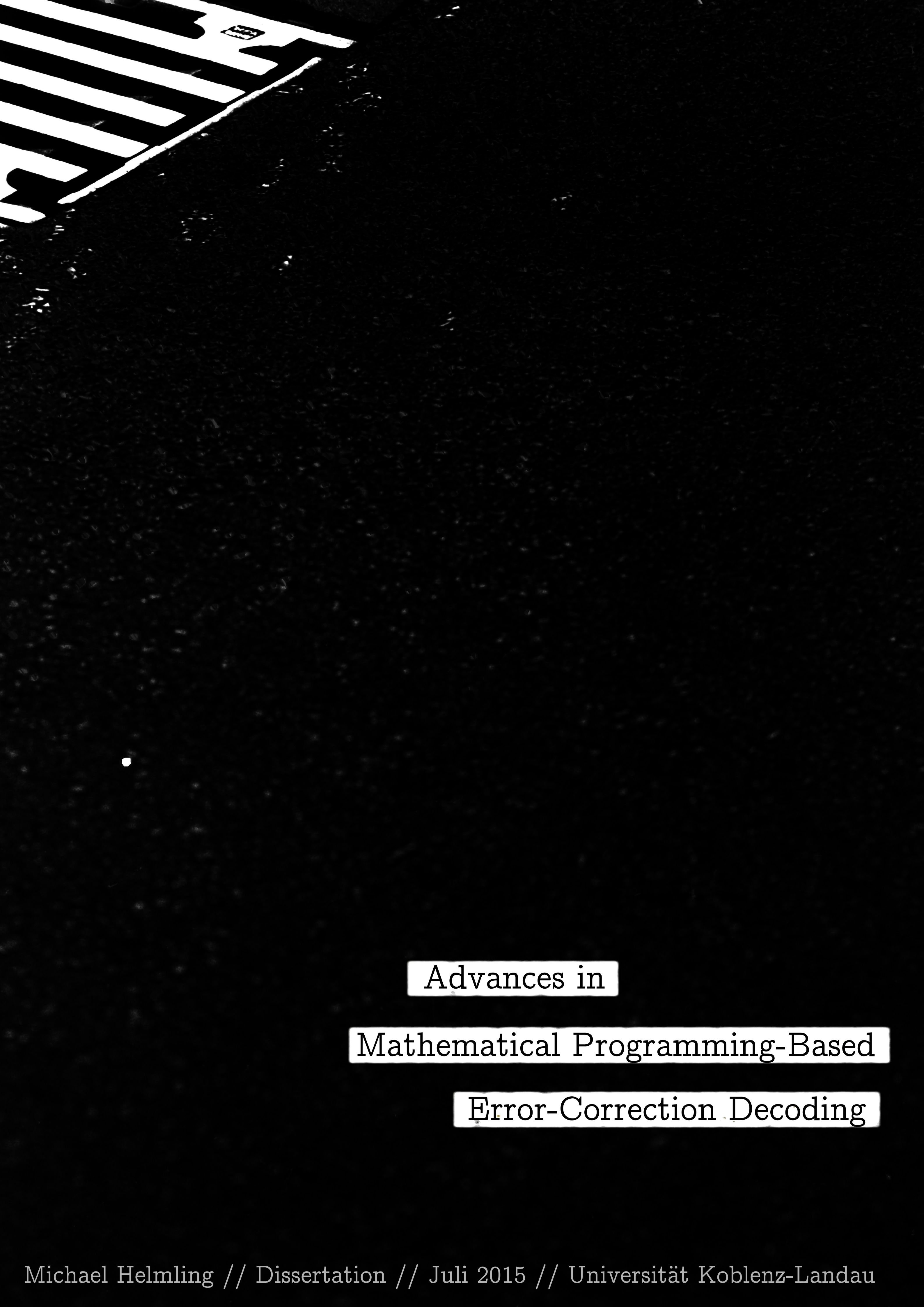


Advances in Mathematical Programming-Based Error-Correction Decoding

Michael Helmling // Dissertation // Juli 2015 // Universität Koblenz-Landau

Cover art: Viola Simon, *Binary Channel*, b/w photograph, 2015. Included with kind permission of the artist.

Advances in Mathematical Programming-Based Error-Correction Decoding

von

Michael Helmling
geb. am 19. Februar 1986 in Kaiserslautern

Angenommene Dissertation zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften

Fachbereich 3: Mathematik/Naturwissenschaften
Universität Koblenz-Landau

Gutachter:
Prof. Dr. Stefan Ruzika
Prof. Dr. Øyvind Ytrehus
Prof. Dr. Pascal O. Vontobel

Datum der mündlichen Prüfung: 17. Juli 2015

Abstract

The formulation of the decoding problem for linear block codes as an integer program (IP) with a rather tight linear programming (LP) relaxation has made a central part of *channel coding* accessible for the theory and methods of *mathematical optimization*, especially integer programming, polyhedral combinatorics and also algorithmic graph theory, since the important class of *turbo codes* exhibits an inherent graphical structure.

We present several novel models, algorithms and theoretical results for error-correction decoding based on mathematical optimization. Our contribution includes a partly combinatorial LP decoder for turbo codes, a fast branch-and-cut algorithm for maximum-likelihood (ML) decoding of arbitrary binary linear codes, a theoretical analysis of the LP decoder's performance for 3-dimensional turbo codes, compact IP models for various heuristic algorithms as well as ML decoding in combination with higher-order modulation, and, finally, first steps towards an implementation of the LP decoder in specialized hardware.

The scientific contributions are presented in the form of seven revised reprints of papers that appeared in peer-reviewed international journals or conference proceedings. They are accompanied by an extensive introductory part that reviews the basics of mathematical optimization, coding theory, and the previous results on LP decoding that we rely on afterwards.

Acknowledgements

Every work of an individual is the work of many. This is especially true for a PhD thesis, which is why I devote my greatest thanks to all of my “many”, who supported me throughout the eventful period of my life from which this thesis has emerged, non-exhaustively (sorry for everyone I forgot) including my family, friends, colleagues and co-authors (this listing is by no means without overlap!). Very special thanks go to my supervisor Stefan (Ruzika), who in my view rather belongs to the *friend* or *colleague* than *boss* category, to Eirik, whom to work with is always a pleasure and who has been extremely supportive throughout my entire time as a PhD student, and to Stefan (Scholl), my colleauge from microelectronic system design who has always been a patient contact for questions related to communications. I am also much obliged to my “predecessor” Akın, who gently introduced me into the subject during the period in which I was allowed to be his research assistant, and the remaining co-authors Frank, Alex and Florian.

The optimization group of the University of Kaiserslautern with which I was during the first half of my PhD deserves special thanks for being such a great environment, not only for work but also for social activities, as well as the ever-growing group at the University of Koblenz.

This work would not exist without financial support by different institutions. I would like to gratefully thank the *Center for Mathematical and Computational Modelling* $(CM)^2$, the *German Research Council (DFG; grant RU-1524/2-1)* and the *German Academic Exchange Service (DAAD)* for funding my work and enabling me to be part of an international research community by supporting numerous visits of conferences and foreign universities.

You would not be able to read this book if it hadn’t been written down. While the typesetting system called *LaTeX* that I availed myself of for this task *does* cost nerves from time to time, putting a mathematical text (and figures!) down on paper *without* it would be true horror without question. I hence express my gratitude to the huge community that makes all this software available for free. Regarding the process of reading this book, a special thank is expedient to everyone who helped proofreading the text, including the anonymous referees of the papers that constitute its contributions.

Last but not least, let me especially thank *you*, dear reader, for showing interest in the results of a sometimes escapist-seeming work.

Contents

Abstract iii

Acknowledgements v

1 Introduction 1

I Foundations 3

2 Notation and Preliminaries 7

3 Optimization Background 9
- 3.1 Polyhedral Theory . . . 10
- 3.2 Linear Programming . . . 14
- 3.3 Integer Programming . . . 18
- 3.4 Combinatorial Optimization . . . 20

4 Coding Theory Background 23
- 4.1 System Setup and the Noisy-Channel Coding Theorem . . . 23
- 4.2 MAP and ML Decoding . . . 27
- 4.3 The AWGN Channel . . . 27
- 4.4 Binary Linear Block Codes . . . 28
- 4.5 Convolutional and Turbo Codes . . . 30
- 4.6 Non-Binary Codes and Higher-Order Modulation . . . 33

5 Optimization Into Coding: The Connection 35
- 5.1 ML Decoding as Integer Program . . . 35
- 5.2 LP Decoding . . . 37
- 5.3 Adaptive LP Decoding . . . 38
- 5.4 Analysis of LP Decoding . . . 39
- 5.5 LP Decoding of Turbo Codes . . . 44

II Contributions 47

6 Paper I – Mathematical Programming Decoding of Binary Linear Codes: Theory and Algorithms 49
- 6.1 Introduction . . . 52

6.2 Basics and Notation 52
6.3 Complexity and Polyhedral Properties 56
6.4 Basics of LPD 60
6.5 LPD Formulations for Various Code Classes 62
6.6 Efficient LP Solvers for BLPD 69
6.7 Improving the Error-Correcting Performance of BLPD 75
6.8 Conclusion 81
References 82

7 Paper II – ML vs. BP Decoding of Binary and Non-Binary LDPC Codes 89
7.1 Introduction 92
7.2 Binary and Non-Binary LDPC Codes 93
7.3 IP-based ML Decoding 96
7.4 Results 98
7.5 Conclusion 101
References 102

8 Paper III – Integer Programming as a Tool for Analysis of Channel Codes 105
8.1 Introduction 108
8.2 Preliminaries 109
8.3 Fundamentals of Solving IP Problems 110
8.4 IP Formulations for Linear Block Codes 111
8.5 Specialized IP Formulation for Turbo Codes 114
8.6 Simulation Results 115
8.7 Conclusion 120
References 120

9 Paper IV – Towards an Exact Combinatorial Algorithm for LP Decoding of Turbo Codes 123
9.1 Introduction 126
9.2 Background and Notation 127
9.3 The LP Relaxation and Conventional Solution Methods 131
9.4 An Equivalent Problem in Constraints Space 132
9.5 The Complete Algorithm 141
9.6 Numerical Results 141
9.7 Improving Error-Correcting Performance 145
9.8 Conclusion and Outlook 146
References 147

10 Paper V – Towards Hardware Implementation of the Simplex Algorithm for LP Decoding 149
10.1 Introduction 152
10.2 The Simplex Algorithm 153
10.3 Designing Hardware 155

10.4 Simplex Performance Study 157
10.5 Conclusion 162
10.6 Outlook and Future Research 162
References 163

11 Paper VI – Efficient Maximum-Likelihood Decoding of Linear Block Codes on Binary Memoryless Channels 165
11.1 Introduction 168
11.2 Notation and Background 169
11.3 Basic Branch-and-Bound Algorithm 170
11.4 Improvements 172
11.5 Minimum Distance Computation 175
11.6 Numerical Results 176
References 178

12 Paper VII – Minimum Pseudoweight Analysis of 3-Dimensional Turbo Codes 181
12.1 Introduction 184
12.2 Coding Scheme 185
12.3 LP Decoding of 3D-TCs 186
12.4 Finite Graph Covers 188
12.5 Ensemble-Average Pseudoweight Enumerators 191
12.6 Searching for the Minimum Pseudoweight 195
12.7 Numerical Results 197
12.8 Conclusion 201
12.A Proof of Proposition 12.1 202
12.B Proof of Proposition 12.3 203
12.C Proof of Lemma 12.5 206
12.D Proof of Lemma 12.6 207
References 208

III Closing 211

13 Conclusions and Future Work 213

Bibliography for Parts I and III 215

Curriculum Vitae 219

Chapter 1

Introduction

The use of error-correcting codes in digital communication systems for the purpose of forward error correction is one of the essential pillars of modern information technology, as most applications of microelectronic devices rely on the assumption that data can be transmitted practically error-free.

Coding is the mathematical answer to the problem of unavoidable *noise* in data transmission, be it due to environmental influences on the carrier medium (e.g. background noise, physical obstacles, …), technical limitations of the transmitter and receiver electronics, or the like. The key idea is to include *redundancy* in the messages before they are sent in order to make them robust against the noise introduced during transmission. In mathematical terms, this is usually accomplished by a *linear map*, called the *code*, operating on fixed-sized words of the input. The receiver contains a complementary *decoder* that estimates the sent data, based on a given input signal it reads from the channel. If code and decoder are well designed, the end-to-end probability of error can in theory be made arbitrarily small, as long as the coding rate is below the capacity of the channel.

For a given code, an optimal decoder would always output the *maximum a-posteriori probability (MAP) codeword*, that is, the codeword that was sent with highest probability, given the observed noisy output of the transmission channel and a probabilistic model of that channel. Under certain assumptions (which are always met in the context of this thesis), the MAP codeword is equivalent to the *maximum-likelihood (ML) codeword*, which is the one maximizing the corresponding likelihood function, i.e., the probability of the received noisy message given that a specific codeword was sent. One notorious problem when it comes to practical implementations of communication systems lies in the complexity of the ML decoding problem, which appears to be increasingly hard, the stronger the code itself becomes. While specific classes of codes along with corresponding decoders that achieve a remarkable error-correction performance have been developed over the last two decades—most prominently, low-density parity-check (LDPC) and turbo codes [1, 2][1]–their decoding algorithms based on iterative message-passing are of heuristic nature, which makes them both suboptimal (they are not ML decoders) and hard to analyze theoretically.

In contrast, linear programming decoding algorithms, which are based on viewing the decoding problem as a *combinatorial optimization problem* formulated as *integer linear program (IP)*,

---

[1] The reference list for this part of the thesis can be found on page 215.

exhibit the potential to fill this gap, as they allow to exploit the rigorous, well-developed theory of mathematical optimization, notably polyhedral combinatorics, to tackle the decoding problem. The proposal of linear programs (LPs) for decoding both LDPC and turbo codes by Feldman *et al.* [3–5] has established a whole new research area of applied mathematics, subsumed under the term "LP decoding", in which several groups have subsequently produced improved formulations, extensions to more general classes of codes, theoretical analysis of the LP decoder's behavior and algorithms to efficiently solve the decoding LP—see Paper I (Chapter 6) for a recent overview of the advances.

This thesis presents my contributions to the research area of LP decoding in form of both theory and algorithms. It is organized in three parts. Part I thoroughly reviews the fundamentals of the three topics involved: mathematical optimization, coding theory, and the basics of their link, LP decoding. That part aims to be theoretically self-contained up to undergraduate mathematics; as the presentation is nevertheless rather compact, suggestions to several textbooks that cover the matter more elaborately are made in each chapter.

Part II of this thesis, which starts on page 49, contains seven peer-reviewed research papers that make up my scientific contributions. Within the thesis, they are refered to as Paper I to Paper VII and can be found in Chapter 6 to Chapter 12, ordered by their respective date of appearance. As I have worked within different projects and together with several groups, the topics of the papers are, to a certain extent, rather diverse, while still being connected via the overall topic *LP decoding*.

Paper I constitutes a literature survey of the LP models and algorithms that have been proposed until its appearance. Paper II is concerned with the comparison of binary and non-binary LDPC codes under both optimal (ML) and suboptimal decoding. Instead of formulating the ML decoding problem by the LP-relaxation of an integer program, Paper III fathoms the power of the integer programming-approach to other coding-related problems, including several heuristic decoders whose performance can be quickly evaluated using an IP. Papers IV and VII are concerned with the special class of turbo codes. Since codewords of turbo codes are related to paths in a graph, they are of particular interest from a combinatorial optimizer's view. In Paper IV, a novel mathematical algorithm is presented that efficiently solves a certain class of optimization problems in which the turbo decoding problem falls. Paper VII on the other hand comprises a thorough study of LP decoding of 3-dimensional turbo codes, a class of codes that has been invented to overcome certain shortages of "traditional" turbo codes. The goal of Paper V is to provide a first step towards an implementation of the LP decoder in specialized hardware. To that end, the decoder's complexity is analyzed from the hardware implementation view, and a study of its performance under limited precision arithmetics is conducted. Finally, Paper VI presents a true ML decoder that is based on a highly efficient branch-and-cut algorithm which incorporates several decoding-specific methods for generating primal and dual bounds.

In Part III, the closing part of this document, conclusions are drawn and outlook for future research is given. It also contains the bibliography for Parts I and III (note that each paper in Part II has its own list of references) and the author's academic CV (in German).

Part I

Foundations

In this first part of the thesis we present the background that is necessary for the main content in Part II. Because our subject is the application of mathematical optimization to the field of coding theory, the presentation splits naturally into the areas *mathematical optimization* (Chapter 3), *coding theory* (Chapter 4) and, finally, a review of the previous contributions by which these two areas have found common ground, in Chapter 5. The following chapters are not solely meant as a preparation for the study of Part II of this thesis, but also as an introduction in its own right to a particularly beautiful field of mathematics which is located on the rather uncrowded boundary between information theory, discrete mathematics, and optimization.

While the scope of this text demands a certain compactness of presentation, and a thesis introduction can impossibly replace a textbook or lecture, we have nevertheless made it our aim that a reader who is familiar with mathematics in general but has no specific proficiency in either optimization or coding theory will be able to comprehend the most important ideas and concepts of the matter. Due to length restrictions, we unfortunatelly could not include detailed examples of most of the presented concepts and results. Instead, those are accompanied, whenever applicable, by sketchy figures that try to illustrate the underlying intuitions in a non-rigorous manner.

The assumptions we make as to the reader's mathematical background mainly consist of undergraduate linear algebra. To a smaller extent, familiarity with graph theory and algorithmic concepts will be helpful, and occasionally we will encounter probabilities. The notation and nomenclature for these background topics are fixed in Chapter 2.

A concise notation throughout the entire thesis is unfortunately rendered impossible by its cumulative nature: the papers in Part II, as they cover a relatively broad selection of topics and are moreover written by diverse groups of authors, use different notational conventions, at least in some aspects. For this introductory part, we have tried to extract the most common notation wherever possible, so that for the attentive reader it should be easy to quickly adapt to the specific notation that is introduced in each of the papers.

Chapter 2

Notation and Preliminaries

Notation

Within the next chapters, we use the symbols $\mathbb{N}$ and $\mathbb{Z}$ for the natural (starting with 1) and integral numbers, respectively, $\mathbb{F}_2$ for the binary field (which is sometimes called GF(2)), $\mathbb{Q}$ for the rational numbers and $\mathbb{R}$ for the reals. If $x \in \mathbb{R}$, then $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the largest integer $\leq x$ and the smallest integer $\geq x$, respectively.

For a set R and $n \in \mathbb{N}$, R^n denotes the n-dimensional vector space over R if R is a field, and the set of n-tuples of elements of R otherwise. In either case, the n-tuples are called *vectors* and operations like addition or comparison, whenever applicable, are understood element-wise: if $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$, then

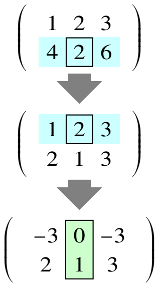
$$x + y = (x_1 + y_1, \dots, x_n + y_n)$$

and

$$x \leq y \text{ if and only if } x_i \leq y_i \text{ for } i = 1, \dots, n.$$

Matrices and Vectors

By $X^{m \times n}$ we denote the set of matrices with m rows and n columns and entries from the set X. For a matrix A, by $A_{i,j}$ we denote the element at the i-th row and j-th column of A, $A_{i,\bullet}$ and $A_{\bullet,j}$ are *the* i-th row and j-th column, respectively, and the transpose of A is the $n \times m$ matrix A^T with entries $A^T_{j,i} = A_{i,j}$. Throughout the text we regard vectors $x \in X^n$ as *column* vectors, i.e., identify them with $k \times 1$ matrices. We sometimes use *(ordered) index sets* instead of individual indexes: for instance, if $x \in X^n$ and $I = (i_1, \dots, i_k)$, where $\{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$, then $x_I = (x_{i_1}, \dots, x_{i_k})$. The same notation is used for row and column indexing, respectively, of matrices.

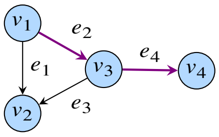


example Gaussian pivot operation

When $A \in R^{m \times n}$ is an $m \times n$ matrix over a ring R, an *elementary row operation* on A is one of the following operations: *(a)* multiply a row of A by a scalar: $A_{i,\bullet} \leftarrow sA_{i,\bullet}$, or *(b)* replace a row by the weighted sum of itself and another row: $A_{i,\bullet} \leftarrow sA_{i,\bullet} + tA_{j,\bullet}$. If R is a field, any finite sequence of elementary row operations (taking the scalars from R) leaves the range $\{Ax \colon x \in R^n\}$ of A unaltered. Such a sequence is called a *Gaussian pivot* on $A_{i,j}$ if it turns the j-th column of A into the i-th unit vector, and *Gaussian elimination* if it changes a submatrix of A into a triangular or diagonal matrix (the terms Gauss-Jordan pivot and, for diagonalization, Gauss-Jordan elimination are also used).

Graphs

A *graph* $G = (V, E)$ is defined by a finite set V, called the *nodes* or *vertices* of G, and a set $E \subseteq V \times V$ of *edges* or *arcs* of G. There are both *undirected* graphs, in which an edge $(u, v) \in E$ is identified with the unordered set $\{u, v\}$, and *directed graphs*, where (u, v) is perceived as an ordered pair. For a node $v \in V$ of a directed graph, we define $\delta^+(v) = \{(v, u) \colon (v, u) \in E\}$ and $\delta^-(v) = \{(u, v) \colon (u, v) \in E\}$ as the *outbound* and *inbound* edges, respectively, of v.



An acyclic directed graph with the path $P = (v_1, v_3, v_4)$ (using vertices) or $P = (e_2, e_4)$ (using edges) highlighted

A finite sequence $P = (v_1, \dots, v_k) \in V^k$ is called a v_1–v_k *path*, or simply *path*, of G, if $(v_i, v_{i+1}) \in E$ for $i = 1, \dots, k-1$. Each path can alternatively be defined by a sequence of edges, $P = (e_1, \dots, e_{k-1})$, where $e_i = (v_i, v_{i+1})$ for $i = 1, \dots, k-1$. The path P is called a *cycle* if $v_1 = v_k$. A graph is *acyclic* if no cycle exists.

For a graph $G = (V, E)$ and $V' \subseteq V$, the *subgraph induced by* V' is the graph $G' = (V', E')$ where $E' = \{(u, v) \in E \colon u \in V' \text{ and } v \in V'\}$ consists of all edges that connect two vertices of V'. Finally, a graph is called *bipartite* if there is a partition $V = V_1 \dot{\cup} V_2$ such that $E \subseteq \{V_1 \times V_2\} \cup \{V_2 \times V_1\}$, i.e., no edge connects two vertices of the same set V_i, $i = 1, 2$.

Complexity

The symbols P and NP are used to denote the complexity classes of problems that are *solvable* (P) and *verifiable* (NP) in polynomial time, and a problem is called *NP-hard* if it is "at least as hard" as *every* problem in NP, i.e., each problem in NP can be reduced to it in polynomial time. The *Landau notation* $f(n) = O(g(n))$ states that the asymptotic growth rate of $f(n) \colon \mathbb{N} \to \mathbb{R}$ is upper bounded by $g(n)$, i.e., there exist $M > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq Mg(n)$ for $n > n_0$. Note that a thorough understanding of complexity theory is not a prerequisite to reading this text.

Probability

Since we only come across basic probability calculations and they are not central to our text, we use simplified notation. Namely, we frequently denote both a random variable and its outcomes by the same symbol, say x, and use $P(x)$ for the probability mass or density function of x, whatever applies—the exact meaning of the symbols will always become clear from the context. If y is another random variable, $P(x \mid y_0)$ is the conditional probability function of x given the event $\{y \in y_0\}$. Similarly, $P(x_0 \mid y)$ is called the likelihood function of y, given $\{x \in x_0\}$. *Bayes' theorem* states that $P(x_0 \mid y_0) = \frac{P(y_0 | x_0) P(x_0)}{P(y_0)}$ for any two events x_0 of x and y_0 of y, provided $P(y_0) \neq 0$.

Chapter 3

Optimization Background

Mathematical optimization is a discipline of mathematics that is concerned with the solution of problems arising from mathematical models that typically describe real-world problems, e.g. in the areas of transportation, production planning, or organization processes. These models are generally of the form

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to (s.t.)} \quad & x \in X, \end{aligned}$$

where $X \subseteq \mathbb{R}^n$, for some $n \in \mathbb{N}$, is the *feasible set* and $f\colon X \to \mathbb{R}$ the *objective function* that evaluates a feasible point $x \in X$; $f(x)$ is called the *objective value* of x. An $x^* \in X$ minimizing the objective function is called an *optimal solution*, the corresponding value $z^* = f(x^*)$ the *optimal objective value*. If $X = \emptyset$, the problem is said to be *infeasible* and we define $z^* = \infty$ in that case. If on the other hand $f(x)$ is unbounded from below among X, we define $z^* = -\infty$.

The theory of optimization further subdivides into several areas that depend on the structure of both f and X. Within this thesis we will encounter three major problem classes: linear programs (LPs), integer linear programs (IPs), and combinatorial optimization problems, which are often modeled by means of an IP.

A common ground in the analysis of these three types of problems is the *polyhedral structure* of the feasible set. The part of polyhedral theory that is necessary for this text is reviewed in Section 3.1. For an LP, the feasible set is a polyhedron that is given explicitly by means of a defining set of linear inequalities, which makes these problems relatively easy to solve. LPs and the *simplex method*, the most important algorithm to solve them, are covered in Section 3.2.

In contrast to LPs, the feasible region of an IP is given only implicitly as the set of *integral* points within a polyhedron. Although the result exhibits again a polyhedral structure (as long as finding an optimal solution is the concern), IPs are much harder to solve than LPs; in fact, integer programming in general is an NP-hard optimization problem. Some theoretical foundations and techniques to nonetheless tackle such problems are collected in Section 3.3.

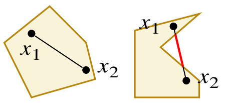
Proofs, examples and detailed explanations are widely omitted in this chapter. For a very exhaustive and detailed textbook on linear programming and the simplex method, see [6]. A complete and rigorous yet challenging reference for linear and integer programming is the book by Schrijver [7]. A well-written algorithm-centric introduction to linear, integer and also nonlinear optimization can be found in [8]. Finally, for integer and combinatorial optimization we refer to [9].

3.1 Polyhedral Theory

The mathematical objects that we talk about in this section live in the n-dimensional Euclidean space $\mathbb{R}^n$. Before we begin to discuss polyhedra and the theory around them, we briefly rush through some basic concepts which are necessary for that task.

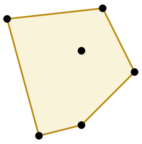
3.1.1 Convex Sets and Cones

Convexity is one of the most important concepts in mathematical optimization: intuitively speaking, a geometric object is convex if the straight line between any two points of the object lies completely inside of it.



a convex and a nonconvex set

3.1 Definition (convex and conic sets and hulls): A set $X \subseteq \mathbb{R}^n$ is called *convex* if for any $x_1, x_2 \in X$ and $\lambda \in [0,1]$ also $\lambda x_1 + (1-\lambda)x_2 \in X$.

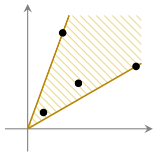


convex hull of six points

A *convex combination* of X is a sum of the form

$$\sum_{x \in X} \lambda_x x \colon \lambda_x \geq 0 \text{ for all } x \in X \text{ and } \sum_{x \in X} \lambda_x = 1, \tag{3.1}$$

where in the case of an infinite X we assume that almost all $\lambda_x = 0$ so that the above expression makes sense. The *convex hull* of X is the smallest convex set containing X or, alternatively, the set of all convex combinations of elements of X. ◁



conic hull of four points

An important class of convex sets is the one of convex *cones*.

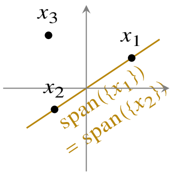
3.2 Definition (convex cones): $X \subseteq \mathbb{R}^n$ is called a (convex) *cone* if for any $x_1, x_2 \in X$ and $\lambda_1, \lambda_2 \geq 0$ also $\lambda_1 x_1 + \lambda_2 x_2 \in X$. A *conic combination* is defined like (3.1) but without the condition $\sum_{x \in X} \lambda_i = 1$. Analogously to the above, the *conic hull* $\operatorname{conic}(X)$ is the smallest cone containing X, or the set of all conic combinations of elements of X. ◁

Geometrically, the conic hull is the largest set that “looks the same” as $\operatorname{conv}(X)$, from the perspective of an observer that sits at the origin.

3.1.2 Affine Spaces

Recall the notion of *linear independence*: a set of points $X = \{x_1, \dots, x_k\} \subseteq \mathbb{R}^n$ is *linearly independent* if no element x of X is contained in the linear subspace that is spanned by the remainder $X \setminus \{x\}$ or, equivalently, if $\operatorname{span}(X') \neq \operatorname{span}(X)$ for all $X' \subsetneq X$. Here, the *span* or *linear hull* of a set $X \subseteq \mathbb{R}^n$ means the smallest linear subspace containing X:

$$\operatorname{span}(X) = \bigcap \{\mathscr{V} \colon \mathscr{V} \text{ is a subspace of } \mathbb{R}^n \text{ and } X \subseteq \mathscr{V}\}\,.$$


x_3
x_1
x_2
$\operatorname{span}(\{x_1\})$
$= \operatorname{span}(\{x_2\})$

x_1 and x_2 are linearly dependent, but both are pairwise linearly independent with x_3

The linear hull has an alternative algebraic characterization by means of linear combinations,

$$\operatorname{span}(X) = \left\{\sum_{i=1}^{k} \lambda_i x_i \colon \lambda_i \in \mathbb{R} \text{ for all } i \in \{1, \dots, k\}\right\}, \tag{3.2}$$

which gives rise to the well-known algebraic formulation of linear independence: X is linearly independent if and only if the system

$$\sum_{i=1}^{k} \lambda_i x_i = 0 \tag{3.3}$$

has the unique solution $\lambda_1 = \dots = \lambda_k = 0$. To see this, note that if there was some $j \in \{1, \dots, k\}$ such that $0 \neq x_j \in \operatorname{span}(X \setminus \{x_j\})$, then (3.2) gave rise to a non-zero solution of (3.3).

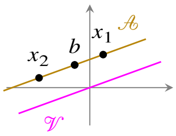
For the study of polyhedra, we need the concepts of *affine* hulls and independence, respectively, which is centered around the notion of an *affine subspace* in a very similar fashion as for the linear case above.

3.3 Definition (affine spaces, hulls, and independence): A set $\mathscr{A} \subseteq \mathbb{R}^n$ is an *affine subspace* of $\mathbb{R}^n$ if there exists a linear subspace $\mathscr{V} \subseteq \mathbb{R}^n$ and some $b \in \mathbb{R}^n$ such that

$$\mathscr{A} = b + \mathscr{V} = \{b + v \colon v \in \mathscr{V}\}\,. \tag{3.4}$$

The *affine hull* $\operatorname{aff}(X)$ of a set $X = \{x_1, \dots, x_k\} \subseteq \mathbb{R}^n$ is the smallest affine subspace containing X. The set X is called *affinely independent* if no $x \in X$ fulfills $x \in \operatorname{aff}(X \setminus \{x\})$ or, equivalently, if $\operatorname{aff}(X) \neq \operatorname{aff}(X')$ for all $X' \subsetneq X$.

Finally, the *dimension* $\dim(\mathscr{A})$ of $\mathscr{A}$ in (3.4) is defined to be the dimension of $\mathscr{V}$. $\triangleleft$



affine space $\mathscr{A} = b + \mathscr{V}$ of dimension 1; $x_1, x_2 \in \mathscr{A}$ are affinely independent

An affine subspace can thus be envisioned as a linear space that has been translated by a vector. This principle is reflected by the algebraic characterization of the affine hull of $X = \{x_1, \dots, x_k\} \subseteq \mathbb{R}^n$ by means of *affine combinations*: first, move to an arbitrary vector of X (without loss of generality, let this be x_1), then add any linear combination of the *directions* from x_1 to the other $k - 1$ elements of X:

$$\operatorname{aff}(X) = \left\{x_1 + \sum_{i=2}^{k} \lambda_i (x_i - x_1) \colon \lambda_i \in \mathbb{R} \text{ for all } i \in \{2, \dots, k\}\right\};$$

by defining $\lambda_1 = 1 - \sum_{i=2}^{k} \lambda_i$ one can easily derive the equivalent definition

$$\operatorname{aff}(X) = \left\{ \sum_{i=1}^{k} \lambda_i x_i \colon \lambda_i \in \mathbb{R} \text{ for all } i \in \{1, \dots, k\} \text{ and } \sum_{i=1}^{k} \lambda_i = 1 \right\}.$$

An algebraic definition of affine dependence can be derived in exactly the same way as for linear dependence: $x_j \in \operatorname{aff}\left(X \setminus \{x_j\}\right)$ if and only if one can write x_j as

$$x_j = x_l + \sum_{i \neq j,l} \lambda_i (x_i - x_l),$$

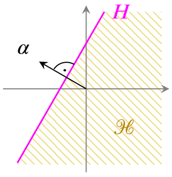
where $l \neq j$, which is in turn equivalent to the fact that

$$\sum_{i=1}^{k} \lambda_i \begin{pmatrix} x_i \\ 1 \end{pmatrix} = 0 \tag{3.5}$$

has a solution with $\lambda_j \neq 0$ (to see this, let $\lambda_l = 1 - \sum_{i \neq l,j} \lambda_i$ and $\lambda_j = -1$). Hence, X is affinely independent if and only if (3.5) has the unique solution $\lambda_1 = \dots = \lambda_k = 0$.

3.1.3 Polyhedra

A *polyhedron* is, intuitively speaking, a closed convex body whose surface decomposes into "flat" pieces. Mathematically, this flatness is grasped by the concept of *halfspaces*.



a hyperplane H and corresponding halfspace $\mathcal{H}$

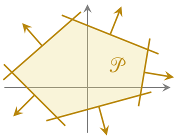
3.4 Definition (polyhedra and polytopes): Let $n \in \mathbb{N}$. A subset $H \subseteq \mathbb{R}^n$ is called a *hyperplane* of $\mathbb{R}^n$ if there exist $a \in \mathbb{R}^n$, $a \neq 0$ and $\beta \in \mathbb{R}$ such that

$$H = \left\{ x \in \mathbb{R}^n \colon a^T x = \beta \right\}.$$

Likewise, a (closed) *halfspace* of $\mathbb{R}^n$ is a set of the form

$$\mathcal{H} = \left\{ x \in \mathbb{R}^n \colon a^T x \leq \beta \right\},$$

where again $0 \neq a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$. One says in the above situation that the hyperplane or halfspace, respectively, is *induced by* the pair (a, β). The intersection of finitely many halfspaces is called a *polyhedron*. A *polytope* is a polyhedron that is bounded.



a 2-dimensional polytope (the affine hull is $\mathbb{R}^2$) defined by five halfspaces

The *dimension* $\dim(\mathcal{P})$ of a polyhedron $\mathcal{P}$ is defined to be the dimension of $\operatorname{aff}(\mathcal{P})$, if $\mathcal{P} \neq \emptyset$, and -1 otherwise. In both cases $\dim(\mathcal{P})$ is one less than the maximum number of affinely independent vectors in $\mathcal{P}$. ◁

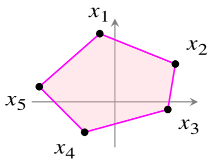
Polyhedra are the fundamental structure of linear and integer linear optimization. From the above definition, it follows that for a polyhedron $\mathcal{P}$ there is a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, for some $m \in \mathbb{N}$, such that

$$\mathcal{P} = \mathcal{P}(A, b) = \{x \in \mathbb{R}^n \colon Ax \leq b\},$$

i.e., $\mathcal{P}$ is the solution set of a system of linear inequalities, each of which defines a halfspace.

Note that polyhedra, being intersections of convex sets, are convex themselves. Complementary to the above *implicit* definition as the solution set of a system $Ax \leq b$, every polyhedron admits, by a theorem of Minkowski, an *explicit* characterization by means of convex and conic combinations.

3.5 Theorem (Minkowski): *The set $\mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedron if and only if there are finite sets $V, W \subseteq \mathbb{R}^n$ such that $\mathcal{P} = \operatorname{conv}(V) + \operatorname{conic}(W)$.* ◁



the same polytope as the convex hull of its five vertices $x_1, \dots, x_5$

If $\mathcal{P}$ in Theorem 3.5 is a polytope, then, since every nonempty cone is unbounded, $W = \emptyset$ must hold; hence, every polytope is the convex hull of its so-called *vertices* or *extreme points*, which are the "corners" of the polytope as shown in the picture on the margin. Vertices are a special instance of *faces* of a polyhedron, which are defined next.

