. Acad. Wiss. Math. Natur. Kl.*, S-BII(176):27–43, 1967.

[Sch87] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley, 1987.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[SS96] M. Sipser and D. Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[ST01] D. Spielman and S. Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 296–305, 2001.

[Tan81] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

[Vit67] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, Apr. 1967.

[vL99] J. H. van Lint. *Introduction to Coding Theory*. Springer-Verlag, 1999.

[VY00] B. Vucetic and J. Yuan. *Turbo Codes*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[Wai02] M. J. Wainwright. *Stochastic processes on graphs with cycles: geometric and variational approaches*. PhD thesis, Massachusetts Institute of Technology, 2002.

[Wib96] N. Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, Linkoping University, Sweden, 1996.

[Wic95] S. Wicker. *Error Control Systems for Digital Communication and Storage*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[WJW02] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. MAP estimation via agreement on (hyper)trees: Message-passing and linear programming approaches. In *Proc. 40th Annual Allerton Conference on Communication, Control, and Computing*, October 2002.

[Yan91] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466, 1991.

[Yed03] J. S. Yedidia. Personal communication, 2003.

[YFW02] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. Technical Report TR2001-22, Mitsubishi Electric Research Labs, January 2002.