3.6 Definition: Let $\mathcal{P} \subseteq \mathbb{R}^n$ be a polyhedron. An inequality of the form

$$a^T x \leq \beta \tag{3.6}$$

with $a \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ is said to be *valid* for $\mathcal{P}$ if it is satisfied for all $x \in \mathcal{P}$. In that event, the set

$$F_{a,\beta} = \left\{x \in \mathcal{P} : a^T x = \beta\right\}$$

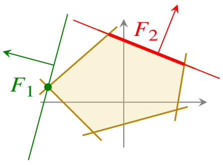
constitutes a *face* of $\mathcal{P}$, namely the *face induced by* (3.6). A zero-dimensional face is called a *vertex* of $\mathcal{P}$, one-dimensional faces are *edges* of $\mathcal{P}$. A face F for which $\dim(F) = \dim(\mathcal{P}) - 1$ is called a *facet* of $\mathcal{P}$. ◁

Note that a face of a polyhedron is a polyhedron itself, as it is obtained by adding the inequalities $\alpha^T x \leq \beta$ and $\alpha^T x \geq \beta$ to the system $Ax \leq b$. For a representation $\mathcal{P}(A, b)$ of the polytope $\mathcal{P}$, each face has another characterization.

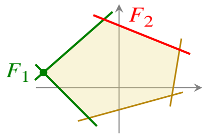
3.7 Lemma: *Let $\mathcal{P} = \mathcal{P}(A, b)$ with $A \in \mathbb{R}^{m \times n}$. For $E \subseteq \{1, \dots, m\}$, the set*

$$F_E = \left\{x \in \mathcal{P} : A_{E,\bullet} x = b_E\right\}$$

is a face of $\mathcal{P}$. If $F_E \neq \emptyset$, its dimension is $n - \operatorname{rank}(A_{\operatorname{eq}(F_E),\bullet})$, where $\operatorname{eq}(F_E)$ is the equality set *of F_E defined by $\operatorname{eq}(F_E) = \left\{i : A_{i,\bullet} x = b \text{ for all } x \in F_E\right\}$.* ◁



two faces induced by valid inequalities with $\dim(F_1) = 0$ and $\dim(F_2) = 1$



the same two faces with their defining equality sets highlighted

Facets are of special importance because they are necessary and sufficient to describe a polyhedron:

3.8 Theorem: *Let $\mathcal{P} = \mathcal{P}(A, b)$ be a polyhedron, and assume that no inequality in $Ax \leq b$ is redundant, i.e., could be removed without altering $\mathcal{P}$. Let $I \cup J$ denote the partition of row indices defined by $I = \left\{i : A_{i,\bullet} x = b_i \text{ for all } x \in \mathcal{P}\right\}$. Then, the inequalities in $A_{J,\bullet} x \leq b_J$ are in one-to-one correspondence (via Lemma 3.7) to the facets of $\mathcal{P}$.* ◁

To describe a polyhedron $\mathcal{P}$, we thus need only $n - \dim(\mathcal{P})$ equations plus as many inequalities as $\mathcal{P}$ has facets. Any inequality that induces neither a facet nor the whole polytope $\mathcal{P}$ can be dropped without changing the feasible set, and every system $\tilde{A}x \leq \tilde{b}$ describing $\mathcal{P}$ needs to include at least one facet-inducing inequality for every facet of $\mathcal{P}$.

3.2 Linear Programming

A *linear program (LP)* is an optimization problem that asks for the minimization of a linear functional over a polyhedron. The most simple form would be

$$\begin{aligned} \min \quad & c^T x && \text{(3.7a)} \\ \text{s.t.} \quad & Ax \leq b, && \text{(3.7b)} \end{aligned}$$

while an LP is said to be in *standard form* if it is stated as follows:

$$\begin{aligned} \min \quad & c^T x && \text{(3.8a)} \\ \text{s.t.} \quad & Ax = b && \text{(3.8b)} \\ & x \geq 0. && \text{(3.8c)} \end{aligned}$$

In both cases, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given, and we denote the feasible set by the letter $\mathscr{P}$ (for (3.7), $\mathscr{P} = \mathscr{P}(A, b)$ in the notation of Section 3.1). Note that both forms are equivalent in the sense that each can be transformed into the other. For example, given an LP in polyhedral form, we can replace x by variables $x^+ \in \mathbb{R}^n$ and $x^- \in \mathbb{R}^n$, representing the positive and negative part of x, respectively, and introduce auxiliary variables $s \in \mathbb{R}^m$ to rewrite (3.7) in standard form as

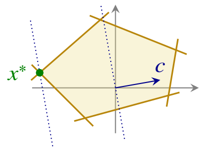
$$\begin{aligned} \min \quad & c^T x^+ - c^T x^- \\ \text{s.t.} \quad & Ax^+ - Ax^- + s = b \\ & x^+ \geq 0,\ x^- \geq 0,\ s \geq 0. \end{aligned}$$

Moreover, if we had a *maximization* problem with objective max $c^T x$, it could be converted to the above forms by the relation

$$\max \left\{ c^T x \colon x \in \mathscr{P} \right\} = -\min \left\{ -c^T x \colon x \in \mathscr{P} \right\}.$$

In view of this equivalence of different LP forms, we will in the following assume whatever form allows for a clear presentation.

Note that if (3.7) has an optimal solution x^* with objective value $z^* = c^T x^*$, we can represent the set of optimal solutions by $\left\{ x \colon Ax \leq b \text{ and } c^T x = z^* \right\}$. This shows that the optimal set is always a *face* of $\mathscr{P}$. In addition, it is easy to show that if $\mathscr{P}$ has *any* vertex (which is always the case if the LP is in standard form), then *every* nonempty face of $\mathscr{P}$ contains a vertex. Hence we can conclude:



example objective c such that $c^T x$ is minimized for x^*; the dotted lines show the hyperplanes $c^T x = 0$ and $c^T x = c^T x^*$

3.9 Observation: *If an LP in standard form has a finite optimal objective value, there is always a vertex of $\mathscr{P}$ which is an optimal solution of the LP.* ◁

The rest of this section is about characterizing and algorithmically finding such an optimal vertex.

3.2.1 Duality

One of the most important concepts in linear programming is that of *duality*, meaning that certain structures always occur in closely related *pairs*. In optimization, those structures include convex cones and systems of linear (in)equalities. For our purposes, the most important result is that of *LP duality*, a close relation of two LPs, as reviewed below.

3.10 Definition: Let an LP in standard form (3.8) be given; we call this the *primal* problem. The associated LP

$$\max \quad b^T y \tag{3.9a}$$

$$\text{s.t.} \quad A^T y \le c \tag{3.9b}$$

is called the linear-programming *dual* of (3.8). ◁

Since we have seen that any LP can be transformed into standard form, one can also compute a dual for every LP. In particular, it is easy to verify that the dual of the dual results in the primal again. The motivation for LP duality lies in the following fundamental theorem.

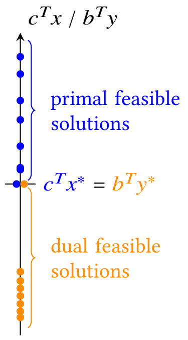
3.11 Theorem (strong duality): *Assume that either* (3.8) *or* (3.9) *are feasible. Then*

$$\min \left\{ c^T x \colon Ax = b,\ x \ge 0 \right\} = \max \left\{ b^T y \colon A^T y \le c \right\},$$

where we include the values $\pm\infty$ *as described on page 9. If both are feasible, then both have an optimal solution.* ◁

Note that Theorem 3.11 implies the statement of *weak duality*, namely that whenever x is feasible for the primal and y is feasible for the dual, then $c^T x \ge b^T y$.

LP duality is extremely useful because it allows for very compact proofs of optimality: if one wants to show that a certain solution x^* of the primal LP is optimal, it suffices to provide a dual feasible y^* with the property that $c^T x^* = b^T y^*$. Such a y^* is called a *witness* for the optimality of x^*.



3.2.2 Primal and Dual Basic Solutions

In this section, we show how to represent a vertex of $\mathscr{P}$ by means of a *basis*. It is assumed that an LP is given in standard form (3.8) and that A has full row rank m.

By Lemma 3.7, any vertex $\bar{x}$ of $\mathscr{P}$, which is a 0-dimensional face, can be characterized by a subset of the constraints of (3.8) that is fulfilled with equality and has rank n. Since (3.8b) has rank m by assumption, $\bar{x}$ is a vertex if and only if there is an index set N with $|N| = n - m$ such that $\bar{x}$ is the unique solution of the system

$$Ax = b,\ x_N = 0. \tag{3.10}$$

For $i \in N$ we can thus disregard the corresponding i-th column of the system $Ax = b$. Hence, we can represent $\bar{x}$ by m linearly independent columns of A. Such a submatrix of A is called a *(simplex) basis* and the corresponding set of column indices is denoted by $B = \{1, \dots, n\} \setminus N$. The variables x_B are called the *basic variables*, x_N are the *non-basic variables*. By (3.10), every vertex $\bar{x}$ is a *basic solution* for (3.8b), i.e.,

$$\bar{x}_B = A_{\bullet,B}^{-1} b \quad \text{and} \quad \bar{x}_N = 0 \tag{3.11}$$

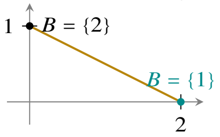
for a basis B of A. Conversely, an arbitrary basic solution of the form (3.11) is a vertex of $\mathscr{P}$ only if additionally $\bar{x} \geq 0$, i.e., it is feasible for (3.8), then called a *basic feasible solution (BFS)* of the LP. Concludingly, $\bar{x}$ being a BFS is necessary and sufficient for $\bar{x}$ being a vertex of $\mathscr{P}$, while it should be noted that, in general, more than one BFS may correspond to the same vertex.

Example LP:

$$\begin{pmatrix}\frac{1}{2} & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\end{pmatrix} = 1$$

$$x_1, x_2 \geq 0$$

For $B = \{1\}$, $A_{\bullet,B}^{-1} b = 2 \cdot 1 = 2$, so $\bar{x} = (\bar{x}_B, \bar{x}_N) = (2, 0)$ is the corresponding BFS.



Let B be a basis of A and denote the feasible region of the dual (3.9) by $\mathscr{D}$. Arguing similarly as above, one can show that a vertex $\bar{y}$ of $\mathscr{D}$ must fulfill $A_{\bullet,B}^T \bar{y}_B = c_B$ (read $A_{\bullet,B}^T$ as $(A_{\bullet,B})^T$) and, in order to be feasible, also

$$c - A^T y \geq 0 \tag{3.12}$$

needs to hold. Hence the vector $\bar{y}$ defined by $\bar{y}^T = c_B^T A_{\bullet,B}^{-1}$ is called the *dual basic solution* associated to $\bar{x}$ defined in (3.11); it is a *dual BFS* if $\bar{y} \in \mathscr{D}$, i.e., if (3.12) holds.

3.2.3 The Simplex Method

If $\bar{x}$ and $\bar{y}$ are an associated pair of primal and dual basic solutions for a basis B of A, it holds that

$$c^T \bar{x} = c_N^T \bar{x}_N + c_B^T \bar{x}_B = 0 + c_B^T A_{\bullet,B}^{-1} b = b^T \bar{y},$$

i.e., the objective values of $\bar{x}$ and $\bar{y}$ for the primal and dual LP, respectively, coincide. In view of Observation 3.9 and Theorem 3.11 this shows that solving an LP is tantamount to finding a basis B for which the associated primal and dual basic solutions $\bar{x}$ and $\bar{y}$ are both feasible ($\bar{y}$ then is a witness of the optimality of $\bar{x}$). The several variants of the *simplex method* comprise algorithms that determine such a basis by a sequence of *basis exchange* operations in each of which a single element of B is exchanged.

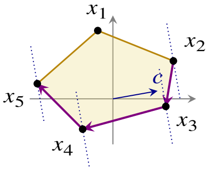
To be more specific, denoting the objective value by $z = c^T x$, by simple calculations starting from the form $A_{\bullet,B} x_B + A_{\bullet,N} x_N = b$ of (3.8b) we obtain the following representation

$$\begin{array}{rclclcrcl} z &=& c_B^T A_{\bullet,B}^{-1} b &+& (c_N^T - c_B^T A_{\bullet,B}^{-1} A_{\bullet,N}) x_N &=& \bar{z} &+& \bar{c}_N^T x_N \\ x_B &=& A_{\bullet,B}^{-1} b &-& A_{\bullet,B}^{-1} A_{\bullet,N} x_N &=& \bar{b} &-& \bar{A}_N x_N \end{array} \tag{3.13}$$

of z and x_B with respect to B in dependence of the values of x_N, where in the second step we have introduced suitable abbreviations $\bar{z}$, $\bar{b}$, $\bar{c}_N$ and $\bar{A}_N$. In this form, we can immediately read off the values $\bar{b}$ of the basic variables and the objective value $\bar{z}$ for the current basic solution that is defined by $x_N = 0$. The vector $\bar{c}_N^T = (c_N^T - \bar{y}^T A_N)$ encodes the dual feasibility (3.12) of that basis. Consequently B must be an optimal basis if both $\bar{b} \geq 0$ (primal feasibility) and $\bar{c}_N \geq 0$ (dual feasibility) hold in (3.13).

Otherwise, we can perform a *simplex step*: assume that the (i,k)-th entry of $\bar{A}_N$ is non-zero. It can be shown that by performing a Gaussian *pivot* on that entry, i.e., turning the relevant column of (3.13) into a unit vector by elementary row operations, one essentially computes a representation of the form (3.13) with respect to the *adjacent basis* $B' = B \setminus \{i\} \cup \{j\}$, where $j \in N$ is the k-th entry of N. This notion of *adjacency* translates to the geometric interpretation, since vertices corresponding to adjacent basises always share an edge of the polyhedron.

The *primal simplex algorithm* starts with a primal BFS (consult the literature for a method called *phase 1* to find such an initial BFS) and then iteratively performs the following steps:



example run of the primal simplex, starting in x_2 and performing three basis exchanges until the optimal x_5 is reached; $c^T x_i$ is decreasing on the path

(1) Choose a column (variable entering the basis) for which $\bar{c}_N$ is negative, i.e., the corresponding entry of $\bar{y}$ not yet dually feasible. This ensures that z is nonincreasing, and it usually decreases.

(2) Choose a row (variable leaving the basis) in such a way as to ensure that the subsequent simplex step maintains primal feasibility; this can be achieved by a simple test called *min-ratio rule*.

(3) Perform the simplex step by pivoting on the column and row selected above.

The corresponding sequence of objective function values is nonincreasing. Under simple conditions on the method of selecting indices, one can show that this procedure results in an optimal basis, indicated by $\bar{c}_N \geq 0$, after a finite number of steps.

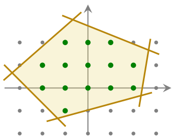
The *dual simplex algorithm*, as the name suggests, sets off from a dual BFS ($\bar{c}_N \geq 0$) and then does essentially the same as its primal counterpart (with the role of rows and columns of (3.13) swapped during the basis exchange), maintaining dual feasibility and a nondecreasing objective function until primal feasibility ($\bar{b} \geq 0$) is established.

Numerous variants and optimizations of the basic method described above exist. An important one is the so-called *revised simplex* which is based on the observation that, especially for $n \gg m$, it is wasteful to pivot the complete system (3.13) in each step. Instead, one maintains a representation of $A_{\bullet,B}^{-1}$ (usually in the form of an *LU factorization*), which can be shown to be sufficient to carry out an iteration of the algorithm. Furthermore, it should be noted that there exists an efficient method to incorporate *upper bounds* on the variables, e.g. of the form $0 \leq x \leq 1$, without having to increase the size of the formulation by the explicit addition of constraints $x \leq 1$. Paper V compares several variants of the simplex algorithm when applied to LP decoding as introduced in Section 5.2.

It has been shown that the worst-case complexity of the simplex algorithm is exponential in the problem size [10]. The very contrary *empirical* observation however is that the number of pivots before optimality is usually in $O(m)$. This explains why, although LP solving algorithms with polynomial worst-case complexity exist, the simplex method is still the most prevalent one in practice.

3.3 Integer Programming

In integer programming, we are concerned with LPs augmented by the additional requirement that the solution be *integral*. Formally, we define an *integer linear program (IP)* as an optimization problem of the form

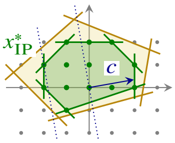


feasible set $\mathscr{P} \cap \mathbb{Z}^2$ and LP relaxation of an IP

$$
\begin{aligned}
\min \quad & c^T x && \text{(3.14a)}\\
\text{s.t.} \quad & Ax \le b && \text{(3.14b)}\\
& x \in \mathbb{Z}^n, && \text{(3.14c)}
\end{aligned}
$$

where we assume that all entries of A, b and c are rational. The LP that results when replacing (3.14c) by $x \in \mathbb{R}^n$ is called its *LP relaxation*. Let $\mathscr{P} = \mathscr{P}(A, b)$ as before denote the feasible set of the LP relaxation and $\mathscr{P}_\mathrm{I} = \operatorname{conv}(\mathscr{P}(A, b) \cap \mathbb{Z}^n)$ the convex hull of integer points in $\mathscr{P}$. Under the above assumption, it can be shown that $\mathscr{P}_\mathrm{I}$ is a polyhedron. Note that solving (3.14) is essentially equivalent to solving $\min\{c^T x \colon x \in \mathscr{P}_\mathrm{I}\}$. Thus an IP can, in principle, be solved by an *LP*: if A' and b' are such that $\mathscr{P}_\mathrm{I} = \mathscr{P}(A', b')$, then the optimal IP solution is also optimal for the LP

$$
\begin{aligned}
\min \quad & c^T x\\
\text{s.t.} \quad & A' x \le b'\\
& x \in \mathbb{R}^n.
\end{aligned}
$$

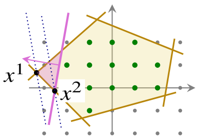


equivalent LP polytope $\mathscr{P}_\mathrm{I} = \operatorname{conv}(\mathscr{P} \cap \mathbb{Z}^2)$ and optimal IP solution x^*_IP

The problem is that, in general, it is hard to derive a description of $\mathscr{P}_\mathrm{I}$ from $\mathscr{P}$. In fact, IPs are NP-hard to solve in general, whereas we have seen above that linear programming is contained in P.

Two complementary approaches for solving IPs are important for this thesis: one is to tighten the LP relaxation (3.14) by adding *cuts*, i.e., inequalities that are valid for $\mathscr{P}_\mathrm{I}$ but not for $\mathscr{P}$. The other, named *branch-&-bound*, is about recursively dividing the feasible space $\mathscr{P} \cap \mathbb{Z}^n$ into smaller subproblems among which the optimal solution is searched, interleaved with the generation of bounds that allow to skip most of these subproblems.

3.3.1 Cutting Planes



example cutting plane that cuts the non-integral initial LP solution x^1 but is valid for $\mathscr{P}_\mathrm{I}$; it implies a new LP solution x^2 with improved objective value

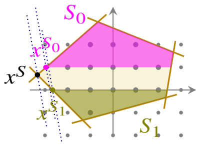
Assume that we try to solve the IP (3.14) by solving its LP relaxation, i.e., minimize $c^T x$ over $\mathscr{P}$ instead of $\mathscr{P}_\mathrm{I}$. If the LP solution x^1 happens to be integral, it is clear that x^1 is also optimal for (3.14). Otherwise, by Theorem 3.8 there must exist at least one inequality that is valid for $\mathscr{P}_\mathrm{I}$ but not for x^1. Any such inequality is called a *cutting plane* or simply *cut* for x^1. If we add a cut to the LP relaxation (3.14) and solve the LP again, we necessarily get a new solution $x^2 \neq x^1$ (because the cut is violated by x^1). Since the feasible space was reduced, $c^T x^2 \ge c^T x^1$ must hold.

This method can be iterated as long as new cuts for the current solution x^i can be generated, leading to a sequence of LP solutions $(x^i)_i$ such that $(c^T x^i)_i$ is monotonically increasing. If the cuts are “good enough” (especially if they include the facets of $\mathcal{P}_I$), some x^k will eventually be feasible for $\mathcal{P}_I$ and hence equal the IP solution x^*. While there exists a cut-generation algorithm, called *Gomory-Chvátal method*, that provably terminates in x^* after a finite number of steps, the number of cuts it usually introduces is prohibitive for practical applications. For many classes of IPs, however, there exist special methods to derive cuts that are based on the specific structure of the problem—we will encounter such a case in Section 5.3.

3.3.2 Branch-&-Bound

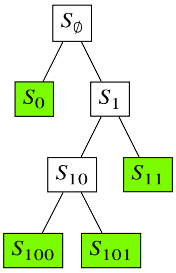
Let us again assume that we solve the LP relaxation of (3.14) and obtain a non-integral LP solution $x^\emptyset \in \mathcal{P} \setminus \mathbb{Z}^n$ with, say, $x_i^\emptyset \notin \mathbb{Z}$. The key idea of LP-based *branch-&-bound* is to define two disjoint subproblems S_0 and S_1 of the original problem $S_\emptyset = (3.14)$ with the property that the optimal IP solution x^* of $S_\emptyset$ is contained in either S_0 or S_1. To be specific, note that necessarily $x_i^* \in \mathbb{Z}$, so either $x_i^* \leq \lfloor x_i^\emptyset \rfloor$ or $x_i^* \geq \lceil x_i^\emptyset \rceil$ needs to hold, which gives rise to the two subproblems

$$S_0: \quad \begin{array}{rl} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & x_i \leq \lfloor x_i^\emptyset \rfloor \\ & x \in \mathbb{Z}^n \end{array} \qquad \text{and} \qquad S_1: \quad \begin{array}{rl} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & x_i \geq \lceil x_i^\emptyset \rceil \\ & x \in \mathbb{Z}^n. \end{array}$$



all integral points in $\mathcal{P}$ are contained in either $S_0 = \{x \in \mathcal{P} : x_2 \geq 1\}$ or $S_1 = \{x \in \mathcal{P} : x_2 \leq 0\}$, the two solutions x^{S_0} and x^{S_1} have larger objective value than x^S

For both, we can again solve the LP relaxation (with the additional constraint on x_i) to obtain LP solutions x^0 and x^1 with objective function values z^0 and z^1, respectively. If both solutions are integral, clearly the one with smaller objective function value is optimal for (3.14). Otherwise, the process can be recursed to split a subproblem into two sub-subproblems (e.g., S_{00} and S_{01}), and so forth, creating a binary tree of “problem nodes” whose leaves accord to problems that either have an integral LP solution or are infeasible—in both cases, no further subdivision is necessary.



an example b&b enumeration tree; the leaves are not further explored because the respective LPs are either infeasible or have integral solutions

While this technique already reduces the search space in a substantial way, using advanced *bounding* allows to further reduce the size of the abovementioned tree. Note that any *feasible* solution $\hat{x} \in \mathcal{P} \cap \mathbb{Z}^n$ gives an upper bound on the optimal objective value $z^* = c^T x^*$, i.e., $c^T \hat{x} \geq c^T x^*$. Furthermore, in any subproblem S, the objective value z^S of the optimal solution x^S to the corresponding LP relaxation is a *lower bound* on the optimal integral solution value $\hat{z}^S$ *of that subproblem.* If we now solve the LP relaxation of some subproblem S obtaining z^S and it holds that $z^S \geq c^T \hat{x}$ for any feasible $\hat{x}$ that has been found earlier, there is no need to subdivide S (even if x^S is not integral): no feasible point of S can improve upon the objective value of $\hat{x}$.

If we otherwise split S into subproblems S' and S'' and solve the respective LP-relaxations to obtain $z^{S'}$ and $z^{S''}$, we can conclude that $\min\{z^{S'}, z^{S''}\} \leq \hat{z}^S = c^T \hat{x}^S$, i.e., the smaller of both is a lower bound on $\hat{z}^S$, because the optimal integral solution $\hat{x}^S$ for S must be in either S' or S''. If this minimum is larger than z^S, we can *improve* the lower bound for S. This bound update can possibly be propagated to the parent of S if S has a sibling for which a lower bound has already been computed, and so forth.

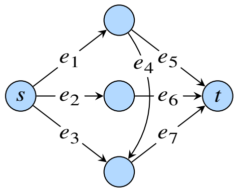
The algorithm terminates if there are either no unexplored subproblems left, or if a feasible solution $\hat{x}$ for (3.14) and a lower bound $z^{\emptyset}$ on the optimal objective value z^* has been found such that $\hat{x} = z^{\emptyset}$: it is then clear that no better solution than $\hat{x}$ exists, so $\hat{x}$ must be optimal.

In practice, the cutting-plane and branch-&-bound approach are often interleaved within a *branch-&-cut* algorithm, where cutting planes might be inserted in each node of the branch-&-bound tree. The algorithm presented in Paper VI is based on such a branch-&-cut strategy.

3.4 Combinatorial Optimization

An optimization problem is called *combinatorial* if the feasible set comprises all subsets of a finite ground set $\Xi = \{\xi_1, \dots, \xi_n\}$ that fulfill a certain property, and the objective value of an $S \subseteq \Xi$ has the form $f(S) = \sum_{\xi \in S} c_\xi$ for given cost values $c_\xi \in \mathbb{R}$ associated to each element $\xi \in \Xi$.

A popular example is the *shortest-path problem*: given a graph $G = (V, E)$, two vertices $s, t \in V$ and cost c_e associated to each edge $e \in E$, it asks for an s–t path P^* (in fact, only paths in which each edge occurs at most once are allowed) with minimum total cost $f(P^*)$, where the objective function $f(P) = \sum_{e \in P} c_e$ accumulates, for each edge e visited by the path P, the cost value c_e assigned to e. Here, the ground set consists of the set of edges $\Xi = E$, and a subset of edges is feasible if it forms (in appropriate ordering) a path in G.



$\Xi = \{e_1, \dots, e_7\}$; feasible are $\{e_1, e_5\}$, $\{e_1, e_4, e_7\}$, $\{e_2, e_6\}$, and $\{e_3, e_7\}$

In a combinatorial optimization problem, one can identify each subset $S \subseteq \Xi$ by its characteristic (or incidence) vector $x^S \in \{0, 1\}^n$, where

$$x_i^S = \begin{cases} 1 & \text{if } \xi_i \in S, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the set $X = \{x^S : S \subseteq \Xi \text{ feasible}\}$ that represents the feasible solutions is a subset of $\{0, 1\}^n$. As furthermore $f(S) = c^T x^S$ with $c = (c_{\xi_1}, \dots, c_{\xi_n})^T$, every combinatorial optimization problem with such an objective function can be represented by the LP

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & x \in \operatorname{conv}(X) \end{aligned}$$

whose feasible polytope $\mathscr{P}$ is a subset of the unit hypercube $[0, 1]^n$.

In case of the shortest s–t path problem, an explicit formulation of the *path polytope*, i.e., the convex hull of incidence vectors of s–t paths, is known. The corresponding LP to solve the shortest path problem is

$$\begin{aligned}
\min \quad & c^T x = \sum_{e \in E} c_e x_e && && \text{(3.15a)}\\
\text{s.t.} \quad & \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e = 1 && && \text{(3.15b)}\\
& \sum_{e \in \delta^+(t)} x_e - \sum_{e \in \delta^-(t)} x_e = -1 && && \text{(3.15c)}\\
& \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = 0 && \text{for all } v \notin \{s, t\} && \text{(3.15d)}\\
& x \in [0, 1]^{|E|}, && && \text{(3.15e)}
\end{aligned}$$

where (3.15b) and (3.15c) ensure that the path starts in s and ends in t, respectively, and the so-called *flow conservation constraints* (3.15d) state that the path must leave any other vertex v as often as it enters v.

In general, however, it is highly nontrivial to find an explicit representation of $\mathscr{P}$. For a large class of hard problems one can nevertheless at least formulate some LP-relaxation $\mathscr{P}' \supseteq \operatorname{conv}(X)$ such that $\mathscr{P}' \cap \{0, 1\}^n = X$, i.e., there is an *integer* programming model

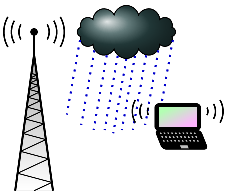
$$\begin{aligned}
\min \quad & c^T x\\
\text{s.t.} \quad & x \in \mathscr{P}'\\
& x \in \{0, 1\}^n
\end{aligned}$$

of the problem, which can then be tackled by the methods presented in Section 3.3. For several problems—in this thesis, most notably the *decoding* problem introduced in the following chapter—this polyhedral approach to combinatorial optimization has led to the most efficient algorithms known.

Chapter 4

Coding Theory Background

How can information be transmitted *reliably* via an *error-prone* transmission system? The mathematical study of this question initiated the emergence of *information theory* and, since one particularly important answer lies in the use of error-correcting codes, of *coding theory* as a mathematical discipline of its own right.



a noisy data transmission scenario

In contrast to *physical* approaches to increase the reliability of communication (e.g. increased transmit power, more sensitive antennas, ...), error-correcting codes offer a completely *ideal* solution to the problem. It has been shown that, by coding, an arbitrary level of reliability is achievable, despite the unavoidable inherent unreliability of any technological equipment—in fact, the unparalleled development of microelectronic devices and their ability to communicate via both wired and wireless networks would not have been possible without the accompanying progresses in coding theory.

This chapter reviews the basics of error-correcting codes and their application to reliable communication. For a more complete coverage of these topics, we refer the interested reader to the broad literature on the subject: the birth of information theory was Shannon's seminal work "A mathematical theory of communication" [11]. Recommendable textbooks on information theory and its applications are [12, 13]. There are several books covering "classical" coding theory (this term is elucidated later), e.g. [14], while modern aspects of coding are collected in [15].

4.1 System Setup and the Noisy-Channel Coding Theorem

The principle of error-correction coding is to preventively include *redundancy* in the transferred messages, thus communicating *more* than just the actual information, with the goal of enabling the receiver to recover that information, even in the presence of noise on the transmission channel. The general system setup we consider is as depicted in Figure 4.1:

- the function by which these "bloated" messages are computed from the original ones is called the *encoder*;
- the *channel* introduces noise to the transmitted signal, i.e., at the receiver there is *uncertainty* about what was actually sent;

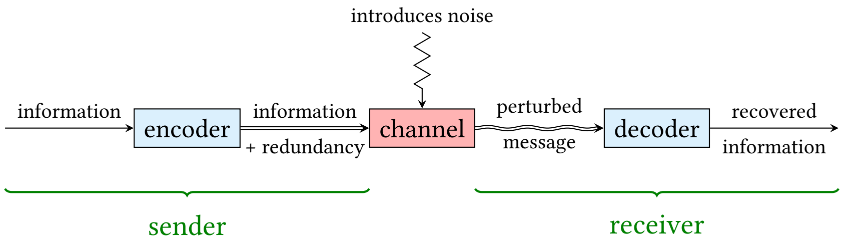
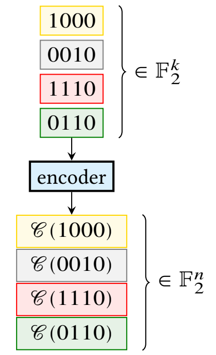


Figure 4.1: Model of the transmission system.

- the *decoder* tries to recover the original information from the received signal.

Throughout this text, we are concerned only with *block codes*, which means that the information enters the encoder in form of chuncks of uniform size (the *information words*), each of which is encoded into a unique coded message of again uniform (but larger) size (the *codewords*). For now, we additionally restrict ourselves to the *binary* case, i.e., the alphabet for both information and codewords is $\mathbb{F}_2 = \{0, 1\}$. This leads to the following definitions.



block encoding of information words into codewords

4.1 Definition (code): An (n, k) *code* is a subset $\mathscr{C} \subseteq \mathbb{F}_2^n$ of cardinality 2^k. A bijective map from $\mathbb{F}_2^k$ onto $\mathscr{C}$ is called an *encoding function for* $\mathscr{C}$.

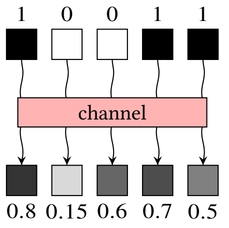
Whenever the context specifies a concrete encoding function, and if there is no risk of ambiguity, the symbol $\mathscr{C}$ will be used interchangeably for both the code and its encoding function.

The numbers k and n are referred to as *information length* and *block length*, respectively. Their quotient $r = k/n < 1$ represents the amount of information per coded bit and is called the *rate* of $\mathscr{C}$.

◁

The concept of redundancy is entailed by the fact that $\mathscr{C}$ is a strict (and usually very small) subset of the space $\mathbb{F}_2^n$, i.e., most of the vectors in $\mathbb{F}_2^n$ are *not* codewords, which is intended to make the codewords much easier to distinguish from each other than the informations words of $\mathbb{F}_2^k$.

Note that in this definition the encoder is secondary to the code. This reflects the fact that, for the topics covered by this text, the structure of the set of codewords is more important than the actual encoding function.



transmission of five bits through a noisy channel with $\mathscr{Y} = [0, 1]$

We assume that the channel through which the codewords are sent is *memoryless*, i.e., that the noise affects each individual bit independently; it thus can be defined as follows.

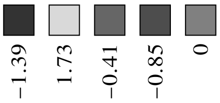
4.2 Definition (binary-input memoryless channel): A *binary-input memoryless channel* is characterized by an output domain $\mathscr{Y}$ and the two conditional probability functions

$$P(y_i \mid x_i = 0) \quad \text{and} \quad P(y_i \mid x_i = 1) \tag{4.1}$$

that specify how the output $y_i \in \mathscr{Y}$ depends on the two possible inputs 0 and 1, respectively (we assume that these two functions are not identical—otherwise, the output would be independent of the input and would thus not contain any information about the latter). Even more compactly, the frequently used *log-likelihood ratio (LLR)*

$$\lambda_i = \ln\left(\frac{P(y_i \mid x_i = 0)}{P(y_i \mid x_i = 1)}\right) \tag{4.2}$$

represents the entire information revealed by the channel about the sent symbol x_i. If $\lambda_i > 0$, then $x_i = 0$ is more likely than $x_i = 1$, and vice versa if $\lambda_i < 0$. The absolute value of λ_i indicates the *reliability* of this tendency. ◁

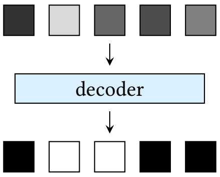


example LLR values for the above transmission

When the receiver observes the result $\lambda \in \mathbb{R}^n$ (we mostly use LLRs in favor of $y \in \mathcal{Y}^n$ from now on) of the transmission of an encoded information word $x = \mathscr{C}(u)$ through the channel, it has to answer the following question: *which codeword $x \in \mathscr{C}$ do I believe has been sent, under consideration of y?* This “decision maker” is called the *decoder*, which is an algorithm realizing a decoding function

$$\textsc{decode}\colon \mathbb{R}^n \to \mathbb{R}^n; \tag{4.3}$$

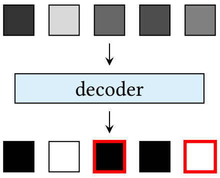
the decoder is intentionally (for reasons that will become clear later) allowed to output not only elements of $\mathbb{F}_2^n$ but arbitrary points of the unit hypercube $[0,1]^n$ (which includes $\mathbb{F}_2^n$ via the canonical embedding). We speak of *decoding success* if $x \in \mathscr{C}$ was sent and $\textsc{decode}(\lambda) = x$, while a *decoding error* occurs if $\textsc{decode}(\lambda) = x' \neq x$, which includes the cases that $x' \in \mathscr{C}$, i.e., the decoder outputs a codeword but not the one that was sent, and $x' \notin \mathscr{C}$, i.e., the decoder does not output a codeword at all.



decoding success for the above transmission

Assuming a *uniform prior* on the sender's side, i.e., that all possible information words occur with the same probability (*source coding*, the task of accomplishing this assumption, is not covered here), the error-correcting performance of a communication system consisting of code, channel, and decoding algorithm can be evaluated by means of its average *frame-error rate*

$$\text{FER} = \frac{1}{|\mathscr{C}|}\sum_{x \in \mathscr{C}} P\left(\textsc{decode}(\lambda) = x \mid x \text{ was sent}\right). \tag{4.4}$$



decoding failure (two faulty bits) for the above transmission

We can now state the main task of coding theory: given a certain channel model (4.1), design an (n,k) code and an accompanying decoder (4.3) such that the demands requested by the desired application are fulfilled, which may include:

- The frame-error rate (4.4) should be sufficiently small in order to ensure reliable communication.
- The rate $r = k/n$ should be as large as possible, because a small rate corresponds to a large number of transmitted bits per information bit, i.e., large coding overhead.
- The block length n should be small: high block lengths generally increase the complexities of both encoder and decoder and may additionally introduce undesirable latencies in e.g. telephony applications.
- The complexity of the decoding algorithm needs to be appropriate.

It is intuitively clear that some of the above goals are opposed to each other. The first two, however, are not as incompatible as one might suspect—Claude Shannon proved a stunning result [11] which implies that, at a *fixed* positive code rate, the error probability can be made arbitrarily small.

4.3 Theorem (noisy-channel coding theorem): *For given* $\varepsilon > 0$ *and* $r < C$*, where* $C > 0$ *depends only on the channel, there exists a code* $\mathscr{C}$ *with rate at least* r*, and a decoding algorithm for* $\mathscr{C}$ *such that the frame-error rate* (4.4) *of the system is below* ε*.* ◁

As beautiful as both the result and its proof (which is explained thoroughly in [12]) are, they are unfortunately completely non-constructive in several ways:

- the decoding error probability vanishes only for the block length n going to infinity;
- the proof makes use of a *random coding* argument, hence it does not say anything about the performance of a concrete, finite code;
- the running time of the theoretical decoding algorithm used in the proof is intractable for practical purposes.

As a consequence of the first two aspects, the search for and construction of "good" codes, i.e., codes that allow for the best error correction at a given finite block length n and rate r, has emerged as an research area on itself, which is nowadays often nicknamed "classical coding theory." For a long time, however, the problem of decoder complexity was not a major focus of the coding theory community. The term "modern coding theory" nowadays refers to a certain paradigm shift that has taken place since the early 1990's, governed by the insight that *suboptimal codes* which are develeoped *jointly* with harmonizing low-complexity decoding algorithms can lead to a higher overall error-correcting performance in practical applications than the "best" codes, if no decoder is able to exploit their theoretical strength within reasonable running time (see [16] for the historical development of coding theory).

The rest of this chapter is organized as follows. In Section 4.2, we discuss both the optimal MAP and the ML decoding rule, which are equivalent in our case. Afterwards, the prevalent additive white Gaussian noise (AWGN) channel model is explained in Section 4.3. Section 4.4 introduces binary linear block codes, a subclass of general block codes that is most important in practice and with some exceptions assumed throughout this thesis. A special type of linear block codes, called turbo codes, is presented in Section 4.5. Finally, Section 4.6 explains how codes and channels can be generalized to the non-binary case.

Note that this chapter does not cover any specific *decoding* algorithm, as both Chapter 5 and the entire Part II are concerned with various decoding approaches using mathematical optimization. For other decoding algorithms, e.g. the ones that are used in today's electronic devices, we refer to the literature.

4.2 MAP and ML Decoding

An optimal decoder (with respect to frame-error rate) would always return the codeword that was sent *with highest probability* among all codewords $x \in \mathscr{C}$, given the observed channel output y:

$$x_{\text{MAP}} = \arg\max_{x \in \mathscr{C}} P(x \mid y). \tag{4.5}$$

This is called *MAP decoding*. By Bayes' theorem, we have

$$P(x \mid y) = \frac{P(y \mid x) P(x)}{P(y)}.$$

Since $P(y)$ is independent of the sent codeword x and by assumption $P(x)$ is constant on $\mathscr{C}$, we obtain the equivalent *ML decoding* rule:

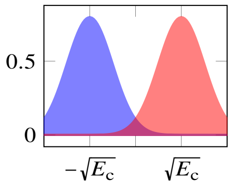
$$x_{\text{ML}} = \arg\max_{x \in \mathscr{C}} P(y \mid x). \tag{4.6}$$

Unfortunately, ML decoding is NP-hard in general [17], which motivates the search for special classes of codes that are both strong and allow for an efficient decoding algorithm which at least approaches the ML error-correction performance. On the other hand, it is desirable to know the frame-error rate for a given code under *exact* ML decoding, because it *(a)* constitutes the ultimate theoretical performance measure of the code itself and *(b)* serves as a "benchmark" for the quality of suboptimal decoding algorithms.

4.3 The AWGN Channel

The most immediate and simple example of a binary-input memoryless symmetric channel as defined in Definition 4.2 is called the *binary symmetric channel (BSC)*: it flips a bit with probability $p < 1/2$ and correctly transmits it with probability $q = 1 - p$, and hence $\lambda_i = \pm \ln(p/q)$, i.e., there are only two possible channel outputs.

While the conceptual simplicity of the BSC is appealing, for practical applications it turns out to be simplified too much. Imagine a device in which some incoming electromagnetic signal is translated by a circuit (consisting of e.g. antenna, electronic filters etc.) into a voltage v with expected values v_0 and v_1 for the transmission of a 0 and 1, respectively. For a BSC channel model, we could round v to the closest of those values and pass that "hard" information (either 0 or 1) to the decoder. But clearly, knowing *how far* v is from the value it is rounded to contains valuable information about the *reliability* of the received signal—a value of v close to the mean $(v_1 - v_0)/2$ is less reliable than one close to either v_0 or v_1 (cf. the figures along Definition 4.2). Consequently, the decoder should take that "soft" information into account.



densities $P(y_i \mid x_i = 1)$ (left) and $P(y_i \mid x_i = 0)$ (right) of an AWGN channel

The most prominent *soft-output* channel model is the *AWGN channel*, in which independent Gaussian noise (as it appears frequently in nature) is added to each transmitted symbol. It is characterized by a Gaussian distribution with mean $(-1)^{x_i}\sqrt{E_\mathrm{c}}$ and variance σ^2. Here, E_c is the transmit energy per channel symbol and σ^2 is the noise energy of the channel [15]:

$$P(y_i \mid x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\cdot\left(\frac{y-(-1)^{x_i}\sqrt{E_\mathrm{c}}}{\sigma}\right)^2}. \tag{4.7}$$

Note that the AWGN channel challenges the conceptual distinction between *transmission success* and *transmission error* in favor of an ubiquitous presence of noise: the expected value $\pm\sqrt{E_\mathrm{c}}$ that corresponds to a "noiseless" transmission will be received only with probability zero.

For a given code rate r, an AWGN channel can be specified by a single quantity, called *information-oriented signal-to-noise ratio (SNR)*

$$\mathrm{SNR_b} = \frac{E_\mathrm{b}}{N_0} = \frac{E_\mathrm{c}}{r\cdot 2\sigma^2} \tag{4.8}$$

where $E_\mathrm{b} = E_\mathrm{c}/r$ is the energy per *information* bit and $N_0 = 2\sigma^2$ is called the *double-sided power spectral density.* It can be shown that the i-th LLR value λ_i of an AWGN channel is itself a normally distributed random variable,

$$\lambda_i \sim \mathcal{N}\left(4r(-1)^{x_i}\cdot \mathrm{SNR_b}, 8r\cdot \mathrm{SNR_b}\right), \tag{4.9}$$

hence the specific values of E_b and σ are irrelevant for the channel law.

In order to evaluate the performance of a specific code/decoder pair, it is common to state the frame-error rate not only for a single SNR, but instead to plot (4.4) for a whole range of SNR values (see e.g. Figure 7.3 on page 96). Since in the majority of cases the FER cannot be determined analytically, these *performance curves* are usually obtained by Monte Carlo simulation. To that end, (4.9) is utilized to generate a large number of channel outputs, until a sufficient number of decoding errors ($\textsc{decode}(y) \neq x$) allows for a statistically significant estimation of (4.4).

4.4 Binary Linear Block Codes

4.4 Definition (linear code): A binary (n, k) code $\mathscr{C}$ is called *linear* if $\mathscr{C}$ is a linear subspace of $\mathbb{F}_2^n$. Consequently, a linear code admits a linear encoding function. ◁

Linear codes constitute the by far most important class of codes that are studied in literature. This is justified by the fact that, for binary-input symmetric memoryless channels, the results of Theorem 4.3 continue to hold when restricting to linear codes only [13, Ch. 6.2].

Linearity implies a vast amount of structure and allows codes to be compactly defined by matrices, as introduced below. Note that all operations on binary vectors in this section are performed in $\mathbb{F}_2$, i.e., "modulo 2".

4.5 Definition (dual code and parity-check matrices): The orthogonal complement

$$\mathscr{C}^\perp = \left\{\xi \in \mathbb{F}_2^n \colon \xi^T x = 0 \text{ for all } x \in \mathscr{C}\right\}$$

of a linear (n,k) code $\mathscr{C}$ is called the *dual code of* $\mathscr{C}$, the elements of $\mathscr{C}^\perp$ are *dual codewords* of $\mathscr{C}$.

A matrix $H \in \mathbb{F}_2^{m\times n}$ is a *parity-check matrix for* $\mathscr{C}$ if its rows generate $\mathscr{C}^\perp$ (i.e., the rows contain a basis of $\mathscr{C}^\perp$) and hence the equation

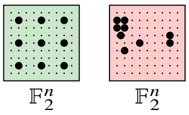
$$\mathscr{C} = \{x \colon Hx = 0\} \tag{4.10}$$

completely characterizes $\mathscr{C}$. In practice, $\mathscr{C}$ is often defined by stating a parity-check matrix H in the first place; in that event, we also speak of H as *the* parity-check matrix of $\mathscr{C}$. ◁

Since $\mathscr{C}^\perp$ is an $(n, n-k)$ code, it follows that $m > n - k$ in the above definition; in practice, $m = n - k$ is usually the case. Because $\mathscr{C}$ is linear, a linear *encoding* function can likewise be defined by means of a so-called *generator matrix* G, the rows of which form a basis of $\mathscr{C}$. Within the scope of this text, however, parity-check matrices are by far more important.

4.4.1 Minimum Hamming Distance

It is intuitively clear that, for a code to be robust against channel noise, the codewords should be maximally "distinguishable" from each other, i.e., there should be enough space between any two of them. Hence, one of the most important measures for the quality of a single code is its minimum distance, as defined below.



intuition of a good code with evenly spread codewords and a bad code

4.6 Definition (minimum distance): The *Hamming weight* $w_{\mathrm{H}}(x)$ of a binary vector x is defined to be the number of 1s among x. The *Hamming distance* $d_{\mathrm{H}}(x,y) = w_{\mathrm{H}}(x-y)$ of two vectors x, y of equal length is the number of positions in which they differ. The *minimum (Hamming) distance* of a linear code $\mathscr{C}$ is defined as

$$d_{\min}(\mathscr{C}) = \min_{\substack{x,y\in\mathscr{C}\\ x\neq y}} d_{\mathrm{H}}(x,y) = \min_{\substack{x,y\in\mathscr{C}\\ x\neq y}} \left|\{i \colon x_i \neq y_i\}\right|;$$

it is equivalent to the minimum Hamming weight among all non-zero codewords,

$$d_{\min}(\mathscr{C}) = \min_{x\in\mathscr{C}\setminus\{0\}} w_{\mathrm{H}}(x), \tag{4.11}$$

by linearity. ◁

The problem of finding the minimum distance of general linear codes is an NP-hard problem [18]. Nevertheless, integer programming techniques allow to compute $d_{\min}$ for codes which are not too large; this topic occurs in Papers III, VI and VII. In Section 5.4, with the *pseudoweight* we will encounter a similar weight measure that is specific to the LP decoding algorithm.

4.4.2 Factor Graphs

Let $\mathscr{C}$ be a binary linear code and $H \in \mathbb{F}_2^{m\times n}$ a parity-check matrix for $\mathscr{C}$. As noted before in (4.10), the condition

$$Hx = 0 \tag{4.12}$$

is necessary and sufficient for x being a codeword of $\mathscr{C}$. A *row-wise* viewpoint of (4.12) leads to the following definition.

4.7 Definition: Let $\mathscr{C}$ be a linear (n,k) code defined by the $m \times n$ parity-check matrix H. Any code $\mathscr{C}'$ such that $\mathscr{C} \subseteq \mathscr{C}'$ is called a *supercode* of $\mathscr{C}$. For $j \in \{1,\dots,m\}$, the particular supercode

$$\mathscr{C}_j = \{x\colon H_{j,\bullet}x = 0\}$$

is called the *j-th parity-check of $\mathscr{C}$*. It is a so-called *single parity check (SPC)* code, simply placing a parity condition on the entries $\{x_i\colon H_{j,i} = 1\}$. ◁

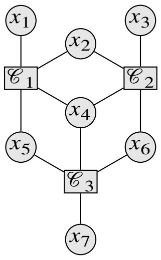
An obvious yet important consequence of the above definition is that

$$\mathscr{C} = \bigcap_j \mathscr{C}_j, \tag{4.13}$$

i.e., a linear code $\mathscr{C}$ is the intersection of the supercodes defined by the rows of a parity-check matrix for $\mathscr{C}$.

The fact that a linear code is characterized by several parity-check conditions placed on subsets of the variables is neatly visualized by a factor graph (or Tanner graph).

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



parity-check matrix and corresponding factor graph of a $(7,4)$ code

4.8 Definition (factor graph): The *factor graph* representing a parity-check matrix $H \subseteq \mathbb{F}_2^{m\times n}$ of a linear code $\mathscr{C}$ is a bipartite undirected graph $G = (V \dot\cup C, E)$ that has *m check nodes* $C = \{\mathscr{C}_1,\dots,\mathscr{C}_m\}$, *$n$ variable nodes* $V = \{x_1,\dots,x_n\}$ and an edge $(\mathscr{C}_j, x_i)$ whenever $H_{j,i} = 1$. ◁

The factor graph representation plays an important role in the analysis and design of codes and decoding algorithms. One of today's most prominent decoding methods, named belief propagation, works by iteratively exchanging messages (representing momentary beliefs or probabilities) between the check and variable nodes, respectively, of the factor graph [19]. Moreover, *graph covers* of the factor graph, as defined below in Section 5.4.3 and used extensively in Paper VII have become an important tool to analyze LP decoding, belief propagation decoding, and their mutual relation [20].

4.5 Convolutional and Turbo Codes

Turbo codes constitute an important class of linear codes, as they were the first to closely approach the promises of Theorem 4.3 using a very efficient decoding algorithm [2]. They are constructed by combining two or more (terminated) *convolutional codes.* For the matter of

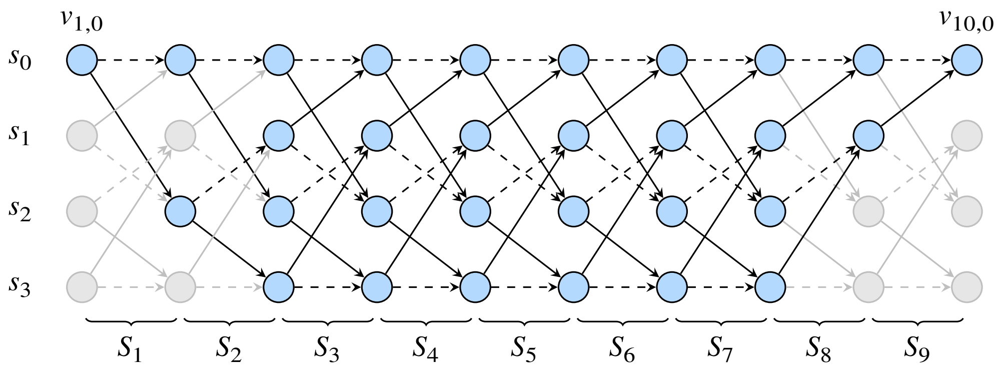


Figure 4.2: An example trellis graph with $k = 9$ segments and $2^d = 4$ states. Dashed edges have input bit $\text{in}(e) = 0$, for solid edges the input is $\text{in}(e) = 1$. Hence, for example, the zero input sequence $u = (0, \dots, 0)$ corresponds to the horizontal path $(v_{1,0}, v_{2,0}, \dots, v_{10,0})$ in T.

this work, it is important that the codewords of a convolutional code are in correspondence to certain *paths in a graph*, as described below.

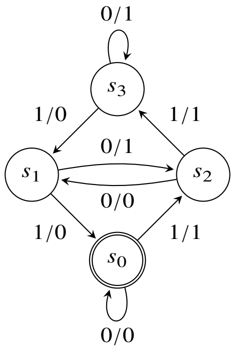
A terminated convolutional (n, k) code $\mathscr{C}$ with rate r (where $1/r = n/k \in \mathbb{N}$) and *memory* $d \in \mathbb{N}$ can be compactly described by a finite state machine (FSM) with 2^d states $S = \{s_0, \dots, s_{2^d-1}\}$ and a state transition function $\delta \colon S \times \mathbb{F}_2 \to S \times \mathbb{F}_2^{1/r}$ that defines the encoding of an information word $u \in \mathbb{F}_2^k$ as follows. Initially, the FSM is in state $s^{(1)} = s_0$. Then the bits u_i of u are subsequently fed into the FSM to determine the codeword $\mathscr{C}(u)$, i.e., in each step $i \in \mathbb{N}$, the current state $s^{(i)} \in S$ together with the i-th input bit u_i determines via

$$\delta(s^{(i)}, u_i) = (s^{(i+1)}, x^{(i)})$$

the subsequent state $s^{(i+1)}$ as well as n/k output bits $x^{(i)} = (x_1^{(i)}, \dots, x_{n/k}^{(i)})$ that constitute the part of the codeword that belongs to u_i. Finally, the machine has to terminate in the zero state, i.e., $s^{(k+1)} = s_0$ is required (this entails that some of the input bits are not free to choose and thus have to be considered as part of the *output* instead; in favor of a clear presentation, however, we ignore this inexactness and assume that u is in advance chosen such that $s^{(k+1)} = s_0$; see e.g. Paper IV for a more rigorous construction). The encoded word $x = \mathscr{C}(u)$ now consists of a concatenation of the $x^{(i)}$, namely,

$$x = \left(x_1^{(1)}, \dots, x_{n/k}^{(1)}, \dots, x_1^{(k)}, \dots, x_{n/k}^{(k)}\right).$$

The FSM of a convolutional code is always defined in such a way that this encoding is a *linear* map, and hence $\mathscr{C}$ is a linear code.



FSM of a rate-1 convolutional code: the edge labels u/x give the respective input (u) and output (x) bits

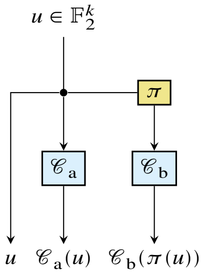
By "unfolding" the FSM along the time domain, we now associate a directed acyclic graph $T = (V, E)$ to the convolutional code $\mathscr{C}$, called its *trellis* (see Figure 4.2). Each vertex of T corresponds to a state of the FSM at a specific time step, such that $V = \{1, \dots, k + 1\} \times S$, where we denote the vertex (i, s) corresponding to state $s \in S$ at step i shortly by $v_{i,s} \in V$.

The edges of T in turn correspond to valid state transitions. For each $i \in \{1, \dots, k\}$ and $s \in S$, there are two edges emerging from $v_{i,s}$, according to the two possible values of u_i (which are encoded in the *input labels* $\mathrm{in}(e) \in \{0,1\}$ of the edges); both their *output labels* $\mathrm{out}(e) \in \mathbb{F}_2^{n/k}$ and target vertices $v_{i+1,s'}$ are determined by the state transition function via

$$\delta(s, \mathrm{in}(e)) = (s', \mathrm{out}(e)).$$

Hence, each edge $e = (v_{i,s}, v_{i+1,s'})$ of T corresponds to the input of one bit at a specific step i and a specific state s of the encoder FSM that is in state s' afterwards; the labels of e define the value of the input bit $u_i = \mathrm{in}(e)$ and the output sequence $x^{(i)} = \mathrm{out}(e)$, respectively. The trellis T is thus "$(k+1)$-partite" in the sense that V partitions into $k+1$ subsets V_i such that edges only exist between two subsequent sets V_i and V_{i+1}. This motivates the definition of the i-th trellis *segment* $S_i = (V_i \cup V_{i+1}, E_i)$ according to the i-th encoding step as the subgraph induced by $V_i \cup V_{i+1}$.

The transition function δ is always designed in such a way that if $\delta(s,0) = \delta(s',x')$ and $\delta(s,1) = \delta(s'',x'')$ then $x' \neq x''$, i.e., at each encoding step, the two outputs corresponding to an input bit 0 and 1, respectively, must be different. As a consequence, the codewords of $\mathscr{C}$ are in one-to-one correspondence with the paths from $v_{1,0}$ to $v_{k+1,0}$ in T: at step i in state s, the next n/k bits of the codeword determine which edge to follow from $v_{i,s}$, while conversely the output label of such an edge fixes the next n/k code bits. Due to the boundary constraints $s^{(1)} = s^{(k+1)} = s_0$, some vertices and edges in the leading as well as the endmost d segments are not part of any such path and therefore are usually removed from T, as shown in the figure.



encoding scheme of a turbo code

In a turbo code $\mathscr{C}_\mathrm{TC}$, now, several convolutional codes are *concatenated* in order to improve upon the rather weak error-correction performance of plain convolutional codes. In the most common form, two identical convolutional codes $\mathscr{C}_\mathrm{a}$ and $\mathscr{C}_\mathrm{b}$ with rate $r = 1$ each are concatenated *parallely*, which means that the information word u is encoded by both, but the entries of u are permuted by a fixed permutation (the *interleaver*) $\pi \in \mathbb{S}_k$ before entering the second component code $\mathscr{C}_\mathrm{b}$. A codeword of the turbo code $\mathscr{C}_\mathrm{TC}$ then consists of the concatenation of a copy of u, $\mathscr{C}_\mathrm{a}(u)$ and $\mathscr{C}_\mathrm{b}(\pi(u))$, so that the overall rate of $\mathscr{C}_\mathrm{TC}$ is $r = 1/3$ (here, again, a small rate loss due to termination is ignored). In a more general setting, the term *turbo-like codes* refers to schemes that include *serial* concatenation, where the output of one convolutional code is used as input for another convolutional code, or any combination of parallel and serial concatenations—an example are 3-D turbo codes which are studied in Paper VII.

Taking the path representation of codewords of $\mathscr{C}_\mathrm{a}$ and $\mathscr{C}_\mathrm{b}$ in their respective trellis graphs T_a and T_b into account, from the above definition of a turbo code $\mathscr{C}_\mathrm{TC}$ we can derive a one-to-one correspondence between codewords of $\mathscr{C}_\mathrm{TC}$ and pairs $(P_\mathrm{a} = (e_1^\mathrm{a}, \dots, e_k^\mathrm{a}), P_\mathrm{b} = (e_1^\mathrm{b}, \dots, e_k^\mathrm{b}))$ of paths in T_a and T_b, respectively, which additionally fulfill that

$$\mathrm{in}(e_i^\mathrm{a}) = \mathrm{in}(e_{\pi(i)}^\mathrm{b}), \tag{4.14}$$

i.e., the i-th edge in P_a must have the same input label as the $\pi(i)$-th edge in P_b, because both equal the i-th input bit u_i. The application of this path–code relationship to decoding by mathematical optimization is introduced in Section 5.5, as it is the basis for the contributions presented in Papers IV and VII.

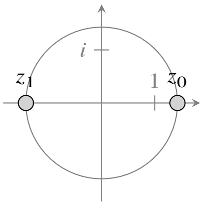
4.6 Non-Binary Codes and Higher-Order Modulation

So far, we assumed *binary* data processing throughout coding and data transmission, as introduced in Section 4.1. While today's microelectronic systems, as is generally known, *internally* rely on the binary representation of data, the above simplification is, in two different but related ways, not the whole story in case of channel coding.

First, observe that the definition of (linear) codes can straightforwardly be generalized to any finite field $\mathbb{F}_q$ for a prime power q: the information and codewords still lie in vector spaces and linear maps can be defined as usual; the parity-check matrix of such a *non-binary code* then has entries in $\{0, \dots, q-1\}$. Several constructions of strong codes rely on a non-binary field, thus a restriction to the binary case would prevent us from using those codes.

Secondly, in many practical transmission systems the signal space is modeled by the *complex* plane, where the real and imaginary axis, respectively, correspond to two different carrier waves (e.g., two sines that are out of phase by $\pi/2$) such that any complex number z represents a linear combination of both waves that is then emitted onto the carrier medium. This technique is called *modulation*. At the receiver's side, a complementary *demodulator* measures the potentially distorted wave and reconstructs (e.g. via Fourier analysis) a point $\tilde{z}$ in the complex plane.

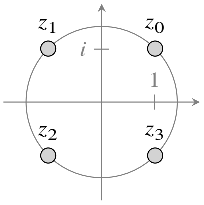
In the most simple *binary* case, two complex numbers (e.g. $z_0 = 1 + 0i$ and $z_1 = -1 + 0i$) are chosen that represent the values 0 and 1, respectively, of a single bit. If we now assume that the channel adds independent Gaussian noise to both carrier waves, this case reduces to the binary AWGN channel as introduced in Section 4.3.



binary modulation

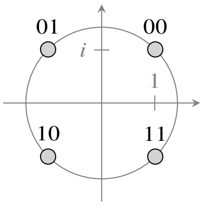
In *higher-order modulation*, however, more than one bit of information is transmitted at once by choosing $Q > 2$ (usually, Q is a power of 2) possible complex signals $\{z_0, \dots, z_{Q-1}\}$. Hence, the channel can be modeled by Q probability functions $P(\tilde{z} \mid 0), \dots, P(\tilde{z} \mid Q-1)$ where $\tilde{z} \in \mathbb{C}$.

Non-binary codes and higher-order modulation can be combined in several ways. Most obviously, if $q = Q$ then to each complex symbol $z_0, \dots, z_{Q-1}$ an element of $\mathbb{F}_q$ can be assigned, such that the channel transmits one entry of the codeword per signal. On the other hand, if e.g. $q = 2$ and $Q = 2^k$, k bits of the codeword can be sent *at once* by mapping the 2^k possible bit configurations to the 2^k chosen complex symbols.



modulation with $Q = 4$ signals

In both cases, the expression for ML decoding (4.6) is more complex than in the binary case. In particular, the linearization to formulate ML decoding as an IP (Section 5.1) is not possible in the same way because there is no individual LLR value corresponding to each channel signal. Nevertheless, ML decoding of non-binary codes was formulated as an IP in [21], and in Paper II we furthermore show how to incorporate higher-order modulation into the IP model.



example mapping of $k = 2$ bits to $4 = 2^k$ complex symbols

Chapter 5

Optimization Into Coding: The Connection

So far, the two subjects introduced above—linear and integer optimization in Chapter 3 on the one hand and coding theory in Chapter 4 on the other—may seem to have little in common: while the first consists of a mixture of (linear) algebra and probability, the latter is concerned with solution algorithms for specific linear or discrete problems. The two areas become linked, however, by the observation that *de*coding a received signal, in particular the (optimal) ML decoding as introduced in Section 4.2, amounts to solving a combinatorial optimization problem that can be formulated as an IP.

Therefore, this section introduces the abovementioned connection by reviewing the IP formulation of ML decoding and is then mainly concerned with a particular LP relaxation of that formulation, called *LP decoding*. The style of writing is intentionally a little more verbose than it was in the two previous chapters because, first, this part is the most probable for the audience to be unfamiliar with and, secondly, due to the recency of the subject, we are not aware of any up-to-date, tutorial-like, yet mathematically stringent document that covers what we believe to be its most important aspects.

A well-written resource for LP decoding is the dissertation of its inventor Jon Feldman [4]. Large parts of Section 5.4 are elaborately presented in [20] which is abounding in examples. Not least, Paper I includes a literature survey of the algorithmic aspects of optimization-based decoding until the time of its writing, as well as a thorough yet short coverage of the underlying theory.

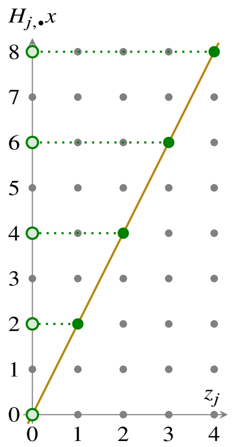
5.1 ML Decoding as Integer Program

While the perception of ML decoding (at least on the BSC) as a combinatorial optimization problem is probably as old as coding theory itself (for example, the proof of its NP-hardness by Berlekamp, McEliece, and Tilborg in 1978 [17] constitutes an obvious connection to the optimization community), it has been only in 1998 that an *integer programming* formulation of the problem was given [22], which consists of linearizations (with respect to $\mathbb{R}$) of both the objective function and the code structure. In the following, we present a slightly modified version (as in [23]) of that construction.

Recall that the ML codeword maximizes the likelihood function $P(y \mid x)$ for a received channel output y (4.6). Since the channel is assumed to be memoryless, we have [4, 22]

$$\begin{aligned} \hat{x}_{\text{ML}} &= \arg\max_{x \in \mathscr{C}} \prod_{i=1}^{n} P(y_i \mid x_i) && \text{(5.1a)} \\ &= \arg\min_{x \in \mathscr{C}} -\sum_{i=1}^{n} \ln P(y_i \mid x_i) && \text{(5.1b)} \\ &= \arg\min_{x \in \mathscr{C}} \sum_{i=1}^{n} (\ln P(y_i \mid 0) - \ln P(y_i \mid x_i)) && \text{(5.1c)} \\ &= \arg\min_{x \in \mathscr{C}} \sum_{i:\, x_i = 1} \ln\left(\frac{P(y_i \mid 0)}{P(y_i \mid 1)}\right) && \text{(5.1d)} \end{aligned}$$

Since the fraction in the last term exactly matches the LLR value λ_i (4.2) which is known to the observer, we see that ML decoding is equivalent to minimizing the linear functional $\lambda^T x$ over all codewords $x \in \mathscr{C}$.



feasible integer points of the linearization $H_{j,\bullet}x - 2z_j = 0$

How can we grasp this condition "$x \in \mathscr{C}$" by an IP? The answer lies in the code-defining equation $Hx = 0$ (4.12) for a given parity-check matrix $H \in \mathbb{F}_2^{m \times n}$, which can be $\mathbb{R}$-linearized in virtue of *auxiliary* integer variables $z \in \mathbb{Z}^m$ as follows: The condition $x \in \mathscr{C}$ is eqivalent to $Hx = 0 \pmod 2$, which in turn is fulfilled if and only if the result of Hx, as an operation in the reals, is a vector whose entries are even numbers. It is thus clear that the formulation

$$\begin{aligned} \min \quad & \lambda^T x && \text{(5.2a)} \\ \text{s.t.} \quad & Hx - 2z = 0 && \text{(5.2b)} \\ & x \in \mathbb{F}_2^n,\ z \in \mathbb{Z}^m && \text{(5.2c)} \end{aligned}$$

models ML decoding because (5.2b) can be achieved by an *integral* vector z if and only if Hx is even.

Note that any IP formulation of the ML decoding problem can be easily modified to output the minimum distance $d_{\min}$ of a code: in view of (4.11), this is equivalent to determine a codeword of minimum Hamming weight. By setting $\lambda = (1, \dots, 1)$, the objective function value (5.2a) equals the Hamming weigth of x, and an additional linear constraint $\sum x_i \geq 1$ excludes the all-zero codeword, such that the IP solution must be a codeword of minimum Hamming weight ([24, 25], see also Papers VI and VII).

Interestingly, the above linearization of the code was apparently "forgotten" and several years later reinvented in 2009 [26] and 2010 [25]. One possible explanation might be that while (5.2) is very compact in terms of size, its LP relaxation is essentially useless: if (5.2c) is replaced by its continuous counterpart, then the feasible region of the x variables is the entire unit hypercube—for any configuration of x, a corresponding real z can be found such that (5.2b) is fulfilled. It should be noted, however, that the formulation (5.2) has found recent justification by the fact that it appears to perform very well with commercial IP solvers [24, 27].

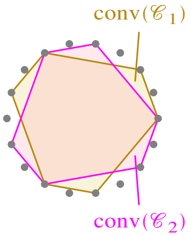
5.2 LP Decoding

It was the *LP decoder* introduced by Feldman [4, 5] that established linear programming in the field of decoding by providing several equivalent IP formulations for which even the LP relaxations exhibit a decoding performance that is of interest for practical considerations.

The essence of Feldman's LP decoder lies in the representation (4.13) of a code, together with the fact that for the SPC codes $\mathscr{C}_j$ (Definition 4.7), a polynomially sized description of $\operatorname{conv}(\mathscr{C}_j)$ by means of (in)equalities and potential auxiliary variables is possible. Instead of providing an LP description of $\operatorname{conv}(\mathscr{C})$ (which in view of the NP-hardness of ML decoding is unlikely to be tractable), the LP decoder thus operates on the relaxation polytope

$$\mathscr{P}(H) = \bigcap_j \operatorname{conv}(\mathscr{C}_j) \supseteq \operatorname{conv}(\mathscr{C}), \tag{5.3}$$

called the *fundamental polytope* [20] of the parity-check matrix H. The vertices of $\mathscr{P}(H)$ are also called *pseudocodewords*.

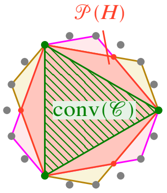


two constitutent polytopes conv($\mathscr{C}_1$) and conv($\mathscr{C}_2$) (the circular dots represent $\mathbb{F}_2^n$)

5.1 Definition (LP decoder): Let H be a parity-check matrix for the linear code $\mathscr{C}$ and λ the vector of channel LLR values. The *LP decoder* LP-DECODE(λ) outputs, for given $\lambda \in \mathbb{R}^n$, the optimal solution $\hat{x}$ of the LP

$$\begin{aligned} \min \quad & \lambda^T x && (5.4a)\\ \text{s.t.} \quad & x \in \mathscr{P}(H), && (5.4b) \end{aligned}$$

where $\mathscr{P}(H)$ is as defined in (5.3). ◁



$\mathscr{P}(H)$ = conv($\mathscr{C}_1$) ∩ conv($\mathscr{C}_2$) is a superset of conv($\mathscr{C}$) with additional fractional pseudocodewords $\notin \mathbb{F}_2^n$

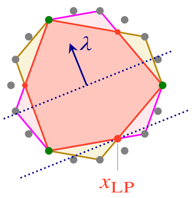
The above definition is a meaningful relaxation of $\operatorname{conv}(\mathscr{C})$ because one can easily show that $\mathscr{P}(H) \cap \{0,1\}^n = \mathscr{C}$, i.e., the codewords of $\mathscr{C}$ and the integral vertices of $\mathscr{P}(H)$ coincide, which proves the following theorem.

5.2 Theorem (ML certificate [4]): *The LP decoder has the* ML certificate property*:*

$$\hat{x} = \text{LP-DECODE}(\lambda) \in \{0,1\}^n \Rightarrow \hat{x} = x_{\text{ML}},$$

i.e., if $\hat{x}$ is integral, it must be the ML codeword. Put another way, solving (5.4) *as an* IP *with the additional constraint $x \in \{0,1\}^n$ constitutes a true ML decoder.* ◁

Note that if we had $\mathscr{P}(H) = \operatorname{conv}(\mathscr{C})$, the LP decoder would actually be an ML decoder. Because this is not the case in general (moreover, it apparently does not hold for any interesting code; see [28]), the inclusion $\operatorname{conv}(\mathscr{C}) \subseteq \mathscr{P}(H)$ is usually strict, and the difference $\mathscr{P}(H) \setminus \operatorname{conv}(\mathscr{C})$ must be due to additional *fractional* vertices of $\mathscr{P}(H)$, i.e., vectors for which at least one entry is neither 0 nor 1.



LLR vector λ for which $x_{\text{LP}} \notin \mathbb{F}_2^n$ is optimal

Feldman gave three different formulations of $\operatorname{conv}(\mathscr{C}_j)$, the convex hull of the SPC codes constituting $\mathscr{C}$, to be used in (5.4b). In the context of this work, only the one described below, which is named Ω in [4] and based on [29], is relevant.

5.3 Theorem: *Let H and $\mathscr{C}$ as above and let $N_j = \{i\colon H_{j,i} = 1\}$ be the indices covered by the j-th parity check $\mathscr{C}_j$ of $\mathscr{C}$. Then the inequalities*

$$\sum_{i\in S} x_i - \sum_{i\in N_j\setminus S} x_i \leq |S| - 1 \qquad \text{for all } S \subseteq \mathcal{N}_j \text{ with } |S| \text{ odd} \tag{5.5a}$$

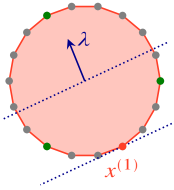
$$0 \leq x_i \leq 1 \qquad \text{for } i = 1, \dots, n \tag{5.5b}$$

precisely define the convex hull of $\mathscr{C}_j$. ◁

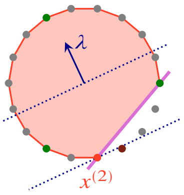
As each inequality (5.5a) explicitly forbids one odd-sized set S, i.e., a configuration for which $H_{j,\bullet}x \equiv 1 \pmod 2$ (it is violated by a binary vector x if and only if $x_i = 1$ for $i \in S$ and $x_i = 0$ for $i \in N_j \setminus S$), they are also called *forbidden-set inequalities*. Note that the number of such inequalities is exponential in the size of N_j, which is why LP decoding was first proposed for codes defined by a *sparse* matrix H, so-called *LDPC* codes [1, 30]. It will however turn out in the following review of adaptive LP decoding that the inequalities (5.5a) can be efficiently *separated*, which renders their exponential quantity harmless in practice.

5.3 Adaptive LP Decoding

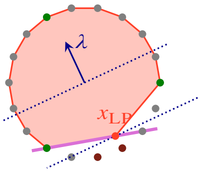
adaptive LP decoding of the previous example:



initial optimization over $[0,1]^n$



cut for $x^{(1)}$ leads to next solution $x^{(2)}$



another cut yields the same solution x_{LP} as before

The prohibitive size of the LP decoding formulation (5.4), especially for dense H and larger block lengths, can be overcome by a cutting plane algorithm (cf. Section 3.3.1), called *adaptive LP decoding*, as proposed in [31, 32]. It starts with the trivial problem of minimizing the objective function over the unit hypercube:

$$\begin{aligned} \min \quad & \lambda^T x \\ \text{s.t.} \quad & x \in [0,1]^n \end{aligned}$$

and then iteratively *refines* the domain of optimization by inserting those forbidden-set inequalities (5.5a) that are *violated* by the current solution, and hence constitute valid *cuts*. The procedure to find a cut in the j-th row of H (it is shown in [31] that, at any time, one row of H can provide at most one cut) is based on the following reformulation of (5.5a):

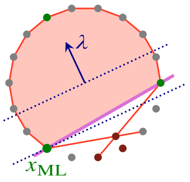
$$\sum_{i\in S}(1 - x_i) + \sum_{i\in N_j\setminus S} x_i \geq 1. \tag{5.6}$$

To find a violating inequality (if it exists) of the form (5.6), an odd-sized set S needs to be found that minimizes the left-hand side of (5.6). It is easy to show [32] that this can be accomplished by taking all i with $x_i > 1/2$ and, if that set is even-sized, remove or add the index i^* for which x_{i^*} is closest to $1/2$.

When no more violating inequalities are found, the solution equals that of (5.4) and the algorithm terminates. The total number of inequalities in the final model is however bounded by n^2, which shows that the adaptive approach indeed overcomes, with respect to size, the problems of the model (5.5).

An important advantage of the separation approach is that one can immediately incorporate *additional* types of cutting planes—if it is known how to solve the corresponding separation problem, i.e., find violated cuts from the current LP solution—in order to tighten the LP relaxation (5.4). A successful method of doing so is by using redundant parity-checks.

5.4 Definition: Let $\mathscr{C}$ be a linear code defined by a parity-check matrix H. A dual codeword $\xi \in \mathscr{C}^\perp$ that does not appear as a row of H is called a *redundant parity-check (RPC)*. An RPC ξ is said to *induce a cut* at the current LP solution x if one of the inequalities (5.5a) derived from ξ is violated by x. ◁



an example RPC cut (in the picture, the cut is a facet of $\operatorname{conv}(\mathscr{C})$) that would lead from the above LP solution to the ML codeword x_{ML}

RPCs are called "redundant" because the rows of H already contain a basis of $\mathscr{C}^\perp$ by definition, thus every RPC must be the (modulo-2) sum of two or more rows of H. The following result [23] gives a strong clue which RPCs might potentially induce cuts.

5.5 Lemma: *Let $\xi \in \mathscr{C}^\perp$ be a dual codeword and x an intermediate solution of the adaptive LP decoding algorithm. If*

$$|\{i\colon \xi_i = 1 \text{ and } x_i \notin \{0,1\}\}| = 1,$$

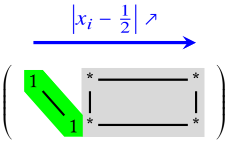
i.e., exactly one index of the fractional part of x is contained in the support of ξ, then ξ induces an RPC cut for x. ◁

An efficient method to search for RPC cuts in view of the above observation works as follows [23, 33]: Given an intermediate LP solution x,

(1) sort the columns of H according to an ascending ordering of $|x_i - 1/2|$,

(2) perform Gaussian elimination on the reordered H to diagonalize its leftmost part, resulting in an alternative parity-check matrix $\tilde{H}$, then

(3) search for cuts among the rows of $\tilde{H}$ as in adaptive LP decoding.



structure of $\bar{H}$: diagonalized part at the left after reordering of columns by $\left|x_i - \frac{1}{2}\right|$

The motivation behind this approach is that, if the submatrix of H corresponding to the fractional part of x has full column rank, the leftmost part of $\tilde{H}$ will be a diagonal matrix, and hence by Lemma 5.5 *every row* of $\tilde{H}$ would induce a cut for x. The results reported in [23] and [33] furthermore suggest that, even if this is not the case and thus the requirements of Lemma 5.5 are not necessarily met, this "sort-&-diagonalize" strategy very often leads to cuts and substantially improves the error-correcting capability of the plain LP decoder that does not involve RPCs.

5.4 Analysis of LP Decoding

While the aspects of LP decoding discussed so far include some useful theoretical results about an *individual* run of the algorithm (most importantly, the ML certificate property given in Theorem 5.2), there is no immediate theoretical approach to determine the *average* error-correction performance (4.4) of a given code and channel under LP decoding, other than using simulations as described in Section 4.3.

In the following, we briefly outline an approach to a theoretical performance analysis of LP decoding that is based on a channel-specific rating of the vertices of the LP decoding polytope, called *pseudoweight.* The theory presented in this section is based on the "plain" LP decoder as defined in Definition 5.1, i.e., does not take the improvement via RPC cuts (Definition 5.4 and the discussion thereafter) into account; most of the results can however be extended to that case in a straightforward manner.

5.4.1 All-Zero Decoding and the Pseudoweight

First we introduce the very useful *all-zeros assumption.*

5.6 Theorem (Feldman [4]): *If the LP decoder* (5.4) *is used on a binary-input memoryless symmetric channel, the probability of decoding error is independent of the sent codeword: the FER* (4.4) *satisfies*

$$\mathrm{FER} = P(\text{LP-}\textsc{decode}(\lambda) \neq 0 \mid 0 \textit{ was sent}).$$

◁

The proof of Theorem 5.6 relies on the symmetry of both the channel and the polytope. The latter is due to the linearity of the code (which implies that $\operatorname{conv}(\mathscr{C})$ basically "looks the same" from any codeword x) and the $\mathscr{C}$-symmetry [4, Ch. 4.4] of $\mathscr{P}(H)$, which extends that symmetry to the relaxed LP polytope. As a consequence of the theorem, when examining the LP decoder's error probability we can always assume that the all-zero codeword $0 \in \mathscr{C}$ was sent, which greatly simplifies analysis.

Assume now that the all-zero codeword is sent through a channel and the result λ is decoded by the LP decoder which solves (5.4) to obtain the optimal solution $\hat{x}$. The decoder fails if there is a vertex x of $\mathscr{P}(H)$ such that

$$\lambda^T x < \lambda^T 0 = 0 \tag{5.7}$$

(we assume here and in the following that in case of ties, i.e., $\lambda^T x = 0$ for some non-zero vertex x, the LP decoder correctly outputs 0; for the AWGN channel, ties can be neglected since they occur with probability 0). The probability $P(\lambda^T x < 0)$ of the event (5.7), also called the *pairwise error probability* between x and 0, depends on the channel. In case of the AWGN channel, by (4.9) we have

$$\lambda_i x_i \sim \mathcal{N}\left(4r \cdot \mathrm{SNR_b} x_i, 8r \cdot \mathrm{SNR_b} x_i^2\right)$$

and because the channel treats symbols independently and furthermore the sum of independent Gaussian variables is again gaussian with mean and variance simply summing up, we obtain

$$\lambda^T x \sim \mathcal{N}\left(4r \cdot \mathrm{SNR_b} \|x\|_1, 8r \cdot \mathrm{SNR_b} \|x\|_2^2\right). \tag{5.8}$$

Hence, $\lambda^T x$ is again Gaussian, and the probability that $\lambda^T x < 0$ computes as (using the abbreviations $\mu = 4r \cdot \mathrm{SNR_b}\, \|x\|_1$ and $\sigma^2 = 8r \cdot \mathrm{SNR_b}\, \|x\|_2^2$)

$$P(\lambda^T x < 0) = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{0} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \mathrm{d}x = \frac{1}{\sqrt{2\pi}} \int_{\frac{\mu}{\sigma}}^{\infty} e^{-\frac{1}{2}x^2} \mathrm{d}x.$$

Introducing the Q-function as $Q(a) = \int_a^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \mathrm{d}x$, we get

$$P(\lambda^T x < 0) = Q\left(\frac{\mu}{\sigma}\right) = Q\left(\sqrt{2r \cdot \mathrm{SNR_b} \frac{\|x\|_1^2}{\|x\|_2^2}}\right) = F\left(\frac{\|x\|_1^2}{\|x\|_2^2}\right),$$

for a monotone function F, which motivates the following definition.

5.7 Definition ([20, 34]): Let x be a non-zero vertex of $\mathcal{P}(H)$. The *(AWGN) pseudoweight* of x is defined as

$$w_{\mathrm{p}}^{\mathrm{AWGN}}(x) = \|x\|_1^2 \big/ \|x\|_2^2 . \tag{5.9}$$

◁

Observe that the pairwise error probability $P(\lambda^T x < 0)$ is a strictly monotonically decreasing function of $w_{\mathrm{p}}^{\mathrm{AWGN}}(x)$: the lower the pseudoweight is, the higher is the probability that the LP decoder wrongly runs into x instead of 0. The AWGN pseudoweight is thus a compact expression that measures the "danger" of decoding error due to a specific vertex $x \in \mathcal{P}(H)$.

5.4.2 The Fundamental Cone

One simple parameter for estimating the average performance of the LP decoder, for a given code $\mathscr{C}$ and a parity-check matrix H, is the *minimal pseudoweight* among the non-zero vertices of $\mathcal{P}(H)$,

$$w_{\mathrm{p,min}}^{\mathrm{AWGN}}(H) = \min\left\{w_{\mathrm{p}}^{\mathrm{AWGN}}(x) \colon x \neq 0 \text{ is a vertex of } \mathcal{P}(H)\right\},$$

which corresponds to the *most probable* non-zero vertex that accidentally becomes optimal instead of the all-zero one. Note that for an integral vertex x we have $w_{\mathrm{p}}^{\mathrm{AWGN}}(x) = w_{\mathrm{H}}(x)$ (cf. Definition 4.6), which shows that for an ML decoder the minimum Hamming weight takes on this role.

The minimum pseudoweight alone is however still a rather rough estimate of the decoding performance: both the *quantity* of minimum-pseudoweight vertices and the (quantities of the) next larger pseudoweights influence the error probability $P(\text{LP-DECODE}(\lambda) \neq 0)$. Therefore, the *pseudoweight enumerator* of $\mathcal{P}(H)$, i.e., a table containing all occuring pseudoweights of the non-zero vertices alongside with their frequencies, would allow for a better estimation of the decoding performance. Finally, for an *exact* computation of the error rate we would need a description of the region

$$\Lambda = \left\{\lambda \colon \lambda^T x \geq 0 \text{ for all } x \in \mathcal{P}(H)\right\} \tag{5.10}$$

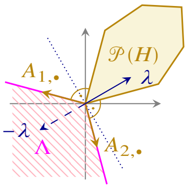
of channel outputs λ for which 0 is the optimal solution of (5.4), and then compute the probability $1 - P(\lambda \in \Lambda)$ by integrating the density function given by (5.8) over Λ. In optimization language, Λ is called the *dual cone* of $\mathscr{P}(H)$.

While the three tasks stated above appear to be ascendingly difficult—no efficient algorithm is known to compute the minimum pseudoweight in general—it turns out that Λ as defined in (5.10) can be determined by LP duality: assume that the LP decoder (5.4) is given in the form

$$\min \quad \lambda^T x \tag{5.11a}$$
$$\text{s.t.} \quad Ax \leq b, \tag{5.11b}$$

where A and b represent (5.5). The dual of (5.11) is

$$\max \quad -b^T y \tag{5.12a}$$
$$\text{s.t.} \quad A^T y = -\lambda \tag{5.12b}$$
$$y \geq 0. \tag{5.12c}$$



dual cone Λ of $\mathscr{P}(H)$ spanned by two rows $A_{1,\bullet}, A_{2,\bullet}$, and an example λ with $-\lambda \in \Lambda$

By Theorem 3.11, 0 is optimal for (5.11) if and only if there is an y that is feasible for (5.12) with $b^T y = 0$. Now 0 is feasible for (5.11), hence $b \geq 0$, which together with (5.12c) implies that $y_j = 0$ whenever $b_j \neq 0$ in a solution y with $b^T y = 0$. Taking a closer look at (5.12), we conclude

$$\lambda \in \Lambda \Leftrightarrow (-\lambda) \in \operatorname{conic}\left(\{A_{j,\bullet} \colon b_j = 0\}\right).$$

Note that Λ is also a polyhedron by Theorem 3.5.

As a by-product of the above calculations, it appears that the rows of $Ax \leq b$ for which $b_j \neq 0$ are irrelevant for Λ and hence for the question whether the decoder fails or not. It is easy to show that deleting those rows leads to $\operatorname{conic}(\mathscr{P}(H))$, which motivates the following definition.

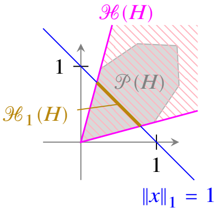
5.8 Definition: The conic hull of the fundamental polytope,

$$\mathscr{K}(H) = \operatorname{conic}(\mathscr{P}(H)),$$

is called the *fundamental cone* of H. By

$$\mathscr{K}_1(H) = \{x \in \mathscr{K}(H) \colon \|x\|_1 = 1\}$$

we denote the intersection of $\mathscr{K}(H)$ with the unit simplex. ◁



From the above discussion we can now formulate the following equivalent conditions for the LP decoder to succeed.

5.9 Corollary: *The following are equivalent:*

(1) *The LP decoder correctly decodes λ to 0.*

(2) $\lambda \in \Lambda$.

(3) *There is no $x \in \mathscr{K}(H)$ with $\lambda^T x < 0$.*

(4) *There is no* $x \in \mathscr{K}_1(H)$ *with* $\lambda^T x < 0$. ◁

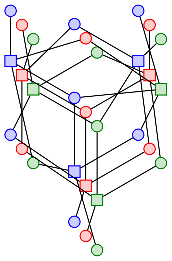
As a consequence, we can study either of the three sets $\mathscr{P}(H)$, $\mathscr{K}(H)$ or $\mathscr{K}_1(H)$ in order to characterize the LP decoder. Note that while the set $\mathscr{K}(H)$ is larger than $\mathscr{P}(H)$, its description complexity is much lower, because we need only as many forbidden-set inequalities (5.5a) as there are 1-entries in H. In addition, observe that the pseudoweight is invariant to scaling, i.e., $w_{\mathrm{p}}^{\mathrm{AWGN}}(\tau x) = w_{\mathrm{p}}^{\mathrm{AWGN}}(x)$ for $\tau > 0$. Consequently, the search for minimum pseudoweight can be restrained to either $\mathscr{K}(H)$ or $\mathscr{K}_1(H)$ as well. For the latter, it takes on the particularly simple form

$$w_{\mathrm{p,min}}^{\mathrm{AWGN}}(H) = \max\left\{\|x\|_2^2 : x \in \mathscr{K}_1(H)\right\}.$$

While the maximization of $\|\cdot\|_2^2$ over a polytope is NP-hard in general, the most effective algorithms to approach the minimum pseudoweight rely on the above formulation; see [35, 36] and Paper VII.

5.4.3 Graph Covers

There is a fascinating *combinatorial* characterization of the fundamental polytope $\mathscr{P}(H)$ derived from the factor graph G of a parity-check matrix H (cf. Definition 4.8). Central to it is the following definition.



a 3-cover of the (7, 4) code shown on page 30

5.10 Definition (graph cover): Let $G = (V \cup C, E)$ be the factor graph associated to a parity-check matrix H with variable nodes V, check nodes C, and edge set E. For $m \in \mathbb{N}$, an *m-cover* of G is a factor graph $\bar{G}$ with variable nodes $\bar{V} = V \times \{1, \dots, m\}$, check nodes $\bar{C} = C \times \{1, \dots, m\}$ and a set of $|E|$ permutations $\{e \in \mathbb{S}_M \colon e \in E\}$ such that the edge set of $\bar{G}$ is

$$\bar{E} = \left\{(\mathscr{C}_j^{(k)}, x_i^{(l)}) \colon (\mathscr{C}_j, x_i) = e \in E \text{ and } \pi_e(k) = l\right\},$$

where by $\mathscr{C}_j^{(k)} = (\mathscr{C}_j, k) \in \bar{C}$ we denote the k-th copy of $\mathscr{C}_j \in C$ (and $V_i^{(l)}$ analogously). ◁

Despite the somewhat heavy notation in the above definition, the idea of a graph cover is rather simple: make m identical copies of G and then, for every edge $e = (\mathscr{C}_j, x_i) \in E$, arbitrarily "rewire" the m copies of e in a one-to-one fashion between the copies of $\mathscr{C}_j$ and x_i.

Since every graph cover of a factor graph G defining a code $\mathscr{C}$ is a factor graph itself, it defines a code $\bar{\mathscr{C}} = \bar{\mathscr{C}}(\bar{G})$ that has m times the block length of $\mathscr{C}$. Let

$$\tilde{x} = (x_1^{(1)}, \dots, x_1^{(m)}, \dots, x_n^{(1)}, \dots, x_n^{(m)}) \in \bar{\mathscr{C}}$$

be such a codeword, where the entries are ordered in the obvious way. Then, the rational n-vector x defined by

$$x_i = \frac{1}{m}\sum_{k=1}^{m} x_i^{(k)}$$

is called the *scaled pseudocodeword* of $\mathscr{C}$ associated to $\tilde{x}$. Let $\mathcal{Q}(H)$ denote the union, over all $M > 0$ and all M-covers $\bar{G}$ of G, of the scaled pseudocodewords associated to all the codewords of $\bar{\mathscr{C}}(\bar{G})$. It then holds that

$$\mathcal{Q}(H) = \mathcal{P}(H) \cap \mathbb{Q}^n,$$

i.e., $\mathcal{Q}(H)$ contains exactly the rational points of $\mathcal{P}(H)$, and hence $\mathcal{P}(H) = \overline{\mathcal{Q}(H)}$.

Graph covers have been proposed in [20], among other things, to study the relationship between LP decoding and iterative methods, which can be shown to always compute solutions that are optimal for some graph cover of G. In Paper VII, we use graph covers to estimate the minimum pseudoweight of ensembles of codes.

5.5 LP Decoding of Turbo Codes

Since one can show that turbo(-like) codes, as introduced in Section 4.5, are special instances of linear block codes, one could compute, for a given turbo code $\mathscr{C}_{\mathrm{TC}}$, a parity-check matrix H defining $\mathscr{C}_{\mathrm{TC}}$, and apply all of the abovementioned theory and algorithms to decode them using linear and integer optimization. In doing so, however, one would neglect the immediate combinatorial structure embodied in $\mathscr{C}$ in virtue of the trellis graphs of the constituent convolutional codes.

In fact, for an individual convolutional code $\mathscr{C}$, it can be shown that ML decoding can be performed by computing a shortest path (due to the simple structure of T, this can be achieved in $O(k)$ time) in the trellis T of $\mathscr{C}$, after having introduced a cost value c_e to each trellis edge $e = (v_{i,s}, v_{i+1,s'}) \in E$: as every time step produces n/k output bits, e determines the entries $x^{(i)} = (x_{(i-1)n/k+1}, \dots, x_{in/k}) = x_I$, for an appropriate index set I, of the codeword x. In view of (5.1), we thus have to define

$$c_e = \sum_{j\colon\ \mathrm{out}(e)_j=1} \lambda_{I_j}$$

such that the edge cost c_e reflects the portion of the objective function $\lambda^T x$ contributed by including e in the path.

When making the transition to a turbo code $\mathscr{C}_{\mathrm{TC}}$, independently computing a shortest path P_{a} and P_{b} in each component trellis T_{a} and T_{b}, respectively, would not ensure that P_{a} and P_{b} are *agreeable*, i.e., fulfill (4.14), and hence match a codeword of $\mathscr{C}_{\mathrm{TC}}$. An ML turbo decoding algorithm would thus need to compute the minimum-cost pair of paths $(P_{\mathrm{a}}, P_{\mathrm{b}})$ in T_{a} and T_{b} that additionally is agreeable. While there is no known combinatorial algorithm that efficiently solves such a type of problem (which can be viewed as a generalization of what is called the *equal flow problem* [37]), it is nontheless possible to combine LP decoding with the trellis structure of turbo codes.

One approach is to resort to an LP formulation of the two shortest path problems on T_{a} and T_{b} as described in Section 3.4, and then link them by adding linear constraints that represent (4.14): first, write down the constraints of (3.15) for each trellis T_{a} and T_{b}, where we assume that the decision variable representing an edge e is called f_e instead of x_e. Then, for $i = 1, \dots, k$, add an additional contstraint

$$\sum_{\substack{e \in E_i^{\mathrm{a}}: \\ \mathrm{in}(e)=1}} f_e = \sum_{\substack{e \in E_{\pi(i)}^{\mathrm{b}}: \\ \mathrm{in}(e)=1}} f_e$$

to model (4.14), where E_i^{x} is the edge set of the ith segment of trellis T_{x}, $\mathrm{x} \in \{\mathrm{a}, \mathrm{b}\}$. By adding these constraints, the resulting polytope is no longer integral, i.e., the constraints introduce *fractional* vertices that do not correspond to an agreeable pair of paths. Note that there are no x variables for the codeword in the model, since their values are uniquely determined by the values of the f_e; it is hence not necessary to include them during the optimization process.

In a similar way as described above (see Paper IV for the details), a cost value derived from the LLR vector can be assigend to each edge such that the *integer* programming version of the model with additional constraints $f_e \in \{0, 1\}$ is equivalent to ML decoding. Its LP relaxation, called the *turbo LP decoder*, thus has similar properties to the LP decoder from Definition 5.1; in particular, it exhibits the ML certificate property.

While the error-correction performance of the turbo LP decoder has shown to be better than that of the usual LP decoder run on a parity-check matrix representation of the turbo code, solving the turbo LP with a generic LP solver, such as the simplex method, does not exploit the abovementioned possibility to decode the constituent convolutional codes in linear time. In Paper IV, we present an algorithm using this linear-time method as a subroutine to solve the turbo decoding LP in a very efficient way.

Part II

Contributions

Chapter 6

Paper I: Mathematical Programming Decoding of Binary Linear Codes: Theory and Algorithms

Michael Helmling, Stefan Ruzika, and Akın Tanatmis

The following is a reformatted and revised copy of a preprint that is publicly available online (http://arxiv.org/abs/1107.3715). The same content appeared in the following publication:

M. Helmling, S. Ruzika, and A. Tanatmis. "Mathematical programming decoding of binary linear codes: theory and algorithms". *IEEE Transactions on Information Theory* 58.7 (July 2012), pp. 4753–4769. DOI: 10.1109/TIT.2012.2191697. arXiv: 1107.3715 `[cs.IT]`

This work was supported in part by the Center of Mathematical and Computational Modeling (CM)^2, University of Kaiserslautern, Kaiserslautern, Germany.

Mathematical Programming Decoding of Binary Linear Codes: Theory and Algorithms

Michael Helmling Stefan Ruzika Akın Tanatmis

Mathematical programming is a branch of applied mathematics and has recently been used to derive new decoding approaches, challenging established but often heuristic algorithms based on iterative message passing. Concepts from mathematical programming used in the context of decoding include linear, integer, and nonlinear programming, network flows, notions of duality as well as matroid and polyhedral theory. This survey article reviews and categorizes decoding methods based on mathematical programming approaches for binary linear codes over binary-input memoryless symmetric channels.

6.1 Introduction

Based on an integer programming (IP)[1] formulation of the maximum likelihood decoding (MLD) problem for binary linear codes, linear programming decoding (LPD) was introduced by Feldman *et al.* [1, 2]. Since then, LPD has been intensively studied in a variety of articles especially dealing with low-density parity-check (LDPC) codes. LDPC codes are generally decoded by heuristic approaches called iterative message-passing decoding (IMPD) subsuming sum-product algorithm decoding (SPAD) [3, 4] and min-sum algorithm decoding (MSAD) [5]. In these algorithms, probabilistic information is iteratively exchanged and updated between component decoders. Initial messages are derived from the channel output. IMPD exploits the sparse structure of parity-check matrices of LDPC and turbo codes very well and achieves good performance. However, IMPD approaches are neither guaranteed to converge nor do they have the maximum likelihood (ML) certificate property, i.e., if the output is a codeword, it is not necessarily the ML codeword. Furthermore, performance of IMPD is poor for arbitrary linear block codes with a dense parity-check matrix. In contrast, LPD offers some advantages and thus has become an important alternative decoding technique. First, this approach is derived from the discipline of mathematical programming which provides analytical statements on convergence, complexity, and correctness of decoding algorithms. Second, LPD is not limited to sparse matrices.

This article is organized as follows. In Section 6.2, notation is fixed and well-known but relevant results from coding theory and polyhedral theory are recalled. Complexity and polyhedral properties of MLD are discussed in Section 6.3. In Section 6.4 a general description of LPD is given. Several linear programming (LP) formulations dedicated to codes with low-density parity-check matrices, codes with high-density parity-check matrices, and turbo-like codes are categorized and their commonalities and differences are emphasized in Section 6.5. Based on these LP formulations, different streams of research on LPD have evolved. Methods focusing on efficient realization of LPD are summarized in Section 6.6, while approaches improving the error-correcting performance of LPD at the cost of increased complexity are reviewed in Section 6.7. Some concluding comments are made in Section 6.8.

6.2 Basics and Notation

This section briefly introduces a number of definitions and results from linear coding theory and polyhedral theory which are most fundamental for the subsequent text.

A binary linear block code C with cardinality 2^k and block length n is a k-dimensional subspace of the vector space $\{0,1\}^n$ defined over the binary field $\mathbb{F}_2$. $C \subseteq \{0,1\}^n$ is given by k basis vectors of length n which are arranged in a $k \times n$ matrix G, called the generator matrix of the code C.[2]

---

[1] See the table on page 83 for a list of the acronyms used in this paper.

[2] Note that single vectors in this paper are generally column vectors; however, in coding theory they are often used as rows of matrices. The transposition of column vector a makes it a row vector, denoted by a^T.

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

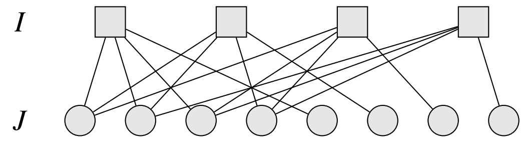


Figure 6.1: Parity-check matrix and Tanner graph of an (8,4) code. The square nodes represent the check nodes, while the round ones correspond to the variables.

The orthogonal subspace $C^{\perp}$ of C is defined as

$$C^{\perp} = \left\{ y \in \{0,1\}^n \colon \sum_{j=1}^{n} x_j y_j \equiv 0 \pmod 2 \text{ for all } x \in C \right\}$$

and has dimension $n-k$. It can also be interpreted as a binary linear code of dimension $n-k$ which is referred to as the dual code of C. A matrix $H \in \{0,1\}^{m \times n}$ whose $m \geq n-k$ rows form a spanning set of $C^{\perp}$ is called a parity-check matrix of C. It follows from this definition that C is the null space of H and thus a vector $x \in \{0,1\}^n$ is contained in C if and only if $Hx \equiv 0 \pmod 2$. Normally, $m = n-k$ and the rows of $H \in \{0,1\}^{(n-k) \times n}$ constitute a basis of $C^{\perp}$. It should be pointed out, however, that most LPD approaches (see Section 6.7) benefit from parity-check matrices being extended by redundant rows. Moreover, additional rows of H never degrade the error-correcting performance of LPD. This is a major difference to IMPD which is generally weakened by redundant parity checks, since they introduce cycles to the Tanner graph.

Let $x, x' \in \{0,1\}^n$. The Hamming distance between x and x' is the number of entries (bits) with different values, i.e., $d(x,x') = \left|\{1 \leq j \leq n \colon x_j \neq x'_j\}\right|$. The minimum (Hamming) distance of a code, $d(C)$, is given by $d(C) = \min\{d(x,x') \colon x, x' \in C, x \neq x'\}$. The Hamming weight of a codeword $x \in C$ is defined as $w(x) = d(x,0)$, i.e., the number of ones in x. The minimum Hamming weight of C is $w(C) = \min\{w(x) \colon x \in C, x \neq 0\}$. For binary linear codes it holds that $d(C) = w(C)$. The error-correcting performance of a code is, at least at high signal-to-noise ratio (SNR), closely related to its minimum distance.

Let $A \in \mathbb{R}^{m \times n}$ denote an $m \times n$ matrix and $I = \{1, \dots, m\}$, $J = \{1, \dots, n\}$ be the row and column index sets of A, respectively. The entry in row $i \in I$ and column $j \in J$ of A is given by $A_{i,j}$. The i^{th} row and j^{th} column of A are denoted by $A_{i,.}$ and $A_{.,j}$, respectively. A vector $e \in \mathbb{R}^m$ is called the i^{th} unit column vector if $e_i = 1$, $i \in I$, and $e_h = 0$ for all $h \in I \setminus \{i\}$.

A parity-check matrix H can be represented by a bipartite graph $G = (V, E)$, called its Tanner graph (Figure 6.1). The vertex set V of G consists of the two disjoint node sets I and J. The nodes in I are referred to as check nodes and correspond to the rows of H whereas the nodes in J are referred to as variable nodes and correspond to columns of H. An edge $[i,j] \in E$ connects node i and j if and only if $H_{i,j} = 1$. Let $N_i = \{j \in J \colon H_{ij} = 1\}$ denote the index set of variables incident to check node i, and analogously $N_j = \{i \in I \colon H_{ij} = 1\}$ for $j \in J$. The degree of a check node i is the number of edges incident to node i in the Tanner graph or, equivalently, $d_c(i) = |N_i|$. The maximum check node degree $d_c^{\max}$ is the degree of the check node $i \in I$ with the largest number of incident edges. The degree of a variable node j, $d_v(j)$, and the maximum variable node degree $d_v^{\max}$ are defined analogously.

Tanner graphs are an example of factor graphs, a general concept of graphical models which is prevalently used to describe probabilistic systems and related algorithms. The term stems from viewing the graph as the representation of some global function in several variables that factors into a product of subfunctions, each depending only on a subset of the variables. In case of Tanner graphs, the global function is the indicator function of the code, and the subfunctions are the parity-checks according to single rows of H. A different type of factor graphs will appear later in order to describe turbo codes. Far beyond these purely descriptive purpose, factor graphs have proven successful in modern coding theory primarily in the context of describing and analyzing IMPD algorithms. See [6] for a more elaborate introduction.

Let C be a binary linear code with parity-check matrix H and $x \in C \subseteq \{0,1\}^n$. The index set $\operatorname{supp}(x) = \{j \in J \colon x_j = 1\}$ is called the support of the codeword x. A codeword $0 \neq x \in C$ is called a minimal codeword if there is no codeword $0 \neq y \in C$ such that $\operatorname{supp}(y) \subseteq \operatorname{supp}(x)$. Finally, D is called a minor code of C if D can be obtained from D by a series of shortening and puncturing operations.

The relationship between binary linear codes and polyhedral theory follows from the observation that a binary linear code can be considered a set of points in $\mathbb{R}^n$, i.e., $C \subseteq \{0,1\}^n \subseteq \mathbb{R}^n$. In the following, some relevant results from polyhedral theory are recalled. For a comprehensive review on polyhedral theory the reader is referred to [7].

6.1 Definition: A subset $\mathcal{P}(A,b) \subseteq \mathbb{R}^n$ such that $\mathcal{P}(A,b) = \{v \in \mathbb{R}^n \colon Av \leq b\}$ where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is called a polyhedron. ◁

In this article, polyhedra are assumed to be rational, i.e., the entries of A and b are taken from $\mathbb{Q}$. The i^{th} row vector of A and the i^{th} entry of b together define a closed halfspace $\{v \in \mathbb{R}^n \colon A_{i,.} v \leq b_i\}$. In other words, a polyhedron is the intersection of a finite set of closed halfspaces. A bounded polyhedron is called a polytope. It is known from polyhedral theory that a polytope can equivalently be defined as the convex hull of a finite set of points. In this work, the convex hull of a binary linear code C is denoted by $\operatorname{conv}(C)$ and referred to as the codeword polytope.

Some characteristics of a polyhedron are its dimension, faces, and facets. To define them, the notion of a valid inequality is needed.

6.2 Definition: An inequality $r^T v \leq t$, where $r \in \mathbb{R}^n$ and $t \in \mathbb{R}$, is valid for a set $\mathcal{P}(A,b) \subseteq \mathbb{R}^n$ if $\mathcal{P}(A,b) \subseteq \{v \colon r^T v \leq t\}$. ◁

The following definition of an active inequality is used in several LPD algorithms.

6.3 Definition: An inequality $r^T v \leq t$, where $r, v \in \mathbb{R}^n$ and $t \in \mathbb{R}$, is active at $v^* \in \mathbb{R}^n$ if $r^T v^* = t$. ◁

Valid inequalities which contain points of $\mathcal{P}(A,b)$ are of special interest.

6.4 Definition: Let $\mathcal{P}(A,b) \subseteq \mathbb{R}^n$ be a polyhedron, let $r^T v \leq t$ be a valid inequality for $\mathcal{P}(A,b)$ and define $F = \{v \in \mathcal{P}(A,b) \colon r^T v = t\}$. Then F is called a face of $\mathcal{P}(A,b)$. F is a proper face if $F \neq \emptyset$ and $F \neq \mathcal{P}(A,b)$. ◁

The dimension $\dim(\mathcal{P}(A,b))$ of $\mathcal{P}(A,b) \subseteq \mathbb{R}^n$ is given by the maximum number of affinely independent points in $\mathcal{P}(A,b)$ minus one. Recall that a set of vectors $v^1, \dots, v^k$ is affinely independent if the system $\{\sum_{i=1}^k \lambda_k v^k = 0, \sum_{i=1}^k \lambda_k = 0\}$ has no solution other than $\lambda_i = 0$ for $i = 1, \dots, k$. If $\dim(\mathcal{P}(A,b)) = n$, then the polyhedron is full-dimensional. It is a well-known result that if $\mathcal{P}(A,b)$ is not full-dimensional, then there exists at least one inequality $A_{i,.} v \leq b_i$ such that $A_{i,.} v = b_i$ holds for all $v \in \mathcal{P}(A,b)$ (see e.g. [7]). Also, we have $\dim(F) \leq \dim(\mathcal{P}(A,b)) - 1$ for any proper face of $\mathcal{P}(A,b)$. A face $F \neq \emptyset$ of $\mathcal{P}(A,b)$ is called a facet of $\mathcal{P}(A,b)$ if $\dim(F) = \dim(\mathcal{P}(A,b)) - 1$.

In the set of inequalities defined by (A,b), some inequalities $A_{i,.} v \leq b_i$ may be redundant, i.e., dropping these inequalities does not change the solution set defined by $Av \leq b$. A standard result states that the facet-defining inequalities give a complete non-redundant description of a polyhedron $\mathcal{P}(A,b)$ [7].

A point $v \in \mathcal{P}(A,b)$ is called a vertex of $\mathcal{P}(A,b)$ if there exist no two other points $v^1, v^2 \in \mathcal{P}(A,b)$ such that $v = \mu_1 v^1 + \mu_2 v^2$ with $0 \leq \mu_1 \leq 1$, $0 \leq \mu_2 \leq 1$, and $\mu_1 + \mu_2 = 1$. Alternatively, vertices are zero dimensional faces of $\mathcal{P}(A,b)$. In an LP problem, a linear cost function is minimized on a polyhedron, i.e., $\min\{c^T x \colon x \in \mathcal{P}(A,b)\}$, $c \in \mathbb{R}^n$. Unless the LP is infeasible or unbounded, the minimum is attained on one of the vertices.

The number of constraints of an LP problem may be very large, e.g. Section 6.5 contains LPD formulations whose description complexity grows exponentially with the block length for general codes. In such a case it would be desirable to only include the constraints which are necessary to determine the optimal solution of the LP with respect to a given objective function. This can be accomplished by iteratively solving the associated separation problem, defined as follows.

6.5 Definition: Let $\mathcal{P}(A,b) \subset \mathbb{R}^n$ be a rational polyhedron and $v^* \in \mathbb{R}^n$ a rational vector. The separation problem is to either conclude that $v^* \in \mathcal{P}(A,b)$ or, if not, find a rational vector $(r,t) \in \mathbb{R}^n \times \mathbb{R}$ such that $r^T v \leq t$ for all $v \in \mathcal{P}(A,b)$ and $r^T v^* > t$. In the latter case, (r,t) is called a valid cut. ◁

We will see applications of this approach in Sections 6.6 and 6.7.

There is a famous result about the equivalence of optimization and separation by Grötschel *et al.* [8].

6.6 Theorem: *Let $\mathcal{P}$ be a proper class of polyhedra (see e.g. [7] for a definition). The optimization problem for $\mathcal{P}$ is polynomial time solvable if and only if the separation problem is polynomial time solvable.* ◁

6.3 Complexity and Polyhedral Properties

In this section, after referencing important NP-hardness results for the decoding problem, we state useful properties of the codeword polytope, exploiting a close relation between coding and matroid theory.

Integer programming provides powerful means for modeling several real-world problems. MLD for binary linear codes is modeled as an IP problem in [2, 9]. Let $y \in \mathbb{R}^n$ be the channel output. In MLD the probability (or, in case of a continuous-output channel, the probability density) $P(y \mid x)$ is maximized over all codewords $x \in C$. Let x^* denote the ML codeword. It is shown in [1] that for a symmetric memoryless channel the calculation of x^* amounts to the minimization of a linear cost function, namely

$$x^* = \arg\max_{x \in C} P(y \mid x) = \arg\min_{x \in C} \sum_{j=1}^{n} \lambda_j x_j, \tag{6.1}$$

where the values $\lambda_j = \log \frac{P(y_j|x_j=0)}{P(y_j|x_j=1)}$ are the so-called log-likelihood ratios (LLR). Consequently the IP formulation of MLD is implicitly given as

$$\min\{\lambda^T x \colon x \in C\}. \tag{6.2}$$

Berlekamp *et al.* have shown that MLD is NP-hard in [10] by a polynomial-time reduction of the three-dimensional matching problem to the decision version of MLD. An alternative proof is via matroid theory: as shall be exposed shortly, there is a one-to-one correspondence between binary matroids and binary linear codes. In virtue of this analogy, MLD is equivalent to the minimum-weight cycle problem on binary matroids. Since the latter contains the max-cut problem, which is known to be NP-hard [11], as a special case, the NP-hardness of MLD follows.

Another problem of interest in the framework of coding theory is the computation of the minimum distance of a given code. Berlekamp *et al.* [10] conjectured that computing the distance of a binary linear code is NP-hard as well, which was proved by Vardy [12] about two decades later. The minimum distance problem can again be reformulated in a matroid theoretic setting. In 1969 Welsh [13] formulated it as the problem of finding a minimum cardinality circuit in linear matroids.

In the following, we assume $C \subseteq \{0, 1\}^n$ to be canonically embedded in $\mathbb{R}^n$ when referring to $\operatorname{conv}(C)$ (see Figure 6.2 for an example). Replacing C by $\operatorname{conv}(C)$ in (6.2) leads to a linear programming problem over a polytope with integer vertices. In general, computing an explicit representation of $\operatorname{conv}(C)$ is intractable. Nevertheless, some properties of $\operatorname{conv}(C)$ are known from matroid theory due to the equivalence of binary linear codes and binary matroids. In the following, some definitions and results from matroid theory are presented. An extensive investigation of matroids can be found in [14] or [15]. The definition of a matroid in general is rather technical.

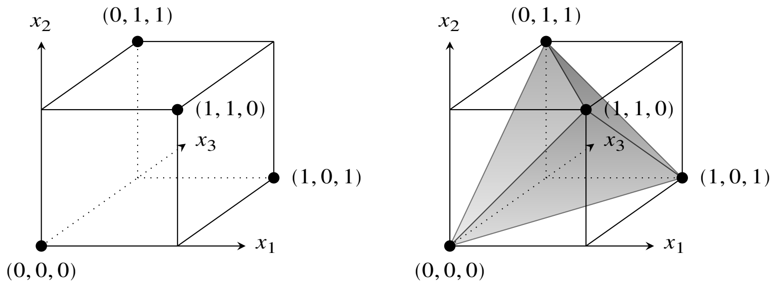


Figure 6.2: The codewords of the single parity-check code $C = \{x \in \mathbb{F}_2^3 \colon x_1 + x_2 + x_3 \equiv 0 \pmod 2\}$ and the polytope $\operatorname{conv}(C)$ in $\mathbb{R}^3$.

6.7 Definition: A matroid $\mathscr{M}$ is an ordered pair $\mathscr{M} = (J, \mathscr{U})$ where J is a finite ground set and $\mathscr{U}$ is a collection of subsets of J, called the independent sets, such that (1)–(3) hold.

(1) $\emptyset \in \mathscr{U}$.

(2) If $u \in \mathscr{U}$ and $v \subset u$, then $v \in \mathscr{U}$.

(3) If $u_1, u_2 \in \mathscr{U}$ and $|u_1| < |u_2|$ then there exists $j \in u_2 \setminus u_1$ such that $u_1 \cup \{j\} \in \mathscr{U}$. ◁

In this work, the class of $\mathbb{F}_2$-representable (i.e., binary) matroids is of interest. A binary $m \times n$ matrix H defines an $\mathbb{F}_2$-representable matroid $\mathscr{M}[H]$ as follows. The ground set $J = \{1, \dots, n\}$ is defined to be the index set of the columns of H. A subset $U \subseteq J$ is independent if and only if the column vectors $H_{\cdot,u}$, $u \in U$ are linearly independent in the vector space defined over the field $\mathbb{F}_2$. A minimal dependent set, i.e., a set $\mathscr{V} \in 2^J \setminus \mathscr{U}$ such that all proper subsets of $\mathscr{V}$ are in $\mathscr{U}$, is called a circuit of $\mathscr{M}[H]$. If a subset of J is a disjoint union of circuits then it is called a cycle.

The incidence vector $x^{\mathscr{C}} \in \mathbb{R}^n$ corresponding to a cycle $\mathscr{C} \subseteq J$ is defined by

$$x_j^{\mathscr{C}} = \begin{cases} 1 & \text{if } j \in C, \\ 0 & \text{if } j \notin C. \end{cases}$$

The cycle polytope is the convex hull of the incidence vectors corresponding to all cycles of a binary matroid.

Some more relationships between coding theory and matroid theory (see also [16]) can be listed: a binary linear code corresponds to a binary matroid, the support of a codeword corresponds to a cycle (therefore, each codeword corresponds to the incidence vector of a cycle), the support of a minimal codeword corresponds to a circuit, and the codeword polytope $\operatorname{conv}(C)$ corresponds to the cycle polytope. Let H be a binary matrix, $\mathscr{M}[H]$ be the binary matroid defined by H (H is a representation matrix of $\mathscr{M}[H]$) and C be the binary linear code defined by H (H is a parity-check matrix of C). It can easily be shown that the dual $C^\perp$ of C is the same object as the dual of the binary matroid $\mathscr{M}[H]$. We denote the dual matroid by $\mathscr{M}[G]$, where G is the generator matrix of C. Usually the matroid related terms are dualized by the prefix “co”. For example, the circuits and cycles of a dual matroid are called cocircuits and cocycles, respectively. The supports of minimal codewords and the supports of codewords in $C^\perp$ are associated with cocircuits and cocycles of $\mathscr{M}[H]$, respectively.

A minor of a parent matroid $\mathscr{M} = (J, \mathscr{U})$ is the sub-matroid obtained from $\mathscr{M}$ after any combination of contraction and restriction operations (see e.g. [14]). In the context of coding theory, contraction corresponds to puncturing, i.e., the deletion of one or more columns from the generator matrix of a parent code, and restriction corresponds to shortening, i.e., the deletion of one or more columns from the parity-check matrix of a parent code.

Next, some results from Barahona and Grötschel [17] which are related to the structure of the cycle polytope are rewritten in terms of coding theory. Kashyap provides a similar transfer in [18]. Several results are collected in Theorem 6.8.

6.8 Theorem: *Let C be a binary linear code.*

(a) *If $d(C^\perp) \geq 3$ then the codeword polytope is full-dimensional.*

(b) *The box inequalities*

$$0 \leq x_j \leq 1 \quad \textit{for all } j \in J \tag{6.3}$$

and the cocircuit inequalities

$$\sum_{j \in \mathscr{F}} x_j - \sum_{j \in \operatorname{supp}(q) \setminus \mathscr{F}} x_j \leq |\mathscr{F}| - 1 \quad \textit{for all } \mathscr{F} \subseteq \operatorname{supp}(q) \textit{ with } |\mathscr{F}| \textit{ odd}, \tag{6.4}$$

where $\operatorname{supp}(q)$ *is the support of a dual minimal codeword q, are valid for the codeword polytope.*

(c) *The box inequalities $x_j \geq 0, x_j \leq 1$ define facets of the codeword polytope if $d(C^\perp) \geq 3$ and $j \in J$ is not contained in the support of a codeword in $C^\perp$ with weight three.*

(d) *If $d(C^\perp) \geq 3$ and C does not contain $H_7^\perp$ ((7,3,4) simplex code) as a minor, and if there exists a dual minimal codeword q of weight* 3*, then the cocircuit inequalities derived from* $\operatorname{supp}(q)$ *are facets of* $\operatorname{conv}(C)$*.* ◁

Part (b) of Theorem 6.8 implies that the set of cocircuit inequalities derived from the supports of all dual minimal codewords provide a relaxation of the codeword polytope. In the polyhedral analysis of the codeword polytope the symmetry property stated below plays an important role.

6.9 Theorem: *[17] If $a^T x \leq \alpha$ defines a face of* $\operatorname{conv}(C)$ *of dimension d, and y is a codeword, then the inequality $\bar{a}^T x \leq \bar{\alpha}$ also defines a face of* $\operatorname{conv}(C)$ *of dimension d, where*

$$\bar{a}_j = \begin{cases} a_j & \textit{if } j \notin \operatorname{supp}(y), \\ -a_j & \textit{if } j \in \operatorname{supp}(y), \end{cases}$$

and $\bar{\alpha} = \alpha - a^T y$. ◁

Using this theorem, a complete description of $\operatorname{conv}(C)$ can be derived from all facets containing a single codeword [17].

Let q be a dual minimal codeword. To identify if the cocircuit inequalities derived from $\operatorname{supp}(q)$ are facet-defining it should be checked if $\operatorname{supp}(q)$ has a chord. For the formal definition of chord, the symmetric difference $\triangle$ which operates on two finite sets is used, defined by $A \triangle B = (A \setminus B) \cup (B \setminus A)$. Note that if $A = \operatorname{supp}(q_1)$, $B = \operatorname{supp}(q_2)$ and $\operatorname{supp}(q_0) = A \triangle B$, then $q_0 \equiv q_1 + q_2 \pmod 2$.

6.10 Definition: Let $q_0, q_1, q_2 \in C^{\perp}$ be dual minimal codewords. If

$$\operatorname{supp}(q_0) = \operatorname{supp}(q_1) \triangle \operatorname{supp}(q_2) \quad \text{and} \quad \operatorname{supp}(q_1) \cap \operatorname{supp}(q_2) = \{j\},$$

then j is called a chord of $\operatorname{supp}(q_0)$. ◁

6.11 Theorem: *[17] Let C be a binary linear code without the* $(7,3,4)$ *simplex code as a minor and let* $\operatorname{supp}(q)$ *be the support of a dual minimal codeword with Hamming weight at least* 3 *and without chord. Then for all* $\mathscr{F} \subseteq \operatorname{supp}(q)$ *with* $|\mathscr{F}|$ *odd, the inequality*

$$\sum_{j \in \mathscr{F}} x_j - \sum_{j \in \operatorname{supp}(q) \setminus \mathscr{F}} x_j \leq |\mathscr{F}| - 1$$

defines a facet of $\operatorname{conv}(C)$. ◁

Optimizing a linear cost function over the cycle polytope, known as the cycle problem in terms of matroid theory, is investigated by Grötschel and Truemper [19]. The work of Feldman *et al.* [2] enables to use the matroid theoretic results in the coding theory context. As shown above, solving the MLD problem for a binary linear code is equivalent to solving the cycle problem on a binary matroid. In [19], binary matroids for which the cycle problem can be solved in polynomial time are classified, based on Seymour's matroid decomposition theory [20]. Kashyap [16] shows that results from [19] are directly applicable to binary linear codes. The MLD problem as well as the minimum distance problem can be solved in polynomial time for the code families for which the cycle problem on the associated binary matroid can be solved in polynomial time. This code family is called polynomially almost-graphic codes [16].

An interesting subclass of polynomially almost-graphic codes are geometrically perfect codes. Kashyap translates the sum of circuits property (see [19]) to the realm of binary linear codes. If the binary matroid associated with code C has the sum of circuits property then $\operatorname{conv}(C)$ can be described completely and non-redundantly by the box inequalities (6.3) and the cocircuit inequalities (6.4). These codes are referred to as geometrically perfect codes in [16]. The associated binary matroids of geometrically perfect codes can be decomposed in polynomial time into its minors which are either graphic (see [14]) or contained in a finite list of matroids.

From a coding theoretic point of view, a family of error-correcting codes is asymptotically bad if either dimension or minimum distance grows only sublinearly with the code length. Kashyap proves that the family of geometrically perfect codes unfortunately fulfills this property. We refer to [16] for the generalizations of this result.

6.4 Basics of LPD

LPD was first introduced in [2]. This decoding method is, in principle, applicable to any binary linear code over any binary-input memoryless channel.[3] In this section, we review the basics of the LPD approach based on [1].

Although several structural properties of $\operatorname{conv}(C)$ are known, it is in general infeasible to compute a concise description of $\operatorname{conv}(C)$ by means of linear inequalities. In LPD, the linear cost function of the IP formulation is minimized on a relaxed polytope $\mathcal{P}$ where $\operatorname{conv}(C) \subseteq \mathcal{P} \subseteq \mathbb{R}^n$. Such a relaxed polytope $\mathcal{P}$ should have the following desirable properties:

- $\mathcal{P}$ should be easy to describe, and
- integral vertices of $\mathcal{P}$ should correspond to codewords.

Together with the linear representation (6.1) of the likelihood function, this leads to one of the major benefits of LPD, the so-called ML certificate property: If the LP decoder outputs an integral optimal solution, it is guaranteed to be the ML codeword. This is a remarkable difference to IMPD: If no general optimality condition applies (see e.g. [23, Sec. 10.3]), there is no method to provably decide the optimality of a solution obtained by IMPD.

Each row (check node) $i \in I$ of a parity-check matrix H defines the local code

$$C_i = \left\{ x \in \{0,1\}^n \colon \sum_{j=1}^{n} H_{ij} x_j \equiv 0 \pmod 2 \right\}$$

that consists of the bit sequences which satisfy the i^{th} parity-check constraint; these are called local codewords. A particularly interesting relaxation of $\operatorname{conv}(C)$ is

$$\mathcal{P} = \operatorname{conv}(C_1) \cap \cdots \cap \operatorname{conv}(C_m) \subseteq [0,1]^n,$$

known as the fundamental polytope [24]. The vertices of the fundamental polytope, the so-called pseudocodewords, are a superset of C, where the difference consists only of non-integral vertices. Consequently, optimizing over $\mathcal{P}$ implies the ML certificate property. These observations are formally stated in the following result (note that $C = C_1 \cap \cdots \cap C_m$).

6.12 Lemma ([24]): *Let* $\mathcal{P} = \operatorname{conv}(C_1) \cap \cdots \cap \operatorname{conv}(C_m)$. *If* $C = C_1 \cap \cdots \cap C_m$ *then* $\operatorname{conv}(C) \subseteq \mathcal{P}$ *and* $C = \mathcal{P} \cap \{0,1\}^n$. ◁

The description complexity of the convex hull of any local code $\operatorname{conv}(C_i)$ and thus $\mathcal{P}$ is usually much smaller than the description complexity of the codeword polytope $\operatorname{conv}(C)$.

LPD can be written as optimizing the linear objective function on the fundamental polytope $\mathcal{P}$, i.e.,

$$\min\{\lambda^T x \colon x \in \mathcal{P}\}. \tag{6.5}$$

---

[3] In fact, Flanagan *et al.* [21] have recently generalized a substantial portion of the LPD theory to the nonbinary case. Similarly, work has been done to include channels with memory; see e.g. [22].

Based on (6.5), the LPD algorithm which we refer to as bare linear programming decoding (BLPD) is derived.

Bare LP decoding (BLPD)

Input: $\lambda \in \mathbb{R}^n$, $\mathcal{P} \subseteq [0,1]^n$.

Output: ML codeword or ERROR.

```
Solve the LP given in (6.5).
if LP solution $x^*$ is integral then
    Output $x^*$
else
    Output ERROR
end if
```

Because of the ML certificate property, if BLPD outputs a codeword, then it is the ML codeword. BLPD succeeds if the transmitted codeword is the unique optimum of the LP given in (6.5). BLPD fails if the optimal solution is non-integral or the ML codeword is not the same as the transmitted codeword. Note that the difference between the performance of BLPD and MLD is caused by the decoding failures for which BLPD finds a non-integral optimal solution. It should be emphasized that in case of multiple optima it is assumed that BLPD fails.

In some special cases, the fundamental polytope $\mathcal{P}$ is equivalent to $\operatorname{conv}(C)$, e.g., if the underlying Tanner graph is a tree or forest [24]. In these cases MLD can be achieved by BLPD. Note that in those cases also MSAD achieves MLD performance [5].

Observe that the minimum distance of a code can be understood as the minimum ℓ_1 distance between any two different codewords of C. Likewise the fractional distance of the fundamental polytope $\mathcal{P}$ can be defined as follows.

6.13 Definition: [2] Let $V(\mathcal{P})$ be the set of vertices (pseudocodewords) of $\mathcal{P}$. The fractional distance $d_{\text{frac}}(\mathcal{P})$ is the minimum ℓ_1 distance between a codeword and any other vertex of $V(\mathcal{P})$, i.e.

$$d_{\text{frac}}(\mathcal{P}) = \min \left\{ \sum_{j=1}^{n} \left| x_j - v_j \right| : x \in C,\ v \in V(\mathcal{P}),\ x \neq v \right\}.$$

◁

It follows that the fractional distance is a lower bound for the minimum distance of a code: $d(C) \geq d_{\text{frac}}(\mathcal{P})$. Moreover, both definitions are related as follows. Recall that on the binary symmetric channel (BSC), MLD corrects at least $\lceil d(C)/2 \rceil - 1$ bit flips. As shown in [1], LPD succeeds if at most $\lceil d_{\text{frac}}(\mathcal{P})/2 \rceil - 1$ errors occur on the BSC.

Analogously to the minimum distance, the fractional distance is equivalent to the minimum ℓ_1 weight of a non-zero vertex of $\mathcal{P}$. This property is used by the fractional distance algorithm (FDA) to compute the fractional distance of a binary linear code [1]. If $\mathcal{M}$ is the set of inequalities describing $\mathcal{P}$, let $\mathcal{M}_I$ be the subset of those inequalities which are not active at the all-zero codeword. Note that these are exactly the inequalities with a non-zero right hand side. In FDA the weight function $\sum_{j \in J} x_j$ is subsequently minimized on $\mathscr{P} \cap f$ for all $f \in \mathscr{M}_I$ in order to find the minimum-weight non-zero vertex of $\mathscr{P}$.

Fractional distance algorithm (FDA)

Input: $\mathscr{P} \subseteq [0,1]^n$.

Output: Minimum-weight non-zero vertex of $\mathscr{P}$.

```
for all $f \in \mathscr{M}_I$ do
    Set $\mathscr{P}' = \mathscr{P} \cap f$.
    Solve $\min \left\{ \sum_{j \in J} x_j : x \in \mathscr{P}' \right\}$.
end for
Choose the minimum value obtained over all $\mathscr{P}'$.
```

A more significant distance measure than d_{frac} is the so-called pseudo-distance which quantifies the probability that the optimal solution under LPD changes from one vertex of $\mathscr{P}$ to another [24, 25]. Likewise, the minimum pseudo-weight is defined as the minimum pseudo-distance from 0 to any other vertex of $\mathscr{P}$ and therefor identifies the vertex (pseudocodeword) which is most likely to cause a decoding failure. Note that the pseudo-distance takes the channel's probability measure into account and thus depends on the chosen channel model.

Albeit no efficient algorithms are known to compute the exact minimum pseudo-weight of the fundamental polytope of a code, promising heuristics as well as analytical bounds have been proposed [24–26].

6.5 LPD Formulations for Various Code Classes

This section reviews various formulations of the polytope $\mathscr{P}$ from (6.5), leading to optimized versions of the general BLPD algorithm for different classes of codes.

In Step 1 of BLPD the LP problem is solved by a general purpose LP solver. These solvers usually employ the simplex method since it performs well in practice. The simplex method iteratively examines vertices of the underlying polytope until the vertex corresponding to the optimal solution is reached. If there exists a neighboring vertex for which the objective function can be improved in the current step, the simplex method moves to this vertex. Otherwise it stops. The procedure of moving from one vertex to an other is called a simplex iteration. Details on the simplex algorithm can be found in classical books about linear programming (see e.g. [27]).

The efficiency of the simplex method depends on the complexity of the constraint set describing the underlying polytope. Several such explicit descriptions of the fundamental polytope $\mathscr{P}$ have been proposed in the LPD literature. Some can be used for any binary linear code whereas others are specialized for a specific code class. Using alternative descriptions of $\mathscr{P}$, alternative LP decoders are obtained. In the following, we are going to present different LP formulations.

6.5.1 LP Formulations for LDPC Codes

The solution algorithm referred to as BLPD in Section 6.4 was introduced by Feldman *et al.* [2]. In order to describe $\mathcal{P}$ explicitly, three alternative constraint sets are suggested by the authors by the formulations BLPD1, BLPD2, and BLPD3. In the following, some abbreviations are used to denote both the formulation and the associated solution (decoding) algorithm, e.g., solving an LP, subgradient optimization, neighborhood search. The meaning will be clear from the context.

The first LP formulation, BLPD1, of [2] is applicable to LDPC codes.

$$\begin{aligned}
\min \quad & \lambda^T x \qquad \text{(BLPD1)} && && \text{(6.6a)}\\
\text{s.t.} \quad & \sum_{S \in E_i} w_{i,S} = 1 && i = 1, \dots, m && \text{(6.6b)}\\
& x_j = \sum_{\substack{S \in E_i \\ \text{with } j \in S}} w_{i,S} && \forall j \in N_i,\ i = 1, \dots, m && \text{(6.6c)}\\
& 0 \le x_j \le 1 && j = 1, \dots, n && \text{(6.6d)}\\
& 0 \le w_{i,S} \le 1 && \forall S \in E_i,\ i = 1, \dots, m && \text{(6.6e)}
\end{aligned}$$

Here, $E_i = \{S \subseteq N_i \colon\ |S| \text{ even}\}$ is the set of valid bit configurations within N_i. The auxiliary variables $w_{i,S}$ used in this formulation indicate which bit configuration $S \in E_i$ is taken at parity check i. In case of an integral solution, (6.6b) ensures that exactly one such configuration is attained at every checknode, while (6.6c) connects the actual code bits, modeled by the variables x_j, to the auxiliary variables: $x_j = 1$ if and only if the set $S \in E_i$ contains j for every check node i. Note that here we consider the LP relaxation, so it is not guaranteed that a solution of the above program is indeed integral.

A second linear programming formulation for LDPC codes, BLPD2, is obtained by employing the so-called forbidden set (FS) inequalities [28]. The FS inequalities are motivated by the observation that one can explicitly forbid those value assignments to variables where $|S|$ is odd. For all local codewords in C_i it holds that

$$\sum_{j \in S} x_j - \sum_{j \in N_i \setminus S} x_j \le |S| - 1 \qquad \text{for all } S \in \Sigma_i$$

where $\Sigma_i = \{S \subseteq N_i \colon\ |S| \text{ odd}\}$. Feldman *et al.* show in [2] that for each single parity-check code C_i, the FS inequalities together with the box inequalities $0 \le x_j \le 1, j \in J$ completely and non-redundantly describe $\operatorname{conv}(C_i)$ (the case $|N_i| = 3$ as depicted in Figure 6.2 is the only exception where the box inequalities are not needed). In a more general setting, Grötschel proved this result for the cardinality homogeneous set systems [29].

If the rows of H are considered as dual codewords, the set of FS inequalities is a reinvention of cocircuit inequalities explained in Section 6.3. BLPD2 is given below.

$$\begin{array}{lll}
\min & \lambda^T x \qquad \text{(BLPD2)} & \\
\text{s.t.} & \sum\limits_{j \in S} x_j - \sum\limits_{j \in N_i \setminus S} x_j \leq |S| - 1 & \forall S \in \Sigma_i,\ i = 1, \ldots, m \\
& 0 \leq x_j \leq 1 & j = 1, \ldots, n
\end{array}$$

Feldman *et al.* [2] apply BLPD using formulations BLPD1 or BLPD2 to LDPC codes. Under the BSC, the error-correcting performance of BLPD is compared with the MSAD on an random rate-1/2 LDPC code with $n = 200$, $d_v = 3$, $d_c = 6$; with the MSAD, SPAD on the random rate-1/4 LDPC code with $n = 200$, $d_v = 3$, $d_c = 4$; with the MSAD, SPAD, MLD on the random rate-1/4 LDPC code with $n = 60$, $d_v = 3$, $d_c = 4$. On these codes, BLPD performs better than MSAD but worse than SPAD. Using BLPD2, the FDA is applied to random rate-1/4 LDPC codes with $n = 100$, 200, 300, 400, $d_v = 3$ and $d_c = 4$ from an ensemble of Gallager [30]. For $(n-1, n)$ Reed-Muller codes [31] with $4 \leq n \leq 512$ they compare the classical distance with the fractional distance. The numerical results suggest that the gap between both distances grows with increasing block length.

Another formulation for LDPC codes is given in Section 6.6.2 in the context of efficient implementations.

In a remarkable work, Feldman and Stein [32] have shown that the Shannon capacity of a channel can be achieved with LP decoding, which implies a polynomial-time decoder and the availability of an ML certificate. To this end, they use a slightly modified version of BLPD1 restricted to expander codes, which are a subclass of LDPC codes. See [32] for a formal definition of expander codes as well as the details of the corresponding decoder.

6.5.2 LP Formulations for Codes with High-Density Parity-Check Matrices

The number of variables and constraints in BLPD1 as well as the number of constraints in BLPD2 increase exponentially in the check node degree. Thus, for codes with high-density parity-check matrices, BLPD1 and BLPD2 are computationally inefficient. A polynomial-sized formulation, BLPD3, is based on the parity polytope of Yannakakis [33]. There are two types of auxiliary variables in BLPD3. The variable $p_{i,k}$ is set to one if k variable nodes are set to one in the neighborhood of parity-check i, for k in the index set

$$K_i = \left\{0, 2, \ldots, 2\left\lfloor \frac{|N_i|}{2} \right\rfloor\right\}.$$

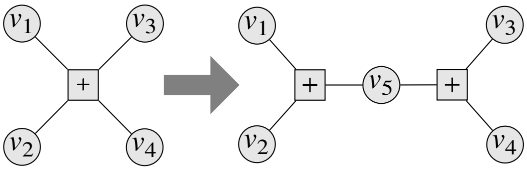


Figure 6.3: Check node decomposition for high-density parity-check codes according to [34].

Furthermore, the variable $q_{j,i,k}$ is set to one if variable node j is one of the k variable nodes set to one in the neighborhood of check node i.

$$\begin{aligned}
\min \quad & \lambda^T x \qquad \text{(BLPD3)} & \\
\text{s.t.} \quad & x_j = \sum_{k \in K_i} q_{j,i,k} & i \in N_j,\ j = 1, \dots, n \\
& \sum_{k \in K_i} p_{i,k} = 1 & i = 1, \dots m \\
& \sum_{j \in N_i} q_{j,i,k} = k p_{i,k} & k \in K_i,\ i = 1, \dots m \\
& 0 \le x_j \le 1 & j = 1, \dots, n \\
& 0 \le p_{i,k} \le 1 & k \in K_i,\ i = 1, \dots, m \\
& 0 \le q_{j,i,k} \le p_{i,k} & k \in K_i,\ j = 1, \dots, n,\ i \in N_j
\end{aligned}$$

Feldman *et al.* [2] show that BLPD1, BLPD2, and BLPD3 are equivalent in the sense that the x-variables of the optimal solutions in all three formulations take the same values.

The number of variables and constraints in BLPD3 increases as $O(n^3)$. By applying a decomposition approach, Yang *et al.* [34] show that an alternative LP formulation which has size linear in the length and check node degrees can be obtained (it should be noted that independently from [34] a similar decomposition approach was also proposed in [35]). In the LP formulation of [34] a high degree check node is decomposed into several low degree check nodes. Thus, the resulting Tanner graph contains auxiliary check and variable nodes. Figure 6.3 illustrates this decomposition technique: a check node with degree 4 is decomposed into 2 parity checks each with degree at most 3. The parity-check nodes are illustrated by squares. In the example, original variables are denoted by $v_1, \dots, v_4$ while the auxiliary variable node is named v_5. In general, this decomposition technique is iteratively applied until every check node has degree less than 4. The authors show that the total number of variables in the formulation is less than doubled by the decomposition. For the details of the decomposition [34] is referred.

For the ease of notation, suppose K is the set of parity-check nodes after decomposition. If $d_c(k) = 3$, $k \in K$, then the parity-check constraint k is of the form $v_1^k + v_2^k + v_3^k \equiv 0 \pmod 2$. Note that with our notation some of these variables v_s^k might represent the same variable node v_j, e.g. v_5 from Figure 6.3 would appear in two constraints of the above form, as v_s^1 and $v_{s'}^2$, respectively. Yang *et al.* show that the parity-check constraint $v_1^k + v_2^k + v_3^k \equiv 0 \pmod 2$ can be replaced by the linear constraints

$$\begin{aligned} v_1^k + v_2^k + v_3^k &\leq 2, \\ v_1^k - v_2^k - v_3^k &\leq 0, \\ v_2^k - v_1^k - v_3^k &\leq 0, \\ v_3^k - v_1^k - v_2^k &\leq 0 \end{aligned}$$

(for a single check node of degree 3 the box inequalities are not needed). If $d_c(k) = 2$ then $v_1^k = v_2^k$ along with the box constraints models the parity-check. The constraint set of the resulting LP formulation, which we call cascaded linear programming decoding (CLPD), is the union of all constraints modeling the $|K|$ parity checks.

$$\begin{array}{lll} \min & \bar{\lambda}^T v \qquad \text{(CLPD)} & \\ \text{s.t.} & \sum_{j \in S} v_j^k - \sum_{j \in N_k \setminus S} v_j^k \leq |S| - 1 & \forall S \in \Sigma_k,\ k = 1, \ldots, |K| \\ & 0 \leq v_j \leq 1 & \text{if } d_c(i) \leq 2\ \forall\, i \colon j \in N_i \end{array}$$

In the objective function only the v variables corresponding to the original x variables have non-zero coefficients. Thus, the objective function of CLPD is the same as of BLPD1. The constraints in CLPD are the FS inequalities used in BLPD2 with the property that the degree of the check node is less than 4.

Yang *et al.* prove that the formulations introduced in [2] and CLPD are equivalent. Again, equivalence is used in the sense that in an optimal solution, the x-variables of BLPD1, BLPD2, BLPD3, and the variables of the CLPD formulation which correspond to original x-variables take the same values. Moreover, it is shown that CLPD can be used in FDA. As a result, the computation of the fractional distance for codes with high-density parity-check matrices is also facilitated. Note that using BLPD2, the FDA algorithm has polynomial running time only for LDPC codes. If $\mathcal{P}$ is described by the constraint set of CLPD, then in the first step of the FDA, it is sufficient to choose the set $\mathcal{F}$ from the facets formed by cutting planes of type $v_1^k + v_2^k + v_3^k = 2$ where v_1^k, v_2^k, and v_3^k are variables of the CLPD formulation. Additionally, an adaptive branch & bound method is suggested in [36] to find better bounds for the minimum distance of a code. On a random rate-1/4 LDPC code with $n = 60$, $d_v = 3$ and $d_c = 4$, it is demonstrated that this yields a better lower bound than the fractional distance does.

6.5.3 LP Formulations for Turbo-Like Codes

The various LP formulations outlined so far have in common that they are derived from a parity-check matrix which defines a specific code. A different approach is to describe the encoder by means of a finite state machine, which is the usual way to define so-called convolutional codes. The bits of the information word are subsequently fed into the machine, each causing a state change that emits a fixed number of output bits depending on both the current state and the input. In a systematic code, the output always contains the input bit. The codeword, consisting

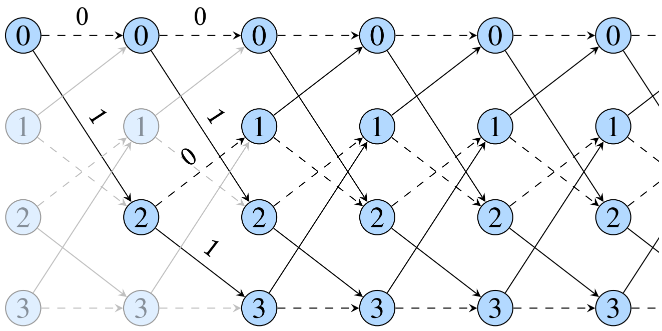


Figure 6.4: Excerpt from a trellis graph with four states and initial state 0. The style of an edge indicates the respective information bit, while the labels refer to the single parity bit.

of the concatenation of all outputs, can thus be partitioned into the systematic part which is a copy of the input and the remaining bits, being referred to as the parity output.

A convolutional code is naturally represented by a trellis graph (Figure 6.4), which is obtained by unfolding the state diagram in the time domain. Each vertex of the trellis represents the state at a specific point in time, while edges correspond to valid transitions between two subsequent states and are labelled by the corresponding input and output bits. Each path from the starting node to the end node corresponds to a codeword.[4] The cost of a codeword is derived from the received LLR values and the edge labels on the path associated with this codeword. See [23] for an in-depth survey of these concepts.

Convolutional codes are the building blocks of turbo codes, which revolutionized coding theory because of their near Shannon limit error-correcting performance [37]. An (n, k) turbo code consists of two convolutional codes C_a and C_b, each of input length k, which are linked by a so-called interleaver that requires the information bits of C_a to match those of C_b after being scrambled by some permutation $\pi \in \mathbb{S}_k$ which is fixed for a given code.[5] It is this coupling of rather weak individual codes and the increase of complexity arising therefrom that entails the vast performance gain of turbo codes. A typical turbo code (and only this case is covered here; it is straightforward to generalize) consists of two identical systematic encoders of rate $\frac{1}{2}$ each. Only one of the encoders C_a and C_b, however, contributes its systematic part to the resulting codeword, yielding an overall rate of $2/3$, i.e. $n = 3k$ (since their systematic parts differ only by a permutation, including both would imply an embedded repetition code). We thus partition a codeword x into the systematic part x^s and the parity outputs x^a and x^b of C_a and C_b, respectively.

---

[4] We intentionally do not discuss trellis termination here and assume that the encoder always ends in a fixed terminal state; cf. [23] for details.

[5] Using exactly two constituent convolutional encoders eases notation and is the most common case, albeit not being essential for the concept—in fact, recent development suggest that the error-correcting performance benefits from adding a third encoder [38].

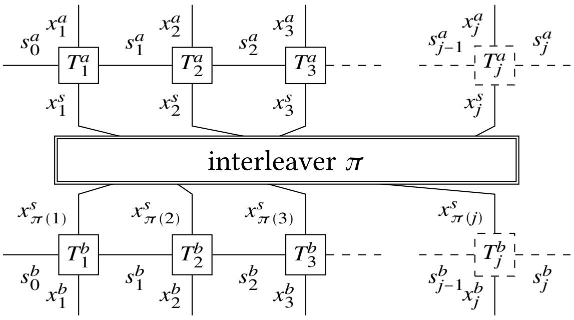


Figure 6.5: The Forney-style factor graph of a turbo code. The interleaver π links the systematic bits x^s of both encoders C_a (upper part) and C_b (lower part).

A turbo code can be compactly represented by a so-called Forney-style factor graph (FFG) as shown in Figure 6.5. As opposed to Tanner graphs, in an FFG all nodes are functional nodes, whereas the (half-)edges correspond to variables. In our case, there are variables of two types, namely state variables s_j^ν ($\nu \in \{a,b\}$), reflecting the state of C_ν at time step j, and a variable for each bit of the codeword x. Each node T_j^ν represents the indicator function for a valid state transition in C_ν at time j and is thus incident to one systematic and one parity variable as well as the "before" and "after" state s_{j-1}^ν and s_j^ν, respectively. Note that such a node T_j^ν corresponds to a vertical "slice" (often called a segment) of the trellis graph of C_ν, and each valid configuration of T_j^ν is represented by exactly one edge in the respective segment.

Turbo codes are typically decoded by IMPD techniques operating on the factor graph. Feldman [1] in contrast introduced an LP formulation, turbo code linear programming decoding (TCLPD), for this purpose. This serves as an example that mathematical programming is a promising approach in decoding even beyond formulations based on parity-check matrices.

In TCLPD, the trellis graph of each constituent encoder C^ν is modeled by flow conservation and capacity constraints [39], along with side constraints appropriately connecting the flow variables f^ν to auxiliary variables x^s and x^ν, respectively, which embody the codeword bits.

For $\nu \in \{a,b\}$, let $G_\nu = (S_\nu, E_\nu)$ be the trellis corresponding to C_ν, where S_ν is the index set of nodes (states) and E_ν is the set of edges (state transitions) e in G_ν. Let $s^{\text{start},\nu}$ and $s^{\text{end},\nu}$ denote the unique start and end node, respectively, of G_ν. We can now define a feasible flow f^ν in the trellis G_ν by the system

$$\sum_{e \in \text{out}(s^{\text{start},\nu})} f_e^\nu = 1, \qquad \sum_{e \in \text{in}(s^{\text{end},\nu})} f_e^\nu = 1, \tag{6.7a}$$

$$\sum_{e \in \text{out}(s)} f_e^\nu = \sum_{e \in \text{in}(s)} f_e^\nu \qquad \forall\, s \in S_\nu \setminus \{s^{\text{start},\nu}, s^{\text{end},\nu}\}, \tag{6.7b}$$

$$f_e^\nu \geq 0 \qquad \forall\, e \in E_\nu. \tag{6.7c}$$

Let I_j^ν and O_j^ν denote the set of edges in G_ν whose corresponding input and output bit, respectively, is a 1 (both being subsets of the j-th segment of G_ν), the following constraints relate the codeword bits to the flow variables:

$$x_j^\nu = \sum_{e \in O_j^\nu} f_e^\nu \qquad \text{for } j = 1, \dots, k \text{ and } \nu \in \{a, b\}, \tag{6.8a}$$

$$x_j^s = \sum_{e \in I_j^a} f_e^a \qquad \text{for } j = 1, \dots, k, \tag{6.8b}$$

$$x_{\pi(j)}^s = \sum_{e \in I_j^b} f_e^b \qquad \text{for } j = 1, \dots, k. \tag{6.8c}$$

We can now state TCLPD as

$$\begin{aligned} \min \quad & \sum_{\nu \in \{a,b\}} (\lambda^\nu)^T x^\nu + (\lambda^s)^T x^s \qquad \text{(TCLPD)} \\ \text{s.t.} \quad & \text{(6.7a)–(6.8c) hold,} \end{aligned}$$

where λ is split in the same way as x.

The formulation straightforwardly generalizes to all sorts of "turbo-like" codes, i.e., codes built by convolutional codes plus interleaver conditions. In particular, Feldman and Karger have applied TCLPD to repeat-accumulate (RA(l)) codes [40]. The encoder of an RA(l) repeats the information bits l times, and then sends them to an interleaver followed by an accumulator, which is a two-state convolutional encoder. The authors derive bounds on the error rate of TCLPD for RA codes which were later improved and extended by Halabi and Even [41] as well as by Goldenberg and Burshtein [42].

Note that all x variables in TCLPD are auxiliary: we could replace each occurence by the sum of flow variables defining it. In doing so, (6.8b) and (6.8c) break down to the condition

$$\sum_{e \in I_{\pi(j)}^a} f_e^a = \sum_{e \in I_j^b} f_e^b \qquad \text{for } j = 1, \dots, k. \tag{6.9}$$

Because the rest of the constraints defines a standard network flow, TCLPD models a minimum cost flow problem plus the k additional side constraints (6.9). Using a general purpose LP solver does not exploit this combinatorial substructure. As was suggested already in [1], in [43] Lagrangian relaxation is applied to (6.9) in order to recover the underlying shortest-path problem. Additionally, the authors of [43] use a heuristic based on computing the K shortest paths in a trellis to improve the decoding performance. Via the parameter K the trade-off between algorithmic complexity and error-correcting performance can be controlled.

6.6 Efficient LP Solvers for BLPD

A successful realization of BLPD requires an efficient LP solver. To this end, several ideas have been suggested in the literature. CLPD (cf. Section 6.5) can be considered an efficient LPD approach since the number of variables and constraints are significantly reduced. We review several others in this section.

6.6.1 Solving the Separation Problem

The approach of Taghavi and Siegel [44] tackles the large number of constraints in BLPD2. In their separation approach called adaptive linear programming decoding (ALPD), not all FS inequalities are included in the LP formulation as in BLPD2. Instead, they are iteratively added when needed. As in Definition 6.5, the general idea is to start with a crude LP formulation and then improve it. Note that this idea can also be used to improve the error-correcting performance (see Section 6.7). In the initialization step, the trivial LP $\min\{\lambda^T x \colon x \in [0,1]^n\}$ is solved. Let $(x^*)^k$ be the optimal solution in iteration k. Taghavi and Siegel show that it can be checked in $O(md_c^{\max} + n\log n)$ time if $(x^*)^k$ violates any FS inequality derived from $H_{i,\cdot}x = 0 \pmod 2$ for all $i \in I$ (recall that $m \times n$ is the dimension of H and $d_c^{\max}$ is the maximum maximum check-node degree). This check can be considered as a special case of the greedy separation algorithm (GSA) introduced in [29]. If some of the FS inequalities are violated then these inequalities are added to the formulation and the modified LP is solved again with the new inequalities. ALPD stops if the current optimal solution $(x^*)^k$ satisfies all FS inequalities. If $(x^*)^k$ is integral then it is the ML codeword, otherwise an error is output. ALPD does not yield an improvement in terms of frame error rate since the same solutions are found as in the formulations in the previous section. However, the computational complexity is reduced.

An important algorithmic result of [44] is that ALPD converges to the same optimal solution as BLPD2 with significantly fewer constraints. It is shown empirically that in the last iteration of ALPD, less constraints than in the formulations BLPD2, BLPD3, and CLPD are used. Taghavi and Siegel [44] prove that their algorithm converges to the optimal solution on the fundamental polytope after at most n iterations with at most $n(m+2)$ constraints.

Under the binary-input additive white Gaussian noise channel (BIAWGNC), [44] uses various random (d_v, d_c)-regular codes to test the effect of changing the check node degree, the block length, and the code rate on the number of FS inequalities generated and the convergence of their algorithm. Setting $n = 360$ and rate $R = 1/2$, the authors vary the check node degree in the range of 4 to 40 in their computational testing. It is observed that the average and the maximum number of FS inequalities remain below 270. The effect of changing block length n between 30 and 1920 under $R = 1/2$ is demonstrated on a $(3,6)$-regular LDPC code. For these codes, it is demonstrated that the number of FS inequalities used in the final iteration is generally between $0.6n$ and $0.7n$. Moreover, it is reported that the number of iterations remain below 16. The authors also investigate the effect of the rate on the number of FS inequalities created. Simulations are performed on codes with $n = 120$ and $d_v = 3$ where the number of parity checks m vary between 15 and 90. For most values of m it is observed that the average number of FS inequalities ranges between $1.1m$ and $1.2m$. For ALPD, BLPD2, and SPAD (50 iterations), the average decoding time is testet for $(3,6)$-regular and $(4,8)$-regular LDPC codes with various block lengths. It is shown that ALPD outperforms BLPD with respect to computation time, whil still being slower than SPAD. Furthermore, increasing the check node degree does not increase the computation time of ALPD as much as the computation time of BLPD. The behavior of ALPD, in terms of the number of iterations and the FS inequalities used, under increasing SNR is tested on a $(3,6)$-regular LDPC code with $n = 240$. It is concluded that ALPD performs more iterations and uses more FS inequalities for the instances it fails. Thus, decoding time decreases with increasing SNR.

In [45] ALPD is improved further in terms of complexity. The authors use some structural properties of the fundamental polytope. Let $(x^*)^k$ be an optimal solution in iteration k. In [44] it is shown that, if $(x^*)^k$ does not satisfy an FS inequality derived from check node i, then $(x^*)^k$ satisfies all other FS inequalities derived from i with strict inequality. Based on this result, Taghavi *et al.* [45] modify ALPD and propose the decoding approach we refer to as modified adaptive linear programming decoding (MALPD). In the $(k+1)^{\text{st}}$ iteration of MALPD, it is checked in $O(md_c^{\max})$ time if $(x^*)^k$ violates any FS inequality derived from $H_{i,.}x = 0 \pmod 2$ for some $i \in I$. This check is performed only for those parity checks $i \in I$ which do not induce any active FS inequality at $(x^*)^k$. Moreover, it is proved that inactive FS inequalities at iteration k can be dropped. In any iteration of MALPD, there are at most m FS inequalities. However, the dropped inequalities might be inserted again in a later iteration; therefore the number of iterations for MALPD can be higher than for ALPD.

6.6.2 Message Passing-Like Algorithms

An approach towards low complexity LPD of LDPC codes was proposed by Vontobel and Kötter in [46]. Based on an FFG representation of an LDPC code, they derive an LP, called primal linear programming decoding (PLPD), which is based on BLPD1. The FFG, shown in Figure 6.6, and the Tanner graph are related as follows.

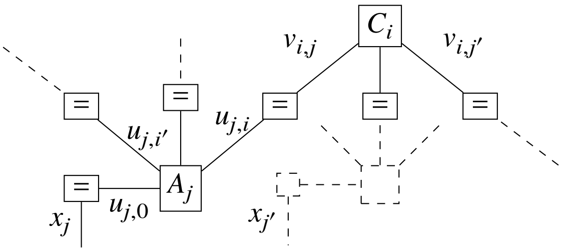


Figure 6.6: A Forney-style factor graph for PLPD.

For each parity check, the FFG exhibits a node C_i which is incident to a variable-edge $v_{i,j}$ for each $j \in N_i$ and demands those adjacent variables to form a configuration that is valid for the local code C_i, i.e., their sum must be even. This corresponds to a check node in the Tanner graph and thus to (6.6b) and (6.6c) except that now there are, for the moment, independent local variables $v_{i,j}$ for each C_i. Additionally, the FFG generalizes the concept of row-wise local codes C_i to the columns of H, in such a way that the j^{th} column is considered a local repetition code A_j that requires the auxiliary variables $u_{j,i}$ for each $i \in N_j \cup \{0\}$ to be either all 1 or all 0. By this, the variable nodes of the Tanner graph are replaced by check nodes A_j—recall that in an FFG all nodes have to be check nodes. There is a third type of factor nodes, labelled by "=", which simply require all incident variables to take on the same value. These are used to establish consistency between the row-wise variables $v_{i,j}$ and the column-wise variables $u_{j,i}$ as well as connecting the codeword variables x_j to the configurations of the A_j.

From this discussion it is easily seen that the FFG indeed ensures that any configuration of the x_j is a valid codeword. The outcome of writing down the constraints for each node and relaxing integrality conditions on all variables is the LP

$$\begin{aligned}
\min \quad & \lambda^T x \qquad \text{(PLPD)} & \\
\text{s.t.} \quad & x_j = u_{j,0} & j = 1, \dots, n, \\
& u_{j,i} = v_{i,j} & \forall (i,j) \in I \times J \colon H_{i,j} = 1, \\
& u_{j,i} = \sum_{S \in A_j, S \ni j} \alpha_{j,S} & \forall i \in N_j,\ j = 1, \dots, n, \\
& \sum_{S \in A_j} \alpha_{j,S} = 1 & \text{for all } j = 1, \dots, n, \\
& v_{i,j} = \sum_{S \in E_i, S \ni j} w_{i,S} & \forall j \in N_i,\ i = 1, \dots, m, \\
& \sum_{S \in E_i} w_{i,S} = 1 & \text{for all } i = 1, \dots, m, \\
& \alpha_{j,S} \geq 0 & \forall S \in A_j,\ j = 1, \dots, n, \\
& w_{i,S} \geq 0 & \forall S \in E_i,\ i = 1, \dots, m,
\end{aligned}$$

where the sets E_i are defined as in BLPD1.

While bloating BLPD1 in this manner seems inefficient at first glance, the reason behind is that the LP dual of PLPD, leads to an FFG which is topologically equivalent to the one of the primal LP, which allows to use the graphical structure for solving the dual. After manipulating constraints of the dual problem to obtain a closely related, "softened" dual linear programming decoding (SDLPD) formulation, the authors propose a coordinate-ascent-type algorithm resembling the min-sum algorithm and show convergence under certain assumptions. In this algorithm, all the edges of FFG are updated according to some schedule. It is shown that the update calculations required during each iteration can be efficiently performed by the SPAD. The coordinate-ascent-type algorithm for SDLPD is guaranteed to converge if all the edges of the FFG are updated cyclically.

Under the BIAWGNC, the authors compare the error-correcting performance of the coordinate-ascent-type algorithm (max iterations: 64, 256) against the performance of the MSAD (max iterations: 64, 256) on the $(3,6)$-regular LDPC code with $n = 1000$ and rate $R = 1/2$. MSAD performs slightly better than the coordinate-ascent-type algorithm. In summary, [46] shows that it is possible to develop LP based algorithms with complexities similar to IMPD.

The convergence and the complexity of the coordinate-ascent-type algorithm proposed in [46] are studied further in [47] by Burshtein. His algorithm has a new scheduling scheme and its convergence rate and computational complexity are analyzed under this scheduling. With this new scheduling scheme, the decoding algorithm from [46] yields an iterative approximate LPD algorithm for LDPC codes with complexity in $O(n)$. The main difference between the two algorithms is the selection and update of edges of the FFG. In [46] all edges are updated cyclically during one iteration, whereas in [47], only few selected edges are updated during one particular iteration. The edges are chosen according to the variable values obtained during previous iterations.

6.6.3 Nonlinear Programming Approach

As an approximation of BLPD for LDPC codes, Yang *et al.* [36] introduce the box constraint quadratic programming decoding (BCQPD) whose time complexity is linear in the code length. BCQPD is a nonlinear programming approach derived from the Lagrangian relaxation (see [7] for an introduction to Lagrangian relaxation) of BLPD1. To achieve BCQPD, a subset of the set of the constraints are incorporated into the objective function. To simplify notation, Yang *et al.* rewrite the constraint blocks (6.6b) and (6.6c) in the general form $Ay = b$ by defining a single variable vector $y = (x, w)^T \in \{0, 1\}^K$ (so K is the total number of variables in BLPD1) and choosing A and b appropriately. Likewise, the objective function coefficients are rewritten in a vector c, wich equals λ followed by the appropriate number of zeros. The resulting formulation is $\min\{c^T y \colon Ay = b, y \in [0, 1]^K\}$. Using a multiplier $\alpha > 0$, the Lagrangian of this problem is

$$\begin{aligned} \min \quad & c^T y + \alpha (Ay - b)^T (Ay - b) \\ \text{s.t.} \quad & 0 \leq y_k \leq 1 \qquad \text{for } k = 1, \dots, K. \end{aligned}$$

If $Ay = b$ is violated then a positive value is added to the original objective function $c^T y$, i.e., the solution y is penalized. Setting $Q = 2\alpha A^T A$ and $r = c - 2\alpha A^T b$ the BCQPD problem

$$\begin{aligned} \min \quad & y^T Q y + 2r^T y \qquad \text{(BCQPD)} \\ \text{s.t.} \quad & 0 \leq y_k \leq 1 \qquad \text{for } k = 1, \dots, K \end{aligned}$$

is obtained. Since Q is a positive semi-definite matrix, i.e., the objective function is convex, and since the set of constraints constitutes a box, each y_k can be minimized separately. This leads to efficient serial and parallel decoding algorithms. Two methods are proposed in [36] to solve the BCQPD problem, the projected successive overrelaxation method (PSORM) and the parallel gradient projection method (PGPM). These methods are generalizations of Gauss-Seidel and Jacobi methods [48] with the benefit of faster convergence if proper weight factors are chosen. PSORM and PGPM benefit from the low-density structure of the underlying parity-check matrix.

One of the disadvantages of IMPD is the difficulty of analyzing the convergence behavior of such algorithms. Yang *et al.* showed both theoretically and empirically that BCQPD converges under some assumptions if PSORM or PGPM is used to solve the quadratic programming problem. Moreover, the complexity of BCQPD is smaller than the complexity of SPAD. For numerical tests, the authors use a product code with block length $4^5 = 1024$ and rate $(3/4)^5 = 0.237$. The BIAWGNC is used. It is observed that the PSORM method converges faster than PGPM. The error-correcting performance of SPAD is poor for product codes due to their regular structure. For the chosen product code, Yang *et al.* demonstrate that PSORM outperforms SPAD in computational complexity as well as in error-correcting performance.

6.6.4 Efficient LPD of SPC Product Codes

The class of single parity-check (SPC) product codes is of special interest in [34]. The authors prove that for SPC product codes the fractional distance is equal to the minimum Hamming distance, which explains why the error-correcting performance of LPD approaches MLD for these codes at high SNR values. Furthermore, they propose a low complexity algorithm which approximately computes the CLPD optimum for SPC product codes. This approach is based on the observation that the parity-check matrix of an SPC product code can be decomposed into component SPC codes. A Lagrangian relaxation of CLPD is obtained by keeping the constraints from only one component code in the formulation and moving all other constraints to the objective function with a penalty vector. The resulting Lagrangian dual problem is solved by subgradient algorithms (see [7]). Two alternatives, subgradient decoding (SD) and joint subgradient decoding (JSD) are proposed. It can be proved that subgradient decoders converge under certain assumptions.

The number of iterations performed against the convergence behavior of SD is tested on the (4,4) SPC product code, which has length $n = 256$, rate $R = (3/4)^4 \approx 0.32$ and is defined as the product of four SPC codes of length 4 each. All variants tested (obtained by keeping the constraints from component code $j = 1, 2, 3, 4$ in the formulation) converge in less than 20 iterations. For demonstrating the error-correcting performance of SD if the number of iterations are set to $5, 10, 20, 100$, the (5,2) SPC product code ($n = 25$, rate $R = (4/5)^2 = 0.64$) is used. The error-correcting performance is improved by increasing the number of iterations. Under the BIAWGNC, this code and the (4,4) SPC product code are used to compare the error-correcting performance of SD and JSD with the performance of BLPD and MLD. It should be noted that for increasing SNR values, the error-correcting performance of BLPD converges to that of MLD for SPC codes. JSD and SD approach the BLPD curve for the code with $n = 25$. For the SPC product code with $n = 256$ the subgradient algorithms perform worse than BLPD. For both codes, the error-correcting performance of JSD is superior to SD. Finally, the $(10, 3)$ SPC product code with $n = 1000$ and rate $R = (9/10)^3 \approx 0.729$ is used to compare the error-correcting performance of SD and JSD with the SPAD. Again the BIAWGNC is used. It is observed that SD performs slightly better than the SPAD with a similar computational complexity. JSD improves the error-correcting performance of the SD at the cost of increased complexity.

6.6.5 Interior Point Algorithms

Efficient LPD approaches based on interior point algorithms are studied by Vontobel [49], Wadayama [50], and Taghavi *et al.* [45]. The use of interior point algorithms to solve LP problems as an alternative to the simplex method was initiated by Karmarkar [51]. In these algorithms, a starting point in the interior of the feasible set is chosen. This starting point is iteratively improved by moving through the interior of the polyhedron in some descent direction until the optimal solution or an approximation is found. There are various interior point algorithms and for some, polynomial time convergence can be proved. This is an advantage over the simplex method which has exponential worst case complexity.

The proposed interior point algorithms aim at using the special structure of the LP problem. The resulting running time is a low-degree polynomial function on the block length. Thus, fast decoding algorithms based on interior point algorithms may be developed for codes with large block lengths. In particular affine scaling algorithms [49], primal-dual interior point algorithms [45, 49] and primal path following interior point algorithm [50] are considered. The bottleneck operation in interior point methods is to solve a system of linear equations depending on the current iteration of the algorithm. Efficient approaches to solve this system of equations are proposed in [45, 49], the former containing an extensive study, including investigation of appropriate preconditioners for the often ill-conditioned equation system. The speed of convergence to the optimal vertex of the algorithms in [50] and [45] under the BIAWGNC are demonstrated on a nearly $(3, 6)$-regular LDPC code with $n = 1008$, $R = 1/2$ and a randomly-generated $(3, 6)$-regular LDPC code with $n = 2000$, respectively.

6.7 Improving the Error-Correcting Performance of BLPD

The error-correcting performance of BLPD can be improved by techniques from integer programming. Most of the improvement techniques can be grouped into cutting plane or branch & bound approaches. In this section, we review the improved LPD approaches mainly with respect to this categorization.

6.7.1 Cutting Plane Approaches

The fundamental polytope $\mathcal{P}$ can be tightened by cutting plane approaches. In the following, we refer to valid inequalities as inequalities satisfied by all points in $\operatorname{conv}(C)$. Valid cuts are valid inequalities which are violated by some non-integral vertex of the LP relaxation. Feldman *et al.* [2] already address this concept; besides applying the “Lift and project” technique which is a generic tightening method for integer programs [52], they also strengthen the relaxation by introducing redundant rows into the parity-check matrix (or, equivalently, redundant parity-checks into the Tanner graph) of the given code (cf. Section 6.2). When using the BLPD2 formulation, we derive additional FS inequalities from the redundant parity-checks without increasing the number of variables. We refer to such inequalities as redundant parity-check (RPC) inequalities. RPC inequalities may include valid cuts which increase the possibility that LPD outputs a codeword. An interesting question relates to the types of inequalities required to describe the codeword polytope $\operatorname{conv}(C)$ exactly. It turns out that $\operatorname{conv}(C)$ cannot be described completely by using only FS and box inequalities; the $(7, 3, 4)$ simplex code (dual of the $(7, 4, 3)$ Hamming code) is given as a counter-example in [2]. More generally, it can be concluded from [53] that these types of inequalities do not suffice to describe all facets of a simplex code.

RPCs can also be interpreted as dual codewords. As such, for interesting codes there are exponentially many RPC inequalities. The RPC inequalities cutting off the non-integral optimal solutions are called RPC cuts [44]. An analytical study under which circumstances RPCs can induce cuts is carried out in [24]. Most notably, it is shown that RPCs obtained by adding no more than $(g-2)/2$ dual codewords, where g is the length of a shortest cycle in the Tanner graph, never change the fundamental polytope.

There are several heuristic approaches in the LPD literature to find cut inducing RPCs [2, 44, 54, 55]. In [2], RPCs which result from adding any two rows of H are appended to the original parity-check matrix. The authors of [45] find RPCs by randomly choosing cycles in the fractional subgraph of the Tanner graph, which is obtained by choosing only the fractional variable nodes and the check nodes directly connected to them. They give a theorem which states that every possible RPC cut must be generated by such a cycle. Their approach is a heuristic one since the converse of that theorem does not hold. In [54] the column index set corresponding to an optimal LP solution is sorted. By re-arranging H and bringing it to row echelon form, RPC cuts are searched. In [55], the parity-check matrix is reformulated such that unit vectors are obtained in the columns of the parity-check matrix which correspond to fractional valued bits in the optimal solution of the current LP. RPC cuts are derived from the rows of the modified parity-check matrix.

The approaches in [28], [44], and [55] rely on a noteworthy structural property of the fundamental ploytope. Namely, it can be shown that no check node of the associated Tanner graph (regardless of the existence of redundant parity-checks) can be adjacent to only one non-integral valued variable node.

Feldman *et al.* [2] test the lift and project technique on a random rate-1/4 LDPC code with $n = 36$, $d_v = 3$ and $d_v = 4$ under the BIAWGNC. Moreover, a random rate-1/4 LDPC code with $n = 40$, $d_v = 3$, and $d_c = 4$ is used to demonstrate the error-correcting performance of BLPD when the original parity-check matrix is extended by all those RPCs obtained by adding any two rows of the original matrix. Both tightening techniques improve the error-correcting performance of BLPD, though the benefit of the latter is rather poor, due to the abovementioned condition on cycle lengths.

The idea of tightening the fundamental polytope is usually implemented as a cutting plane algorithm, i.e., the separation problem is solved (see Definition 6.5 and Section 6.6.1). In cutting plane algorithms, an LP is solved which contains only a subset of the constraints of the corresponding optimization problem. If the optimal LP solution is a codeword then the cutting plane algorithm terminates and outputs the ML codeword. Otherwise, valid cuts from a predetermined family of valid inequalities are searched. If some valid cuts are found, they are added to the LP formulation and the LP is resolved. In [44, 54, 55] the family of valid cuts is FS inequalities derived from RPCs.

In [54] the main motivation for the greedy cutting plane algorithm is to improve the fractional distance. This is demonstrated for the $(7,4,3)$ Hamming code, the $(24,12,8)$ Golay code and a $(204,102)$ LDPC code. As a byproduct under the BSC it is shown on the $(24,12,8)$ Golay code and a $(204,102)$ LDPC code that the RPC based approach of [54] improves the error-correcting performance of BLPD.

In the improved LPD approach of [44], first ALPD (see Section 6.6) is applied. If the solution is non-integral, an RPC cut search algorithm is employed. This algorithm can be briefly outlined as follows:

(1) Given a non-integral optimal LP solution x^*, remove all variable nodes j for which x^*_j is integral from the Tanner graph.

(2) Find a cycle by randomly walking through the pruned Tanner graph.

(3) Sum up (in $\mathbb{F}_2$) the rows H which correspond to the check nodes in the cycle.

(4) Check if the resulting RPC introduces a cut.

The improved decoder of [44] performs noticeably better than BLPD and SPAD. This is shown under the BIAWGNC on $(3, 4)$-regular LDPC codes with $n = 32, 100, 240$.

The cutting plane approach of [55] is based on an IP formulation of MLD, which is referred to as IPD (note that this formulation was already mentioned in [9]). Auxiliary variables $z \in \mathbb{Z}^m$ model the binary constraints $Hx = 0$ over $\mathbb{F}_2$ in the real number field $\mathbb{R}^n$.

$$\begin{aligned} \min \quad & \lambda^T x && \text{(IPD)} \\ \text{s.t.} \quad & Hx - 2z = 0 \\ & x \in \{0, 1\}^n,\ z \in \mathbb{Z}^m \end{aligned}$$

In [55], the LP relaxation of IPD is the initial LP problem which is solved by a cutting plane algorithm. Note that the LP relaxation of IPD is not equivalent to the LP relaxations given in Section 6.5. In almost all improved (in the error-correcting performance sense) LPD approaches reviewed in this article first the BLPD is run. If BLPD fails, some technique to improve BLPD is used with the goal of finding the ML codeword at the cost of increased complexity. In contrast, the approach by Tanatmis *et al.* in [55] does not elaborate on the solution of BLPD, but immediately searches for cuts which can be derived from arbitrary dual codewords. To this end, the parity-check matrix is modified and the conditions under which certain RPCs define cuts are checked. The average number of iterations performed and the average number of cuts generated in the separation algorithm decoding (SAD) of [55] are presented for the $(3, 6)$ random regular codes with $n = 40$, 80, 160, 200, 400 and for the $(31, 10)$, $(63, 39)$, $(127, 99)$, $(255, 223)$ BCH codes. Both performance measures seem to be directly proportional to the block length. The error-correcting performance of SAD is measured on the random regular $(3, 4)$ LDPC codes with block length 100 and 200, and Tanner's $(155, 64)$ group structured LDPC code [56]. It is demonstrated that the improved LPD approach of [55] performs better than BLPD applied in the adaptive setting [44] and better than SPAD. One significant numerical result is that SAD proposed in [55] performs much better than BLPD for the $(63, 39)$ and $(127, 99)$ BCH codes, which have high-density parity check matrices. In all numerical simulations the BIAWGNC is used.

Yufit *et al.* [57] improve SAD [55] and ALPD [44] by employing several techniques. The authors propose to improve the error-correcting performance of these decoding methods by using RPC cuts derived from alternative parity-check matrices selected from the automorphism group of C, $\mathrm{Aut}(C)$. In the alternative parity-check matrices, the columns of the original parity-check matrix are permuted according to some scheme. At the first stage of Algorithm 1 of [57], SAD is used to solve the MLD problem. If the ML codeword is found then Algorithm 1 terminates, otherwise an alternative parity-check matrix from $\mathrm{Aut}(C)$ is randomly chosen and the SAD is applied again. In the worst case this procedure is repeated N times where N denotes a predetermined constant. A similar approach is also used to improve ALPD in Algorithm 2 of [57]. Yufit *et al.* enhance Algorithm 1 with two techniques to improve the error-correcting performance and complexity. The first technique, called parity-check matrix adaptation, is to alter the parity-check matrix prior to decoding such that at the columns of the parity-check matrix which correspond to least reliable bits, i.e., bits with the smallest absolute LLR values, unit vectors are obtained. The second technique, which is motivated by MALPD of [45], is to drop the inactive inequalities at each iteration of SAD, in order to avoid that the problem size increases from iteration to iteration. Under the BIAWGNC, it is demonstrated on the $(63, 36, 11)$ BCH code and the $(63, 39, 9)$ BCH code that SAD can be improved both in terms of error-correcting performance and computational complexity.

6.7.2 Facet Guessing Approaches

Based on BLPD2, Dimakis *et al.* [28] improve the error-correcting performance of BLPD with an approach similar to FDA (see Section 6.4). They introduce facet guessing algorithms which iteratively solve a sequence of related LP problems. Let x^* be a non-integral optimal solution of BLPD, x^{ML} be the ML codeword, and $\mathcal{F}$ be a set of faces of $\mathcal{P}$ which do not contain x^*. This set $\mathcal{F}$ is given by the set of inequalities which are not active at x^*.

The set of active inequalities of a pseudocodeword v is denoted by $\mathbb{A}(v)$. In facet guessing algorithms, the objective function $\lambda^T x$ is minimized over $f \cap \mathcal{P}$ for all $f \in \mathcal{K} \subseteq \mathcal{F}$ where $\mathcal{K}$ is an arbitrary subset of $\mathcal{F}$. The optimal solutions are stored in a list. In random facet guessing decoding (RFGD), $|\mathcal{K}|$ of the faces $f \in \mathcal{F}$ are chosen randomly. If $\mathcal{K} = \mathcal{F}$ then exhaustive facet guessing decoding (EFGD) is obtained. From the list of optimal solutions, the facet guessing algorithms output the integer solution with minimum objective function value. It is shown that EFGD fails if there exists a pseudocodeword $v \in f$ such that $\lambda^T v < \lambda^T x^{\text{ML}}$ for all $f \in \mathbb{A}(x^{\text{ML}})$. For suitable expander codes this result is combined with the following structural property of expander-based codes also proven by the authors. The number of active inequalities at some codeword is much higher than at a non-integral pseudocodeword. Consequently, theoretical bounds on the decoding success conditions of the polynomial time algorithms EFGD and RFGD for expander codes are derived. The numerical experiments are performed under the BIAWGNC, on Tanner's $(155, 64)$ group-structured LDPC code and on a random LDPC code with $n = 200$, $d_v = 3$, $d_c = 4$. For these codes the RFG algorithm performs better than the SPAD.

6.7.3 Branch & Bound Approaches

Linear programming based branch & bound is an implicit enumeration technique in which a difficult optimization problem is divided into multiple, but easier subproblems by fixing the values of certain discrete variables. We refer to [7] for a detailed description. Several authors improved LPD using the branch & bound approach.

Breitbach *et al.* [9] solved IPD by a branch & bound approach. Depth-first and breadth-first search techniques are suggested for exploring the search tree. The authors point out the necessity of finding good bounds in the branch & bound algorithm and suggest a neighborhood search heuristic as a means of computing upper bounds. In the heuristic, a formulation is used which is slightly different to IPD. We refer to this formulation as alternative integer programming decoding (AIPD). AIPD can be obtained by using error vectors. Let $\bar{y} = \frac{1}{2}\left(1 - \operatorname{sign}(\lambda)\right)$ be the hard decision for the LLR vector λ obtained from the BIAWGNC. Comparing $\bar{y} \in \{0,1\}^n$ with a codeword $x \in C$ results in an error vector $e \in \{0,1\}^n$, i.e., $e = \bar{y} + x \pmod 2$. Let $s = H\bar{y}$, and define $\bar{\lambda}$ by $\bar{\lambda}_i = |\lambda_i|$. IPD can be reformulated as

$$\begin{aligned} \min \quad & \bar{\lambda}^T e \qquad \text{(AIPD)} \\ \text{s.t.} \quad & He - 2z = s \\ & e \in \{0,1\}^n,\ z \in \mathbb{Z}^m. \end{aligned}$$

In the neighborhood search heuristic of [9], first a feasible starting solution e^0 is calculated by setting the coordinates of e^0 corresponding to the $n - m$ most reliable bits (i.e., those $j \in J$ such that $|y_j|$ are largest) to 0. These are the non-basic variables while the m basic variables are found from the vector $s \in \{0,1\}^m$. Starting from this solution a neighborhood search is performed by exchanging basic and non-basic variables. The tuple of variables yielding a locally best improvement in the objective function is selected for iterating to the next feasible solution.

In [9], numerical experiments are performed under the BIAWGNC, on the $(31, 21, 5)$ BCH code, the $(64, 42, 8)$ Reed-Muller code, the $(127, 85, 13)$ BCH code and the $(255, 173, 23)$ BCH code. The neighborhood search with single position exchanges performs very similar to MLD for the $(31, 21, 5)$ BCH code. As the block length increases the error-correcting performance of the neighborhood search with single position exchanges gets worse. An extension of this heuristic allowing two position exchanges is applied to the $(64, 42, 8)$ Reed-Muller code, the $(127, 85, 13)$ BCH code, and the $(255, 173, 23)$ BCH code. The extended neighborhood search heuristic improves the error-correcting performance at the cost of increased complexity. A branch & bound algorithm is simulated on the $(31, 21, 5)$ BCH code and different search tree exploration schemes are investigated. The authors suggest a combination of depth-first and breadth-first search.

In [58], Draper *et al.* improve the ALPD approach of [44] with a branch & bound technique. Branching is done on the least certain variable, i.e., x_j such that $\left|x_j^* - 1/2\right|$ is smallest for $j \in J$. Under the BSC, it is observed on Tanner's $(155, 64, 20)$ code that the ML codeword is found after few iterations in many cases.

In [36] two branch & bound approaches for LDPC codes are introduced. In ordered constant depth decoding (OCDD) and ordered variable depth decoding (OVDD), first BLPD1 is solved. If the optimal solution x^* is non-integral, a subset $\mathcal{T} \subseteq \mathcal{E}$ of the set of all non-integral bits $\mathcal{E}$ is chosen. Let $g = |\mathcal{T}|$. The subset $\mathcal{T}$ is constituted from the least certain bits. The term "ordered" in OCDD and OVDD is motivated by this construction. It is experimentally shown in [36] that choosing the least certain bits is advantageous in comparison to a random choice of bits. OVDD is a breadth first branch & bound algorithm where the depth of the search tree is restricted to g. Since this approach is common in integer programming, we do not give the details of OVDD and refer to [7] instead. For OVDD, the number of LPs solved in the worst case is $2^{g+1} - 1$.

In OCDD, m-element subsets $\mathcal{M}$ of $\mathcal{T}$, i.e., $\mathcal{M} \subseteq \mathcal{T}$ and $m = |\mathcal{M}|$, are chosen. Let $b \in \{0,1\}^m$. For any $\mathcal{M} \subseteq \mathcal{T}$, 2^m LPs are solved, each time adding a constraint block

$$x_k = b_k \text{ for all } k \in \mathcal{M}$$

to BLPD1, thus fixing m bits. Let $\hat{x}$ be the solution with the minimum objective function value among the 2^m LPs solved. If $\hat{x}$ is integral, OCDD outputs $\hat{x}$; otherwise another subset $\mathcal{M} \subseteq \mathcal{T}$ is chosen. Since OCDD exhausts all m-element subsets of $\mathcal{T}$, in the worst case $\binom{g}{m}2^m + 1$ LPs are solved.

The branch & bound based improved LPD of Yang *et al.* [36] can be applied to LDPC codes with short block length. For the following numerical tests, the BIAWGNC is used. Under various settings of m and g it is shown on a random LDPC code with $n = 60$, $R = 1/4$, $d_c = 4$, and $d_v = 3$ that OCDD has a better error-correcting performance than BLPD and SPAD. Several simulations are done to analyze the trade-off between complexity and error-correcting performance of OCDD and OVDD. For the test instances and parameter settings[6] used in [36] it has been observed on the above-mentioned code that OVDD outperforms OCDD. This behavior is explained by the observation that OVDD applies the branch & bound approach on the most unreliable bits. On a longer random LDPC code with $n = 1024$, $R = 1/4$, $d_c = 4$, and $d_v = 3$, it is demonstrated that the OVDD performs better than BLPD and SPAD.

Another improved LPD technique which can be interpreted as a branch & bound approach is randomized bit guessing decoding (RBGD) of Dimakis *et al.* [28]. RBGD is inspired from the special case that all facets chosen by RFGD (see Section 6.7.2) correspond to constraints of type $x_j \geq 0$ or $x_j \leq 1$. In RBGD, $k = c \log n$ variables, where $c > 0$ is a constant, are chosen randomly. Because there are 2^k different possibile configurations of these k variables, BLPD2 is run 2^k times with associated constraints for each assignment. The best integer valued solution in terms of the objective function λ is the output of RBGD. Note that by setting k to $c \log n$, a polynomial complexity in n is ensured. Under the assumption that there exists a unique ML codeword, exactly one of the 2^k bit settings matches the bit configuration in the ML codeword. Thus, RBGD fails if a non-integral pseudocodeword with a better objective function value coincides with the ML codeword in all k components. For some expander codes, the probablilty that the RBGD finds the ML codeword is given in [28]. To find this probability expression, the authors first prove that, for some expander-based codes, the number of non-integral components in any pseudocodeword scales linearly in block length.

Chertkov and Chernyak [59] apply the loop calculus approach [60], [61] to improve BLPD. Loop calculus is an approach from statistical physics and related to cycles in the Tanner graph representation of a code. In the context of improved LPD, it is used to either modify objective function coefficients [59] or to find branching rules for branch and bound [62]. Given a parity-check matrix and a channel output, linear programming erasure decoding (LPED)

---

[6] The parameters m and g are chosen such that OVDD and OCDD have similar worst case complexity.

[59] first solves BLPD. If a codeword is found then the algorithm terminates. If a non-integral pseudocodeword is found then a so-called critical loop is searched by employing loop calculus. The indices of the variable nodes along the critical loop form an index set $\mathscr{M} \subseteq J$. LPED lowers the objective function coefficients λ_j of the variables x_j, $j \in \mathscr{M}$, by multiplying λ_j with ϵ, where $0 \leq \epsilon < 1$. After updating the objective function coefficients, BLPD is solved again. If BLPD does not find a codeword then the selection criterion for the critical loop is improved. LPED is tested on the list of pseudocodewords found in [35] for Tanner's $(155, 64, 20)$ code. It is demonstrated that LPED corrects the decoding errors of BLPD for this code.

In [62], Chertkov combines the loop calculus approach used in LPED [59] with RFGD [28]. We refer to the combined algorithm as loop guided guessing decoding (LGGD). LGGD differs from RFGD in the sense that the constraints chosen are of type $x_j \geq 0$ or $x_j \leq 1$ where j is in the index set M, the index set of the variable nodes in the critical loop. LGGD starts with solving BLPD. If the optimal solution is non-integral then the critical loop is found with the loop calculus approach. Next, a variable x_j, $j \in M$, is selected randomly and two partial LPD problems are deduced. These differ from the original problem by only one equality constraint $x_j = 0$ or $x_j = 1$. LGGD chooses the minimum of the objective values of the two subproblems. If the corresponding pseudocodeword is integral then the algorithm terminates. Otherwise the equality constraints are dropped, a new $j \in M$ along the critical loop is chosen, and two new subproblems are constructed. If the set M is exhausted, the selection criterion of the critical loop is improved. LGGD is very similar to OCDD of [36] for the case that $g = |M|$ and $m = 1$. In LGGD branching is done on the bits in the critical loop whereas in OCDD branching is done on the least reliable bits. As in [59], LGGD is tested on the list of pseudocodewords generated in [35] for Tanner's $(155, 64, 20)$ code. It is shown that LGGD improves BLPD under the BIAWGNC.

SAD of [55] is improved in terms of error-correcting performance by a branch & bound approach in [57]. In Algorithm 3 of [57], first SAD is employed. If the solution is non-integral then a depth-first branch & bound is applied. The non-integral valued variable with smallest LLR value is chosen as the branching variable. Algorithm 3 terminates as soon as the search tree reaches the maximally allowed depth D_p. Under the BIAWGNC, on the $(63, 36, 11)$ BCH code and the $(63, 39, 9)$ BCH code Yufit *et al.* [57] demonstrate that the decoding performance of Algorithm 3 (enhanced with parity-check matrix adaptation) approaches MLD.

6.8 Conclusion

In this survey we have shown how the decoding of binary linear block codes benefits from a wide range of concepts which originate from mathematical optimization—mostly linear programming, but also quadratic (nonlinear) and integer programming, duality theory, branch & bound methods, Lagrangian relaxation, network flows, and matroid theory. Bringing together both fields of research does lead to promising new algorithmic decoding approaches as well as deeper structural understanding of linear block codes in general and special classes of codes—like LDPC and turbo-like codes—in particular. The most important reason for the success of this connection is the formulation of MLD as the minimization of a linear function over the codeword polytope conv(C). We have reviewed a variety of techniques of how to approximate this polytope, whose description complexity in general is too large to be computed efficiently.

For further research on LPD of binary linear codes, two general directions can be distinguished. One is to decrease the algorithmic complexity of LPD towards reducing the gap between LPD and IMPD, the latter of which still outperforms LPD in practice. The other direction aims at increasing error-correcting performance, tightening up to MLD performance. This includes a continued study of RPCs as well as the characterization of other, non-RPC facet-defining inequalities of the codeword polytope.

There are other lines of research related to LPD and IMPD which are not covered in this article. Flanagan *et al.* [21] have generalized LP decoding, along with several related concepts, to nonbinary linear codes. Another possible generalization is to extend to different channel models [22]. Connecting two seemingly different decoding approaches, structural relationship between LPD and IMPD has been discussed in [63]. Moreover, the discovery that both decoding methods are closely related to the Bethe free energy approximation, a tool from statistical physics, has initiated vital research [64]. Also, of course, research on IMPD itself, independent of LPD, is still ongoing with high activity. A promising direction of research is certainly the application of message passing techniques to mathematical programming problems beyond LPD [65].

Acknowledgment

We would like to thank Pascal O. Vontobel and Frank Kienle for their comments and suggestions. We also thank the anonymous referees for the helpful reviews.

References

[1] J. Feldman. “Decoding error-correcting codes via linear programming”. PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 2003.

[2] J. Feldman, M. J. Wainwright, and D. R. Karger. “Using linear programming to decode binary linear codes”. *IEEE Transactions on Information Theory* 51.3 (Mar. 2005), pp. 954–972. DOI: 10.1109/TIT.2004.842696. URL: www.eecs.berkeley.edu/~wainwrig/Papers/FelWaiKar05.pdf.

[3] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. “Factor graphs and the sum-product algorithm”. *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 498–519. DOI: 10.1109/18.910572. URL: www.comm.utoronto.ca/frank/papers/KFL01.pdf.

[4] S. M. Aji and R. J. McEliece. “The generalized distributive law”. *IEEE Transactions on Information Theory* 46.2 (Mar. 2000), pp. 325–343. DOI: 10.1109/18.825794.

[5] N. Wiberg. “Codes and decoding on general graphs”. PhD thesis. Linköping, Sweden: Linköping University, 1996.

Table 6.7: List of abbreviations used in Paper I.

AIPD = alternative integer programming decoding
ALPD = adaptive linear programming decoding
BCQPD = box-constrained quadratic programming decoding
BIAWGNC = binary-input additive white Gaussian noise channel
BLPD = bare linear programming decoding
BSC = binary symmetric channel
CLPD = cascaded linear programming decoding
EFGD = exhaustive facet guessing decoding
FDA = fractional distance algorithm
FFG = Forney style factor graph
FS = forbidden set
GSA = greedy separation algorithm
IMPD = iterative message-passing decoding
IP = integer programming
IPD = integer programming decoding
JSD = joint subgradient decoding
LLR = log-likelihood ratio
LDPC = low-density parity-check
LGGD = loop guided guessing decoding
LP = linear programming
LPD = linear programming decoding
LPED = linear programming erasure decoding
MALPD = modified adaptive linear programming decoding
ML = maximum likelihood
MLD = maximum-likelihood decoding
MSAD = min-sum algorithm decoding
OCDD = ordered constant depth decoding
OVDD = ordered variable depth decoding
PGPM = parallel gradient projection method
PLPD = primal linear programming decoding
PSORM = projected successive overrelaxation method
RA = repeat accumulate
RBGD = randomized bit guessing decoding
RFGD = randomized facet guessing decoding
RPC = redundant parity-check
SAD = separation algorithm decoding
SD = subgradient decoding
SDLPD = softened dual linear programming decoding
SNR = signal-to-noise ratio
SPAD = sum-product algorithm decoding
SPC = single parity-check
TCLPD = turbo code linear programming decoding

[6] H.-A. Loeliger. “An introduction to factor graphs”. *IEEE Signal Processing Magazine* 21.1 (Jan. 2004), pp. 28–41. ISSN: 1053-5888. DOI: 10.1109/MSP.2004.1267047.

[7] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience series in discrete mathematics and optimization, John Wiley & Sons, 1988.

[8] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* 2nd ed. Algorithms and Combinatorics. Springer, 1993.

[9] M. Breitbach et al. “Soft-decision decoding of linear block codes as optimization problem”. *European Transactions on Telecommunications* 9 (1998), pp. 289–293.

[10] E. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. “On the inherent intractability of certain coding problems”. *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 954–972. DOI: 10.1109/TIT.1978.1055873.

[11] R. M. Karp. “Reducibility among combinatorial problems”. In: *Complexity of Computer Computations.* Ed. by R. E. Miller, J. W. Thatcher, and J. D. Bohlinger. The IBM Research Symposia Series. Springer US, 1972, pp. 85–103. ISBN: 978-1-4684-2003-6. DOI: 10.1007/978-1-4684-2001-2_9.

[12] A. Vardy. “The intractability of computing the minimum distance of a code”. *IEEE Transactions on Information Theory* 43.6 (Nov. 1997), pp. 1757–1766. DOI: 10.1109/18.641542.

[13] D. J. A. Welsh. “Combinatorial problems in matroid theory”. In: *Combinatorial Mathematics and its Applications.* Ed. by D. J. A. Welsh. London, U.K.: Academic Press, 1971, pp. 291–307.

[14] J. G. Oxley. *Matroid Theory.* Oxford University Press, 1992.

[15] D. J. A. Welsh. *Matroid Theory.* L. M. S. Monographs. Academic Press, 1976.

[16] N. Kashyap. “A decomposition theory for binary linear codes”. *IEEE Transactions on Information Theory* 54.7 (July 2008), pp. 3035–3058. DOI: 10.1109/TIT.2008.924700. URL: http://www.ece.iisc.ernet.in/~nkashyap/Papers/code_decomp_final.pdf.

[17] F. Barahona and M. Grötschel. “On the cycle polytope of a binary matroid”. *Journal of Combinatorial Theory Series B* 40 (1986), pp. 40–62.

[18] N. Kashyap. “On the convex geometry of binary linear codes”. In: *Proceedings of the Inaugural UC San Diego Workshop on Information Theory and its Applications (ITA).* La Jolla, CA, Feb. 2006. URL: http://ita.ucsd.edu/workshop/06/talks.

[19] M. Grötschel and K. Truemper. “Decomposition and optimization over cycles in binary matroids”. *Journal of Combinatorial Theory Series B* 46 (1989), pp. 306–337.

[20] P. D. Seymour. “Decomposition of regular matroids”. *Journal of Combinatorial Theory Series B* 28 (1980), pp. 305–359.

[21] M. Flanagan et al. “Linear-programming decoding of nonbinary linear codes”. *IEEE Transactions on Information Theory* 55.9 (Sept. 2009), pp. 4134–4154. ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2025571. arXiv: 0804.4384 `[cs.IT]`.

[22] A. Cohen et al. “LP decoding for joint source-channel codes and for the non-ergodic Polya channel”. *IEEE Communications Letters* 12.9 (2008), pp. 678–680. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2008.080713.

[23] S. Lin and D. Costello Jr. *Error Control Coding*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2004. ISBN: 0130426725.

[24] P. O. Vontobel and R. Koetter. *Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes*. 2005. arXiv: cs/0512078 `[cs.IT]`.

[25] G. D. Forney Jr. et al. “On the effective weights of pseudocodewords for codes defined on graphs with cycles”. In: *Codes, systems, and graphical models*. Ed. by B. Marcus and J. Rosenthal. Vol. 123. The IMA Volumes in Mathematics and its Applications. Springer Verlag, New York, Inc., 2001, pp. 101–112.

[26] M. Chertkov and M. G. Stepanov. “Polytope of correct (linear programming) decoding and low-weight pseudo-codewords”. In: *Proceedings of IEEE International Symposium on Information Theory*. St. Petersburg, Russia, July 2011, pp. 1648–1652. DOI: 10.1109/ISIT.2011.6033824.

[27] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[28] A. G. Dimakis, A. A. Gohari, and M. J. Wainwright. “Guessing facets: polytope structure and improved LP decoding”. *IEEE Transactions on Information Theory* 55.8 (Aug. 2009), pp. 4134–4154. DOI: 10.1109/TIT.2009.2023735. arXiv: 0709.3915 `[cs.IT]`.

[29] M. Grötschel. “Cardinality homogeneous set systems, cycles in matroids, and associated polytopes”. In: *The Sharpest Cut: The Impact of Manfred Padberg and His Work*. MPS-SIAM Series on Optimization. Society for Industrial Mathematics, 2004. Chap. 8, pp. 99–120.

[30] R. G. Gallager. “Low-density parity-check codes”. *IRE Transactions on Information Theory* 8.1 (Jan. 1962), pp. 21–28. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057683.

[31] G. D. Forney Jr. “Codes on graphs: normal realizations”. *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 529–548. DOI: 10.1109/18.910573.

[32] J. Feldman and C. Stein. “LP decoding achieves capacity”. In: *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, Jan. 2005, pp. 460–469. ISBN: 0-89871-585-7.

[33] M. Yannakakis. “Expressing combinatorial optimization problems by linear programs”. *Journal of Computer and System Sciences* 43 (1991), pp. 441–466. DOI: 10.1145/62212.62232.

[34] K. Yang, X. Wang, and J. Feldman. “A new linear programming approach to decoding linear block codes”. *IEEE Transactions on Information Theory* 54.3 (Mar. 2008), pp. 1061–1072. DOI: 10.1109/TIT.2007.915712.

[35] M. Chertkov and M. Stepanov. “Pseudo-codeword landscape”. In: *Proceedings of IEEE International Symposium on Information Theory*. Nice, France, June 2007, pp. 1546–1550. DOI: 10.1109/ISIT.2007.4557442.

[36] K. Yang, J. Feldman, and X. Wang. “Nonlinear programming approaches to decoding low-density parity-check codes”. *IEEE Journal on Selected Areas in Communications* 24.8 (Aug. 2006), pp. 1603–1613. DOI: 10.1109/JSAC.2006.879405.

[37] C. Berrou and A. Glavieux. "Near optimum error correcting coding and decoding: turbo-codes". *IEEE Transactions on Communications* 44.10 (Oct. 1996), pp. 1261–1271. ISSN: 0090-6778. DOI: 10.1109/26.539767.

[38] C. Berrou et al. "Improving the distance properties of turbo codes using a third component code: 3D turbo codes". *IEEE Transactions on Communications* 57.9 (Sept. 2009), pp. 2505–2509. DOI: 10.1109/TCOMM.2009.09.070521.

[39] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[40] J. Feldman and D. R. Karger. "Decoding turbo-like codes via linear programming". *Journal of Computer and System Sciences* 68 (4 June 2004), pp. 733–752. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2003.11.005.

[41] N. Halabi and G. Even. "Improved bounds on the word error probability of RA(2) codes with linear-programming-based decoding". *IEEE Transactions on Information Theory* 51.1 (Jan. 2005), pp. 265–280. DOI: 10.1109/TIT.2004.839509.

[42] I. Goldenberg and D. Burshtein. "Error bounds for repeat-accumulate codes decoded via linear programming". In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing*. Brest, France, Sept. 2010, pp. 487–491.

[43] A. Tanatmis, S. Ruzika, and F. Kienle. "A Lagrangian relaxation based decoding algorithm for LTE turbo codes". In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing*. Brest, France, Sept. 2010, pp. 369–373. DOI: 10.1109/ISTC.2010.5613906.

[44] M. H. Taghavi and P. H. Siegel. "Adaptive methods for linear programming decoding". *IEEE Transactions on Information Theory* 54.12 (Dec. 2008), pp. 5396–5410. DOI: 10.1109/TIT.2008.2006384. arXiv: cs/0703123 `[cs.IT]`.

[45] M. H. Taghavi, A. Shokrollahi, and P. H. Siegel. "Efficient implementation of linear programming decoding". *IEEE Transactions on Information Theory* 57.9 (Sept. 2011), pp. 5960–5982. DOI: 10.1109/TIT.2011.2161920. arXiv: 0902.0657 `[cs.IT]`.

[46] P. O. Vontobel and R. Koetter. "On low-complexity linear-programming decoding of LDPC codes". *European Transactions on Telecommunications* 18 (2007), pp. 509–517.

[47] D. Burshtein. "Iterative approximate linear programming decoding of LDPC codes with linear complexity". *IEEE Transactions on Information Theory* 55.11 (2009), pp. 4835–4859. ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2030477.

[48] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Belmont, Massachusetts: Athena Scientific, 1997.

[49] P. O. Vontobel. "Interior-point algorithms for linear-programming decoding". In: *Proceedings of the IEEE Information Theory Workshop*. UC San Diego. La Jolla, CA, Jan. 2008, pp. 433–437.

[50] T. Wadayama. "An LP decoding algorithm based on primal path-following interior point method". In: *Proceedings of IEEE International Symposium on Information Theory*. Seoul, Korea, June 2009, pp. 389–393. DOI: 10.1109/ISIT.2009.5205741.

[51] N. Karmarkar. “A new polynomial-time algorithm for linear programming”. *Combinatorica* 4.4 (1984), pp. 373–396.

[52] L. Lovász and A. Schrijver. “Cones of matrices and set-functions and 0-1 optimization”. *SIAM Journal on Optimization* 1.2 (1991), pp. 166–190.

[53] M. Grötschel and K. Truemper. “Master polytopes for cycles in binary matroids”. *Linear Algebra and its Applications* 114/115 (1989), pp. 523–540.

[54] M. Miwa, T. Wadayama, and I. Takumi. “A cutting-plane method based on redundant rows for improving fractional distance”. *IEEE Journal on Selected Areas in Communications* 27.6 (Aug. 2009), pp. 1005–1012. ISSN: 0733-8716. DOI: 10.1109/JSAC.2009.090818.

[55] A. Tanatmis et al. “A separation algorithm for improved LP-decoding of linear block codes”. *IEEE Transactions on Information Theory* 56.7 (July 2010), pp. 3277–3289. ISSN: 0018-9448. DOI: 10.1109/TIT.2010.2048489.

[56] R. M. Tanner et al. “LDPC block and convolutional codes based on circulant matrices”. *IEEE Transactions on Information Theory* 50.12 (2004), pp. 2966–2984. DOI: 10.1109/TIT.2004.838370.

[57] A. Yufit, A. Lifshitz, and Y. Be’ery. “Efficient linear programming decoding of HDPC codes”. *IEEE Transactions on Communications* 59.3 (Mar. 2011), pp. 758–766. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2011.122110.090729.

[58] S. C. Draper, J. S. Yedidia, and Y. Wang. “ML decoding via mixed-integer adaptive linear programming”. In: *Proceedings of IEEE International Symposium on Information Theory.* Nice, France, June 2007, pp. 1656–1660. DOI: 10.1109/ISIT.2007.4557459.

[59] M. Chertkov and V. Y. Chernyak. “Loop calculus helps to improve belief propagation and linear programming decodings of low-density-parity-check codes”. In: *Proceedings of the 44th Annual Allerton Conference on Communication, Control and Computing.* Monticello, IL, Sept. 2006. arXiv: cs/0609154 `[cs.IT]`.

[60] M. Chertkov and V. Y. Chernyak. “Loop calculus in statistical physics and information science”. *Physical Review E* 73.6 (June 2006), p. 065102. DOI: 10.1103/PhysRevE.73.065102. arXiv: cond-mat/0601487 `[cond-mat.stat-mech]`.

[61] M. Chertkov and V. Y. Chernyak. “Loop series for discrete statistical models on graphs”. *Journal of Statistical Mechanics: Theory and Experiment* 2006 (2006), P06009. DOI: 10.1088/1742-5468/2006/06/P06009. arXiv: cond-mat/0603189 `[cond-mat.stat-mech]`.

[62] M. Chertkov. “Reducing the error floor”. In: *Proceedings of the IEEE Information Theory Workshop.* Tahoe City, CA, Sept. 2007, pp. 230–235. DOI: 10.1109/ITW.2007.4313079.

[63] P. O. Vontobel and R. Koetter. “On the relationship between linear programming decoding and min-sum algorithm decoding”. In: *Proceedings of the International Symposium on Information Theory and Applications (ISITA).* Parma, Italy, Oct. 2004, pp. 991–996.

[64] J. S. Yedidia, W. T. Freeman, and Y. Weiss. “Constructing free-energy approximations and generalized belief propagation algorithms”. *IEEE Transactions on Information Theory* 51.7 (2005), pp. 2282–2312. DOI: 10.1109/TIT.2005.850085.

[65] M. Bayati, D. Shah, and M. Sharma. “Max-product for maximum weight matching: convergence, correctness, and LP duality”. *IEEE Transactions on Information Theory* 54.3 (2008), pp. 1241–1251. DOI: 10.1109/TIT.2007.915695.

Chapter 7

Paper II: ML vs. BP Decoding of Binary and Non-Binary LDPC Codes

Stefan Scholl, Frank Kienle, Michael Helmling, and Stefan Ruzika

This chapter is a reformatted and revised version of the following publication that was presented at the ISTC conference and appeared in the refereed conference proceedings:

S. Scholl, F. Kienle, M. Helmling, and S. Ruzika. "ML vs. BP decoding of binary and non-binary LDPC codes". In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing*. Gothenburg, Sweden, Aug. 2012, pp. 71–75. DOI: 10.1109/ISTC.2012.6325201

ML vs. BP Decoding of Binary and Non-Binary LDPC Codes

Stefan Scholl
Frank Kienle

Michael Helmling
Stefan Rukiza

It has been shown that non-binary LDPC codes exhibit a better error correction performance than binary codes for short block lengths. However, this advantage was up to now only shown under belief-propagation decoding. To gain new insights, we investigate binary and non-binary codes under ML decoding. Our analysis includes different modulation schemes and decoding algorithms. For ML decoding under different modulation schemes, a flexible integer programming formulation is presented. In this paper, we show that short non-binary LDPC codes are not necessarily superior to binary codes with respect to ML decoding. The decoding gain observed under BP decoding originates mainly in the more powerful non-binary decoding algorithm.

7.1 Introduction

Forward error correction is a vital part of digital communication systems. LDPC codes [1] have been adopted in many communication standards, e.g. WiMAX [2], and show an error-correction capability near the Shannon limit for long block lengths. However, for short block lengths there is still a gap in communications performance to the theoretical limit.

There exist both binary and non-binary LDPC codes. For short block lengths, non-binary LDPC codes show a superior performance. The gain of non-binary codes exhibits especially under higher-order modulation, when the information in the receiver can be fully processed on symbol level. These advantages of non-binary LDPC codes have been shown by the DAVINCI project [3], but solely under BP decoding.

When considering a communication scheme based on LDPC channel codes, we have different orthogonal options respecting:

- *channel code*: binary LDPC codes or non-binary LDPC codes,
- *modulation*: BPSK or higher order modulation with bit-level demodulator or symbol-level demodulator,
- *decoding algorithm*: suboptimal heuristics (binary/non-binary BP) or optimal decoding (ML).

These three aspects can be combined together and form a large design space. Not all combinations have been investigated in literature yet, like ML decoding under higher-order modulation. In this paper we analyze each of the schemes with respect to their communications performance. An overview over all considered cases is given in Table 7.1.

In general, the communications performance of a channel coding system is determined by two important factors: *(a)* the *channel code* (code structure) used at the transmitter, and *(b)* the applied *decoding algorithm* at the receiver.

Under BP decoding it is not possible to distinguish between the performance contribution of the code structure and the contribution of the decoding algorithm. Up to now, simulations for non-binary codes have only been presented under suboptimal BP decoding. To investigate the performance loss of these suboptimal BP algorithms, we utilize an ML decoder to derive ML bounds on the decoding performance.

We use an integer programming (IP) formulation to tackle the ML decoding problem. A key feature of the utilized IP formulation is its high flexibility to cover all transmission cases using ML decoding (see Table 7.1). We merge two already known IP formulations for binary and non-binary codes to obtain a new IP formulation that can handle various types of channel codes and modulation schemes. This IP problem is solved optimally and thus exactly solves the ML problem.

The task of this paper is to explore the different transmission schemes with respect to performance loss of the suboptimal BP and its low-complexity variant for non-binary codes,

decoding algorithm	BPSK	64-QAM bit demod.	64-QAM symbol demod.
binary channel codes			
iterative BP (suboptimal)	binary BP	binary BP	
ML (optimal)	IP decoder [4]	IP decoder [4]	our new IP
non-binary channel codes			
iterative BP (suboptimal)	non-binary BP	non-binary BP	non-binary BP
ML (optimal)	IP decoder [4] on binary image	IP decoder [4] on binary image	our new IP
low-complexity decoding [5]	our new IP or modified BP		

Table 7.1: Design space for transmission systems based on LDPC codes.

performance gain of symbol-level demodulation, and comparison of the binary and non-binary code structure.

7.2 Binary and Non-Binary LDPC Codes

In this section we present the used transmission scheme, the decoding algorithms and the code constructions.

Transmission Scheme

Throughout this paper we assume the following transmission scheme. A code C is defined by its parity-check matrix $\mathbf{H}$ of dimension $(N-K) \times N$, which has elements from a Galois field $\mathrm{GF}(q = 2^m) = \{\alpha_0, \alpha_1, \dots, \alpha_{q-1}\}$ as entries. The sent codewords are denoted by $\mathbf{x} = (x_0, x_1, \dots, x_{N-1})$, $x_i \in \mathrm{GF}(q)$.

For a non-binary code $(q > 2)$ the parity-check matrix can alternatively be described by its binary image, a binary $(N_\mathrm{b} - K_\mathrm{b}) \times N_\mathrm{b}$ matrix $\mathbf{H}_\mathrm{b}$ [6] (with $N_\mathrm{b} = Nm$ and $K_\mathrm{b} = Km$). Accordingly, a non-binary codeword can also be expressed as binary vector $\mathbf{x}_\mathrm{b} = (x_{\mathrm{b},0}, \dots, x_{\mathrm{b},N_\mathrm{b}})$ In case of binary codes we have $\mathbf{x} = \mathbf{x}_\mathrm{b}$ and $\mathbf{H} = \mathbf{H}_\mathrm{b}$.

The modulator groups Q codebits of $\mathbf{x}_\mathrm{b}$ and maps them onto one complex modulation symbol $s_i \in \Sigma$. Here, Σ denotes the modulation alphabet of size 2^Q. In case of 64-QAM modulation, Gray mapping is used. For $\mathbf{x} \in C$, denote by $\mathbf{s}(\mathbf{x}) = (s(\mathbf{x})_0, \dots, s(\mathbf{x})_{N_\mathrm{s}-1}) \in \Sigma^{N_\mathrm{s}}$ the vector of transmitted modulation symbols, where $N_\mathrm{s} = N_\mathrm{b}/Q$ is the number of symbols per codeword. A transmission system using a binary code with 64-QAM is called bit-interleaved coded modulation (BICM).

We assume an AWGN channel with noise variance σ^2 that outputs the received vector $\mathbf{y}$.

Regarding the demodulator, we have different possibilities: BPSK demodulation, 64-QAM demodulation on bit level and 64-QAM demodulation on symbol level.

The bit-level demodulators (BPSK and 64-QAM bit-level demodulation) calculate a log-likelihood ratio (LLR) λ for each code bit $x_{\mathrm{b},i}$, namely

$$\lambda(x_{\mathrm{b},i} \mid y_j) = \ln \frac{\sum_{s(\mathbf{x})_j:\ x_{\mathrm{b},i}=0} P(y_j \mid s(\mathbf{x})_j)}{\sum_{s(\mathbf{x})_j:\ x_{\mathrm{b},i}=1} P(y_j \mid s(\mathbf{x})_j)}$$

which reduces to $\lambda_i = \frac{2}{\sigma^2} y_i$ in case of BPSK demodulation.

The symbol demodulator for 64-QAM calculates a log-density ratio (LDR) for each sent symbol. Each LDR consists of q reliabilities $\lambda(k)$, $\alpha_k \in \mathrm{GF}(q)$. The demodulation on symbol level can be described by

$$\lambda_i(k) = \ln\left(\frac{P(y_i \mid s_i = s_k)}{P(y_i \mid s_i = s_0)}\right) = -\frac{1}{2\sigma^2}\left\|y_i - s_k\right\|_2^2 + \frac{1}{2\sigma^2}\left\|y_i - s_0\right\|_2^2 .$$

Throughout this paper we use BPSK or 64-QAM modulation and codes over GF(2) and GF(64).

Belief-Propagation Decoding

LDPC codes can be efficiently decoded with the well-known belief propagation or sum-product algorithm. The general algorithm for codes over $\mathrm{GF}(q)$ is shortly reviewed in the following.

The BP for LDPC codes [7] operates on the Tanner graph of the parity-check matrix $\mathbf{H}$ consisting of variable nodes (corresponding to the code bits or symbols) and check nodes (corresponding to the parity-check constraints). The variable nodes and check nodes are connected by edges and iteratively exchange probabilistic messages over these edges. The number of exchanged messages per edge is equal to $q - 1$. For further details, the reader is referred to [7].

Decoders for binary LDPC codes can be efficiently implemented in hardware. Non-binary BP decoders are far more complex, especially because of the need to store all exchanged messages in memory.

One possibility to overcome these large memory requirements is to restrict the number of exchanged messages to a maximum of n_{m} per edge ($n_{\mathrm{m}} \ll q$). This approach is used in the extended min-sum (EMS) algorithm [5]. Choosing the value of n_{m} carefully can considerably reduce the memory requirements without introducing significant extra loss in communications performance.

$$H^{\text{Macro}} = \begin{array}{c}\left(\begin{array}{cccccccccc|cccc|cccccccccc}
1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0&1&1
\end{array}\right) \\
\underbrace{\hphantom{0000000000000000000}}_{\text{information nodes}}\underbrace{\hphantom{00000000}}_{\text{hidden nodes}}\underbrace{\hphantom{0000000000000000000}}_{\text{parity nodes}}
\end{array}
\begin{array}{l}
\left.\begin{array}{c}\\ \\ \\ \\ \end{array}\right\}\text{CN layers to protect degree-1 VNs}\\
\\
\\
\left.\begin{array}{c}\\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\}\text{CN layers to initialize hidden VNs}\\
\\
\end{array}$$

Figure 7.2: Macro matrix of a multi-edge type LDPC code of rate $R = 1/2$.

Code Construction

For a fair comparison of binary and non-binary LDPC codes it is crucial to use state-of-the-art codes. The codes are designed for proper BP performance in the convergence region. We use multi-edge-type codes [8] as binary LDPC codes and row-optimized non-binary LDPC codes [6]. In the following, the code constructions are described.

We use a so-called macro matrix to describe the code structure of multi-edge type LDPC codes. This kind of structure, which can efficiently be implemented in hardware and is utilized in many communication systems (e.g. [2]) is the common way to represent LDPC codes.

A parity-check matrix is obtained by lifting the macro matrix. Lifting describes the process of establishing shifted identity $z \times z$ matrices at the positions of ones in the macro matrix. A zero indicates an all-zero matrix of size $z \times z$ in the parity-check matrix. The corresponding cyclic shift entries are derived by a progressive edge-growth technique [9].

One showcase macro matrix for a rate $R = 1/2$ multi-edge type code is shown in Figure 7.2. Note that the hidden nodes themselves require an elaborated connectivity structure. Hidden nodes are initialized with a zero LLR; thus, we have to avoid stopping sets for the belief propagation algorithm during code design. Special CN layers are utilized, which have to ensure that the hidden nodes are initialized with values within the first decoding iteration.

The non-binary LDPC codes are regular (2,4) quasi-cyclic codes over GF(64). We have constructed these codes using the graphs by Declercq [10] as macro matrices. To obtain larger graphs of appropriate length, we again use a lifting process utilizing a progressive edge-growth

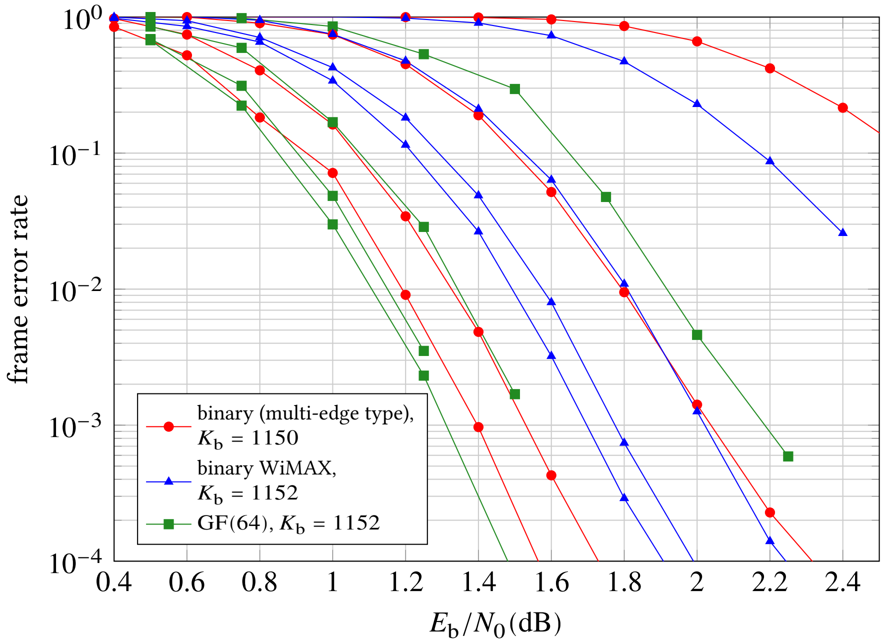


Figure 7.3: BP decoding of three similar binary and non-binary LDPC codes, $N_b \approx 2300$. For each code, the figure shows performance curves with 5, 10, 20 and 40 iterations (less iterations correspond to higher frame error rate).

technique [9]. The non-binary matrix entries are chosen such as to optimize the local minimum distance of the rows [6]. This optimization ensures state-of-the-art performance in the convergence region.

With the presented code constructions, we designed a binary multi-edge type code and a non-binary code over GF(64) that competes with a WiMAX LDPC code. Simulation results under BP decoding are presented in Figure 7.3. The multi-edge type code and the non-binary code obtain a significant gain over the WiMAX code. However, the gain of the non-binary code over the multi-edge type code is negligible for the considered block length of about 2300 bits.

For our further investigations we restrict to much shorter block lengths ($N = 96$), because non-binary codes are stronger for shorter codes and codes of this size can be tackled with the IP-based ML decoder.

7.3 IP-based ML Decoding

ML decoding is an NP-complete problem, and for most block codes of practical size no efficient ML decoding algorithms exist. However, by using an IP formulation, the ML problem for short codes can be tackled with commercially available IP solvers.

Our goal is an ML performance analysis based on IP decoding for the different transmission schemes shown in Table 7.1. We need an IP formulation that is capable of decoding binary and non-binary codes under bit-level and symbol-level demodulation.

In this section, we present a new flexible IP formulation that can be used for all considered transmission schemes. We combine two known approaches, merging the binary IP formulation from [4] with a symbol-level input stage from [11]. This fully flexible formulation can be applied to all transmission schemes in Table 7.1, covering binary and non-binary codes as well as higher-order modulation.

The details of the utilized formulation are presented in the following. For each received symbol y_j and each $s_k \in \Sigma \setminus \{s_0\}$, we introduce a binary decision variable ξ_j^k acting as an indicator that $s(\mathbf{x})_j = s_k$ (the case $s(\mathbf{x})_j = s_0$ is covered by the configuration that all $\xi_j^k = 0$, $k = 1, \dots, q-1$). We assume that s_0 is the symbol which $\mathbf{0}$ is mapped to by the Gray mapping. The constraints

$$\sum_{k=1}^{q-1} \xi_j^k \leq 1 \quad \text{for } j = 0, \dots, N_\mathrm{s} - 1 \tag{7.1}$$

ensure that the decoder decides for exactly one symbol for each transmission. The Gray code used in 64-QAM maps each binary Q-tuple $\mathbf{b} = (b_0, \dots, b_{Q-1})$ to one symbol $g(\mathbf{b}) \in \Sigma$. For $l = 0, \dots, Q-1$, let $\bar{g}(l) = \left\{s_k \in \Sigma : \left(g^{-1}(s_k)\right)_l = 1\right\}$ be the set of channel symbols for which the l-th corresponding bit is 1. With this notation, we can relate the decision variables of the binary code, $\mathbf{x}_\mathrm{b} = (x_{\mathrm{b},i}, \dots, x_{\mathrm{b},N_\mathrm{b}-1})$ to the channel symbols ξ_j^k: Let $i = j_i \cdot m + l_i$ be the division with remainder of i by m, then the constraint for $x_{\mathrm{b},i}$ is

$$x_{\mathrm{b},i} = \sum_{s_k \in \bar{g}(l_i)} \xi_{j_i}^k. \tag{7.2}$$

For the underlying binary code we employ the compact formulation

$$\mathbf{H}_\mathrm{b}\mathbf{x}_\mathrm{b} - 2\mathbf{z} = \mathbf{0} \tag{7.3}$$

with auxiliary variables $\mathbf{z} \in \mathbb{Z}^{N_\mathrm{b}-K_\mathrm{b}}$ [4]. This completes the constraint set; as for the the objective function, a linear expression can be derived immediately from the ML condition:

$$\mathbf{x}_\mathrm{ML} = \arg\max_{\mathbf{x} \in C} P(\mathbf{y} \mid \mathbf{s}(\mathbf{x})) = \arg\max_{\mathbf{x} \in C} \sum_{j=0}^{N_\mathrm{s}-1} \sum_{k=1}^{q-1} \xi_j^k \lambda_j(k)$$

Now we have all the necessary ingredients at hand to state the complete IP:

$$
\begin{array}{lll}
\max & \displaystyle\sum_{j=0}^{N_\mathrm{s}-1}\sum_{k=1}^{q-1} \xi_j^k \lambda_j(k) & \\
\text{s.t.} & \displaystyle\sum_{k=1}^{q-1} \xi_j^k \le 1 & \text{for } j = 0,\dots,N_\mathrm{s}-1 \\
& x_{\mathrm{b},i} = \displaystyle\sum_{s_k \in \bar{g}(l_i)} \xi_{j_i}^k & \text{for } i = 0,\dots,N_\mathrm{b}-1 \\
& \mathbf{H}_\mathrm{b}\mathbf{x}_\mathrm{b} - 2\mathbf{z} = \mathbf{0} & \\
& \xi \in \{0,1\}^{q-1\times N_\mathrm{s}} & \\
& \mathbf{x} \in \{0,1\}^{N_\mathrm{b}},\ \mathbf{z} \in \mathbb{Z}^{N_\mathrm{b}-K_\mathrm{b}} &
\end{array}
$$

7.4 Results

In this chapter, we present simulation results for binary and non-binary LDPC codes. We compare codes with similar code parameters under BP and ML decoding, focussing on the convergence region. As example codes we use a binary multi-edge type code ($K = 50$) and a non-binary code over GF(64) ($K_\mathrm{b} = 48$). For this very short block length the macro matrix (Figure 7.2) of the multi-edge type code is slightly modified to optimize the minimum distance. All codes are of rate $R = 1/2$. The BP decoders run with 40 iterations of layered decoding.

In the following we consider three different cases:

(1) Binary and non-binary BP: comparison with respect to ML performance.

(2) Low-complexity decoding: influence of input message truncation.

(3) Higher-order modulation (64-QAM): comparison of demodulation on symbol level and bit level.

Case 1: Binary and Non-Binary BP

In the first case, we compare binary and non-binary LDPC codes under BPSK modulation. Figure 7.4 shows the simulation results for ML and BP decoding. Using BP decoding the non-binary code outperforms the binary one by 0.5 dB.

Furthermore, as a byproduct we show the performance degradation of the suboptimal BP for LDPC codes. For short block length this degradation is on the order of 1 dB for binary LDPC codes and 0.5 dB for codes over GF(64).

By utilizing an optimal ML decoder we can distinguish between the gain introduced by the non-binary code structure and the non-binary decoding algorithm. Comparing the ML performance (Figure 7.4) reveals that the non-binary code structure is not superior to the binary one. The

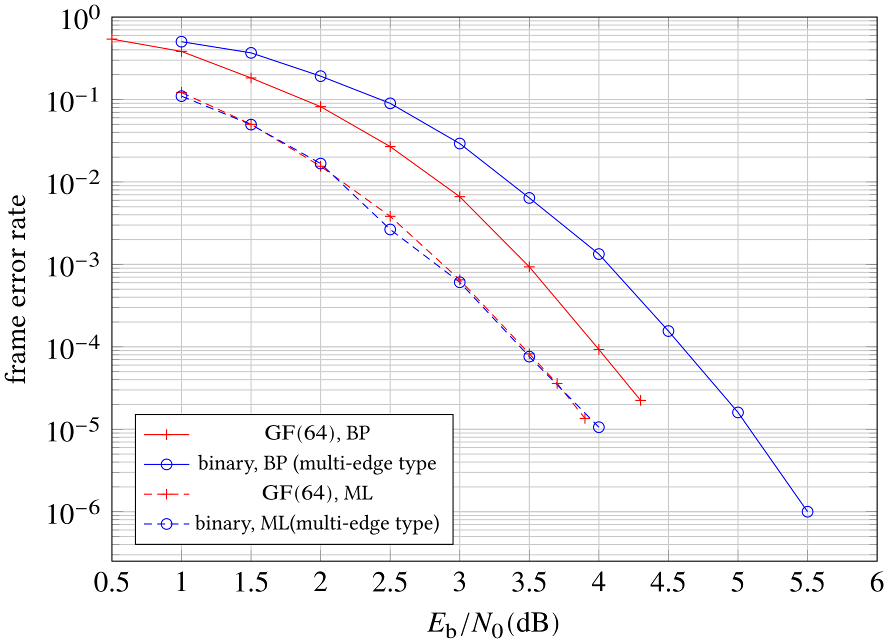


Figure 7.4: Binary vs. non-binary LDPC codes under ML and BP decoding using BPSK modulation.

gain of the non-binary system merely arises from the more powerful but also more complex non-binary BP. This fact poses the question whether it is a good idea to use the high-complexity non-binary BP to improve the error-correction performance in new communication systems, or to rather spend some extra effort (and complexity) for improved decoding of binary LDPC codes. Various different approaches for improved binary BP algorithms can be found in literature, e.g. [12].

However, low-complexity versions of the non-binary BP have to be analyzed and compared to possible new binary decoders with improved decoding performance.

Case 2: Low-Complexity Algorithm

In the second case, we present an analysis of low-complexity EMS decoding [5]. The EMS decoding algorithm is a suboptimal BP that operates with truncated messages (LDRs). This truncation reduces the high memory requirements in hardware implementations [13]. Remind, however, that this truncation introduces a loss in communications performance.

In our case study we analyze the influence of the message truncation at the decoder input. After the input message truncation we use an ML or an unmodified BP decoder. The results are lower bounds for the performance of the EMS and similar message truncation algorithms.

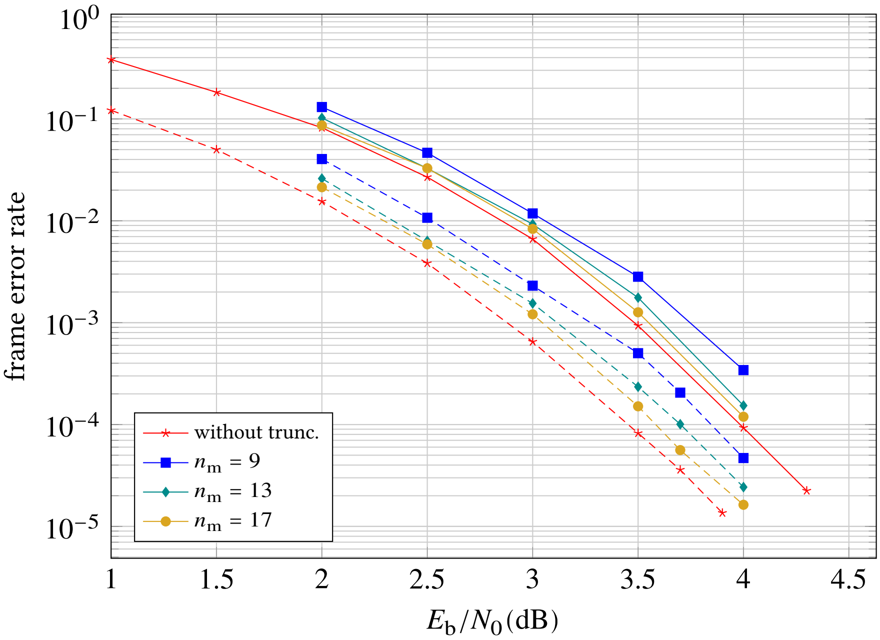


Figure 7.5: Low-complexity decoding: performance loss using truncated messages at the decoder input. The solid lines show the performance of the EMS algorithm, while the dashed ones correspond to ML decoding.

Figure 7.5 shows the simulation results for different numbers of used messages n_m.If we process only 17 out of 64 messages, the memory is greatly reduced and the additional loss is less than 0.1 dB using BP decoding. For the ML decoder the loss is slightly higher. If we use only 9 out of 64 input messages for the BP the complexity reduction is even larger, but more loss in error-correction performance is introduced. However, even with $n_m = 9$ the non-binary code still outperforms the binary code by a few tenths dB.

Case 3: Higher-Order Modulation (64-QAM)

In the third case, we consider higher order modulation (64-QAM). We present a comparison between bit-level and symbol-level demodulators in conjunction with codes over GF(64).

One advantage of non-binary LDPC codes over GF(64) is that a 64-QAM modulator can map the code symbols directly onto modulation symbols. On the receiver side, a symbol-level demodulator and a non-binary decoder can be used to fully process the information on symbol level, resulting in joint demodulation and decoding. This advantage is expected to give additional gain.

We measure the performance gain of the symbol-level demodulator over the bit-level demodulator for the short GF(64) code. The results for decoding under BP and ML are given in Figure 7.6.

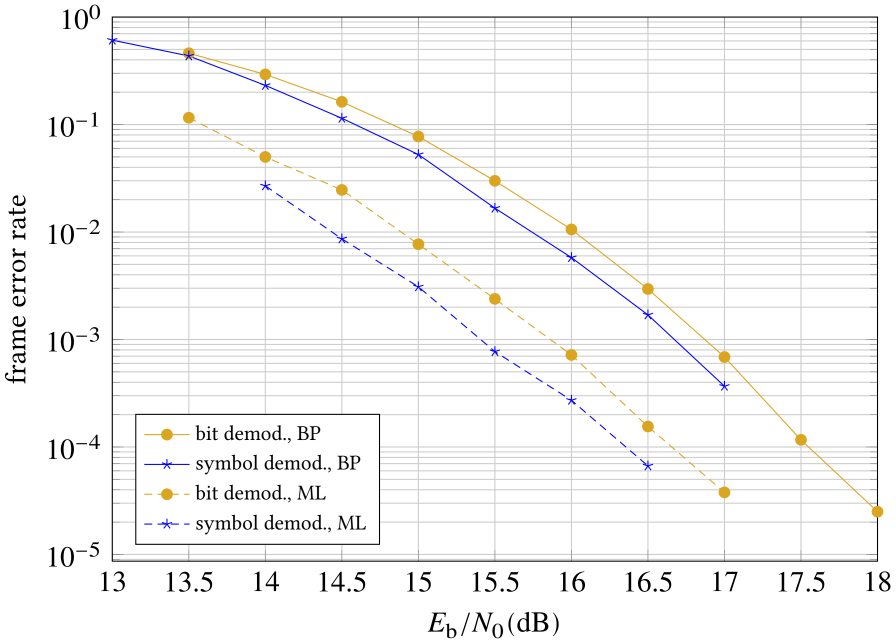


Figure 7.6: 64-QAM: bit-level demodulation vs. symbol-level demodulation.

Under BP decoding the performance gain of the symbol-level demodulator is approximately 0.2 dB. Under ML decoding, the gain is slightly larger.

This simulations show the superior performance of symbol-level demodulation for non-binary LDPC codes. In terms of hardware complexity, the symbol-level demodulator needs to calculate 64 distances for each received symbol. However, the EMS decoding algorithm greatly reduces demodulation complexity, because only n_m distances need to be computed.

Now, we repeat the analysis for a binary code and BICM. With our new IP formulation it is possible to determine the ML performance of the binary code under symbol-level demodulation. Simulation results are presented in Figure 7.7. The symbol-level demodulation provides a gain of a few tenths dB. Additionally, we want to point out that the binary BP looses 2 dB performance with respect to the ML decoder.

7.5 Conclusion

In this paper we investigated binary and non-binary LDPC codes under different modulation schemes and decoding algorithms. Especially the new ML decoding results led to new insights in the decoding performance of non-binary LDPC codes. We have shown that the advantage of non-binary LDPC codes is mainly due to their non-binary decoding algorithm. Furthermore, we analyzed low-complexity decoding by message truncation and showed the superior performance of symbol-level demodulation for higher-order modulation.

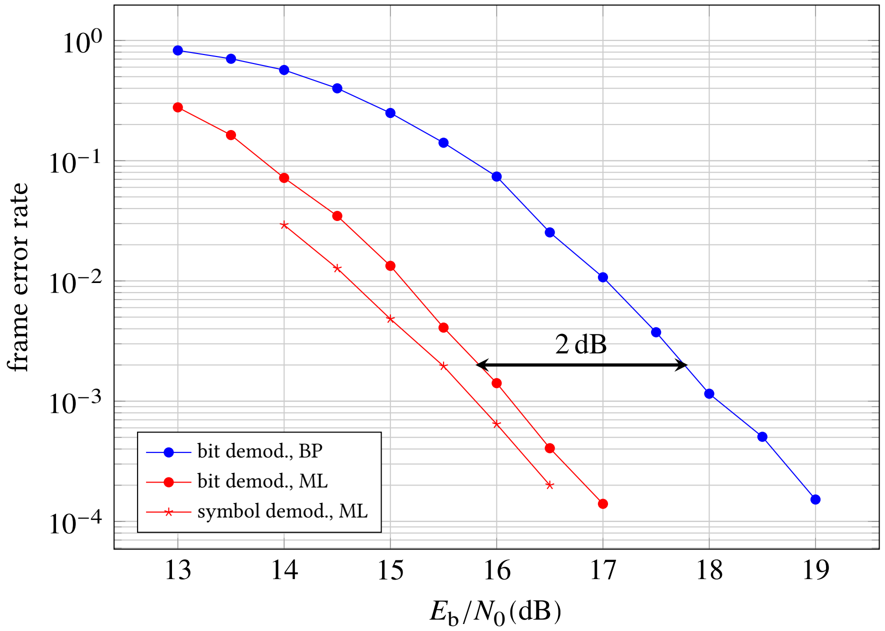


Figure 7.7: BICM: bit-level demodulation vs. symbol-level demodulation for a binary LDPC code with $K_b = 48$.

Acknowledgment

We thank Norbert Wehn and Horst W. Hamacher for their valuable comments and suggestions. We gratefully acknowledge financial support by the Center of Mathematical and Computational Modelling of the University of Kaiserslautern.

References

[1] R. G. Gallager. *Low-Density Parity-Check Codes.* Cambridge, Massachusetts: M.I.T. Press, 1963.

[2] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1.* Feb. 2006. DOI: 10.1109/IEEESTD.2006.99107.

[3] S. Pfletschinger et al. "Performance evaluation of non-binary LDPC codes on wireless channels". In: *Proc. ICT Mobile Summit.* Santander, Spain, June 2009.

[4] M. Breitbach et al. "Soft-decision decoding of linear block codes as optimization problem". *European Transactions on Telecommunications* 9 (1998), pp. 289–293.

[5] A. Voicila et al. “Low-complexity, low-memory EMS algorithm for non-binary LDPC codes”. In: *IEEE International Conference on Communications*. Glasgow, UK, June 2007, pp. 671–676. DOI: 10.1109/ICC.2007.115.

[6] C. Poulliat, M. Fossorier, and D. Declercq. “Design of regular $(2,\mathrm{d}_\mathrm{c})$-LDPC codes over GF($q$) using their binary images”. *IEEE Transactions on Communications* 56.10 (Oct. 2008), pp. 1626–1635. DOI: 10.1109/TCOMM.2008.060527.

[7] D. Declercq and M. Fossorier. “Decoding algorithms for nonbinary LDPC codes over GF(q)”. *IEEE Transactions on Communications* 55.4 (Apr. 2007), pp. 633–643. DOI: 10.1109/TCOMM.2007.894088.

[8] T. J. Richardson and R. L. Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008. ISBN: 978-0-521-85229-6.

[9] X.-Y. Hu, E. Eleftheriou, and D. Arnold. “Regular and irregular progressive edge-growth tanner graphs”. *IEEE Transactions on Information Theory* 51.1 (Jan. 2005), pp. 386–398. DOI: 10.1109/TIT.2004.839541.

[10] D. Declercq. *Sparse Graphs and non-binary LDPC codes Database*. 2011. URL: http://perso-etis.ensea.fr/~declercq/graphs.php.

[11] M. F. Flanagan. “A unified framework for linear-programming based communication receivers”. *IEEE Transactions on Communications* 59.12 (2011), pp. 3375–3387. DOI: 10.1109/TCOMM.2011.100411.100417.

[12] N. Varnica, M. P. C. Fossorier, and A. Kavcic. “Augmented belief propagation decoding of low-density parity check codes”. *IEEE Transactions on Communications* 55.7 (July 2007), pp. 1308–1317. DOI: 10.1109/TCOMM.2007.900611.

[13] T. Lehnigk-Emden and N. Wehn. “Complexity evaluation of non-binary Galois field LDPC code decoders”. In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing*. Brest, France, Sept. 2010, pp. 53–57. DOI: 10.1109/ISTC.2010.5613874.

Chapter 8

Paper III: Integer Programming as a Tool for Analysis of Channel Codes

Stefan Scholl, Frank Kienle, Michael Helmling, and Stefan Ruzika

This chapter is a reformatted and revised version of the following publication that was presented at the SCC conference and appeared in the refereed conference proceedings:

S. Scholl, F. Kienle, M. Helmling, and S. Ruzika. "Integer programming as a tool for analysis of channel codes". In: *Proceedings of the 9th international ITG conference on Systems, Communications and Coding*. Munich, Germany, Jan. 2013, pp. 1–6

Integer Programming as a Tool for Analysis of Channel Codes

Stefan Scholl
Frank Kienle

Michael Helmling
Stefan Rukiza

Linear and integer programming have recently gained interest as new approaches for decoding channel codes. In this paper, we present a framework for the analysis of arbitrary linear block codes based on integer programming. We review how to analyze ML decoding performance and minimum distance. It is shown that integer programming offers an efficient way for ML decoding. Frame error rates for ML decoding of various codes with block lengths of several hundred bits are simulated. Furthermore, we introduce new formulations for weight distributions and reliability-based decoding heuristics, like Chase and ordered statistics decoding. New simulation results for WiMAX LDPC and LTE turbo codes under ML decoding are also shown.

8.1 Introduction

Forward error correction is an essential part of modern digital communication systems. Different channel codes are used in today's communications applications, like LDPC codes, turbo codes, BCH and Reed-Solomon (RS) codes.

In hardware implementations of channel decoders, usually suboptimal decoding heuristics are used that exhibit a low hardware complexity. LDPC codes [1] are decoded by the sum-product algorithm (SPA) [2] based on the Tanner-graph representation [3]. For turbo codes, the turbo decoding scheme based on two convolutional decoders plus an interleaver is utilized [4]. For soft decoding of BCH and RS codes reliability-based heuristics are emerging [5, 6].

The optimal decoding strategy is maximum-likelihood (ML) decoding. However, it is very difficult to carry out ML decoding on block codes of practical interest because of the high computational complexity. For many codes from communication standards, even the performance of ML decoding in terms of frame error rate (FER) is unknown.

Another approach to the decoding problem is mathematical optimization theory. The decoding problem is then formulated as an integer program (IP) or its relaxation as a linear program (LP). Recently, the optimization-based approach has led to new insights in channel coding and to new decoding algorithms. In this paper, we show how mathematical optimization can be used to analyze channel codes. We present a framework, based on IP formulations, which allows us to analyze:

- the ML decoding performance,
- minimum-distance properties and the error floor, and
- reliability-based decoding heuristics.

After introducing the notation and common decoding algorithms in Section 8.2, we present a short introduction into IP in Section 8.3. Different techniques to solve IP problems, like branch-and-bound, cutting-plane algorithms and branch-and-cut, are reviewed.

In Section 8.4 we show IP formulations for analyzing general linear block codes. First, two IP formulations for ML decoding are reviewed. This approach allows for ML decoding simulations of WiMAX LDPC [7] and LTE turbo codes [8] of reasonable length (up to 1000 bits). Moreover, we present two new IP formulations for modelling reliability-based decoding algorithms: Chase decoding [5] and ordered-statistics decoding (OSD) [6]. A third IP formulation deals with the code's minimum-distance properties and allows for error-floor estimation using the weight distribution and the union bound [9].

In Section 8.5 we present an IP formulation dedicated to turbo codes based on network flows. It takes into account the special trellis structure of the turbo codes and has proven to be more efficient than the model for general linear block codes.

Simulation results for ML decoding, the union bound and reliability-based decoding can be found in Section 8.6.

8.2 Preliminaries

In this section we introduce the terminology and briefly describe the widely used decoding algorithms for linear block codes.

8.2.1 Transmission Scheme

Throughout this paper we assume the following transmission scheme: A code C is defined by its parity-check matrix $\mathbf{H}$ of dimension $(N-K) \times N$. The code rate is given by $R = K/N$. A codeword is denoted by $\mathbf{x} = (x_0, x_1, \dots, x_{N-1}), x_i \in \{0, 1\}$ and satisfies $\mathbf{Hx} = \mathbf{0} \pmod 2$. During transmission, the bits are modulated using BPSK and sent over an additive white Gaussian noise (AWGN) channel that outputs the received log-likelihood ratios (LLRs) $\mathbf{y} = (y_0, y_1, \dots, y_{N-1})$. Let $\bar{\mathbf{y}}$ be the hard decision vector of $\mathbf{y}$, then the transmission can be interpreted as an addition of a binary error vector $\mathbf{e}$, such that $\bar{\mathbf{y}} = \mathbf{x} + \mathbf{e}$. The syndrome vector is denoted by $\mathbf{s} = \mathbf{H}\bar{\mathbf{y}} = \mathbf{He}$.

8.2.2 Maximum-Likelihood Decoding

A straightforward ML decoder compares the received LLRs to all code words of the used channel code. Thus 2^K metrics have to be calculated. Since 2^K is very large for codes of practical length, this approach is very time consuming.

8.2.3 Belief-Propagation Decoding

LDPC codes can be decoded efficiently but suboptimally by the well-known SPA [2]. The SPA for LDPC codes works on the Tanner-graph representation of the parity-check matrix $\mathbf{H}$. In the Tanner graph, probabilistic messages are exchanged iteratively between the variable nodes and the check nodes.

Decoders for LDPC codes based on the suboptimal iterative SPA can be implemented efficiently in hardware, but introduce a loss in communications performance due to cycles in the Tanner graph. This problem has a higher impact for codes of smaller block length and higher code rate.

8.2.4 Turbo Decoding

Turbo codes are usually decoded with the well-known turbo decoding algorithm. It basically consists of two convolutional decoders with soft output stages that are connected by an interleaver. The two convolutional decoders iteratively exchange probabilistic messages [4]. Turbo decoders based on this iterative heuristic can efficiently be implemented in hardware. However, they also introduce a loss in communications performance.

8.2.5 Reliability-Based Decoding

Reliability-based decoding algorithms are alternative decoding algorithms for linear block codes. They do not demand a sparse parity-check matrix, so they are suitable for soft-decoding of BCH and RS codes as well. The two main representatives are the Chase algorithm [5] and OSD [6].

In a Chase decoder, at first the l least reliable bit positions (LRP) are determined. All configurations of these bits are enumerated and an algebraic hard decision decoder is applied to each configuration, thereby forming a list of codeword candidates. A list decoding procedure then selects the most probably sent codeword.

OSD is based on a re-encoding procedure and is applicable for arbitrary linear block codes. First, the most reliable independent bits positions (MRIP) have to be found. The MRIPs are treated as information bits and are re-encoded using a modified parity-check or generator matrix to obtain a codeword. To improve decoding performance, this process is repeated several times, but with some of the MRIPs flipped. The process of flipping all possible combinations of l bits and applying re-encoding is called order-l reprocessing. A list decoder selects the most probably sent codeword.

8.3 Fundamentals of Solving IP Problems

In this section, we introduce the general framework of integer programming models and recap some solution strategies.

Let $\mathbf{c} \in \mathbb{Q}^n$, $\mathbf{b} \in \mathbb{Q}^m$, and $\mathbf{A} \in \mathbb{Q}^{m \times n}$ be given. Let $\mathbf{x} \in \mathbb{R}^n$ denote the vector of variables. Integer programming is the mathematical discipline which deals with the optimization of a linear *objective function* $\mathbf{c}^T\mathbf{x}$ over the *feasible set*

$$P_I := P \cap \mathbb{Z}^n := \{\mathbf{x} \colon \mathbf{A}\mathbf{x} = \mathbf{b}\} \cap \mathbb{Z}^n = \{\mathbf{x} \in \mathbb{Z}^n \colon \mathbf{A}\mathbf{x} = \mathbf{b}\}.$$

A general integer programming problem can thus be written as

$$\begin{aligned} \text{min or max} \quad & \mathbf{c}^T\mathbf{x} \\ \text{s. t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n. \end{aligned}$$

Without loss of generality, we may consider minimization problems only. In contrast to *linear programming problems*, solutions are required to be integral. General IP problems—as well as many special cases—are NP-hard. However, due to extensive studies of theoretical properties, the development of sophisticated methods, as well as increasing computational capabilities, IP proved to be very useful to model and solve many real-world optimization problems (e. g. production planning, scheduling, routing problems, ...).

IP problems can be solved in a naïve way by *explicitly enumerating* all possible values of the variable vector $\mathbf{x}$ and choosing the one that yields the minimal objective function value. Though correct, this procedure is guaranteed to be exponential in the number of components of $\mathbf{x}$. To avoid excessive computational effort, a theory inducing more efficient algorithms was developed which relies on techniques like *implicit enumeration*, *relaxation and bounds*, and the usage of *problem-specific structural properties*. In the following, we will highlight some basic concepts while an in-depth exposition to this field can be found in [10].

Branch-and-bound is a wide-spread technique realizing the concept of divide-and-conquer in the context of IP: for each $i \in \{0, \dots, n-1\}$ and $v \in \mathbb{Z}$, an optimal solution $\mathbf{x}^*$ either satisfies $x_i^* \leq v$ or $x_i^* \geq v + 1$. These two constraints induce two subproblems (each possessing a smaller feasible set) and, for at least one of them, x^* is optimal. Iterative application of this *branching step* yields IP problems of manageable size. For each (sub)problem, primal and dual bounds are obtained by relaxation techniques and heuristics. They allow to prune branches of the search tree, thus reducing the search area (*implicit enumeration*). More advanced topics like variable fixing, warm-start techniques, or the computation of bounds contribute to the efficiency of the procedure.

Cutting-plane algorithms rely on the idea of solving the IP problem as the linear programming problem

$$\min\{\mathbf{c}^T\mathbf{x} \colon \mathbf{x} \in \operatorname{conv}(P_I)\}.$$

Yet, the convex hull of the feasible set of the IP problem is in general hard to compute explicitly. Therefore, approaches are developed which iteratively solve the LP relaxation of the IP and compute a *cutting plane*, i.e., a valid inequality separating the feasible set P_I from the optimal solution of the LP relaxation. These cuts are added to the formulation in all subsequent iterations. Important questions address the convergence of this procedure, polyhedral properties of the polyhedra of interest, as well as the generation of strong valid inequalities.

A mixture of strategies such as *branch-and-cut* and the utilization of problem-intrinsic structure might lead to enhanced solution strategies. Implementing an efficient IP problem solver is certainly a demanding task. For a special purpose solver, the problem has to be thoroughly understood, a suitable algorithm has to be chosen and realized efficiently. But there also exists a bandwidth of all-purpose solvers, both open-source (like GLPK, COIN-OR) and commercial (e.g. CPLEX, Gurobi), which may be sufficient to solve various different IP problems such as the ones mentioned in this article.

8.4 IP Formulations for Linear Block Codes

In this section, several IP formulations are presented that cover different aspects of code analysis. All formulations are general and can therefore be applied to arbitrary linear block codes. In the first part of this section we review two closely related IP formulations for the ML decoding problem. In the second part we present two new IP formulations that model Chase decoding and OSD. In the third part, minimum distance properties are investigated.

8.4.1 ML Decoding

The ML decoding problem can be formulated as an integer linear optimization problem in two ways as presented in [11] and [12] as follows:

$$\begin{array}{llcll} \min & \sum_{i=0}^{N-1} y_i x_i & \qquad & \min & \sum_{i=0}^{N-1} |y_i|\, e_i \\ \text{s.t.} & \mathbf{Hx} - 2\mathbf{z} = \mathbf{0} & & \text{s.t.} & \mathbf{He} - 2\mathbf{z} = \mathbf{s} \\ & \mathbf{x} \in \{0,1\}^N,\ \mathbf{z} \in \mathbb{Z}^{N-K} & & & \mathbf{e} \in \{0,1\}^N,\ \mathbf{z} \in \mathbb{Z}^{N-K}. \end{array} \tag{8.1}$$

The formulation on the left relies on the parity-check equation, whereas the one on the right relies on the syndrome and the error vector e. Both formulations model the ML decoding problem exactly and are thus equivalent.

The variables $\mathbf{x}$ model bits of the codeword, while $\mathbf{z}$ is a vector of artificial variables to account for the modulo-2 arithmetic. Interestingly, these formulations prove to be efficient enough such that a general purpose IP solver can tackle the problem for codes of practical interest. For Monte-Carlo simulation of the FER performance, this IP has to be solved millions of times, once for every simulated frame.

We are able to run ML simulations for LDPC codes, turbo codes, BCH codes and the binary image [13] of Reed-Solomon codes with the same IP formulation.

8.4.2 Chase Decoding and OSD

With some modifications, IP formulations can model not only the ML decoding problem, but also some decoding heuristics. We introduce two new formulations, one for Chase decoding and one for OSD, that take into account the reliability based decoding. The key element of the new IP formulations are additional constraints which ensure that reliable bits and unreliable bits are treated basically different.

Chase decoding allows all errors at the least reliable positions to be corrected, but only t errors at the most reliable positions (MRP) (t is the number of bits the algebraic decoder can correct). The set of indices containing the MRP is denoted by I_{MRP}. Chase decoding is modeled as

$$\begin{array}{ll} \min & \sum_{i=0}^{N-1} |y_i|\, e_i \\ \text{s.t.} & \mathbf{He} - 2\mathbf{z} = \mathbf{s} \\ & \sum_{i \in I_{\text{MRP}}} e_i \leq t \\ & \mathbf{e} \in \{0,1\}^N, \mathbf{z} \in \mathbb{Z}^{N-K}. \end{array}$$

OSD decoding corrects l errors at the most reliable independent positions (MRIP) and all errors at the remaining low reliable positions. The set of indices containing the MRIP is denoted by I_{MRIP}. OSD decoding is modeled by the IP

$$\begin{aligned} \min \quad & \sum_{i=0}^{N-1} |y_i|\, e_i \\ \text{s.t.} \quad & \mathbf{He} - 2\mathbf{z} = \mathbf{s} \\ & \sum_{i \in I_{\mathrm{MRIP}}} e_i \le l \\ & \mathbf{e} \in \{0,1\}^N, \mathbf{z} \in \mathbb{Z}^{N-K}. \end{aligned}$$

Note that these formulations induce tighter feasibility polyhedra compared to those in (8.1) due to the additional constraints. It can thus be expected that the related problems are also easier to solve.

8.4.3 IP Formulation for Weight-Profile Analysis

IP formulations are not only able to analyze the FER performance of channel codes and their decoding algorithms. Other interesting code parameters like minimum distance and the weight distribution $\{A_1, A_2, \dots, A_n\}$ can be investigated using IP.

In [14] an IP formulation was presented to calculate the minimum distance. In order to obtain the weight distribution coefficients, we slightly modify this formulation by replacing the weight constraint. The modified formulation for the weight distribution looks for valid codewords $\mathbf{x}$ with weight w. The number of solutions for a specific weight w equals the weight distribution coefficient A_w.

The two IP formulations for minimum-distance computation (left) and weight-distribution analysis (right) are as follows:

$$\begin{aligned} \min \quad & \sum_{i=0}^{N-1} x_i & \qquad \min \quad & 0 \\ \text{s.t.} \quad & \mathbf{Hx} - 2\mathbf{z} = \mathbf{0} & \qquad \text{s.t.} \quad & \mathbf{Hx} - 2\mathbf{z} = \mathbf{0} \\ & \sum_{i=0}^{N-1} x_i \ge 1 & & \sum_{i=0}^{N-1} x_i = w \\ & \mathbf{x} \in \{0,1\}^N,\ \mathbf{z} \in \mathbb{Z}^{N-K} & & \mathbf{x} \in \{0,1\}^N,\ \mathbf{z} \in \mathbb{Z}^{N-K}. \end{aligned} \tag{8.2}$$

where with the latter formulation we instruct the IP solver to output *all* optimal solutions. Note that in contrast to other minimum-distance calculation methods, this approach is applicable to arbitrary linear block codes.

8.5 Specialized IP Formulation for Turbo Codes

Up to now, we have presented IP formulations that are general and can be applied to every linear block code. However, the decoding problem can be modeled more efficiently if special code structures are taken into account. In this section, we present a dedicated formulation for ML decoding of turbo codes based on network flows.

The formulation is based on trellis graphs G_ν of the constituent encoders C^ν, $\nu \in \{a, b\}$, and the fact that ML decoding amounts to the determination of a shortest path through both trellises which respects certain equality constraints in order to guarantee consistency with the interleaver.

The trellis graphs of the constituent encoders C^ν are modeled by flow conservation and capacity constraints [15], along with side constraints appropriately connecting the flow variables $\mathbf{f}^\nu$ ($\nu \in \{a, b\}$) to auxiliary variables $\mathbf{x}^s$ and $\mathbf{x}^\nu$, respectively, which embody the codeword bits.

For $\nu \in \{a, b\}$, let $G_\nu = (S_\nu, E_\nu)$ be the trellis according to C_ν, where S_ν is the index set of nodes (states) and E_ν is the set of edges (state transitions) e in G_ν. For $s \in S$ let $\mathrm{out}(s)$ and $\mathrm{in}(s)$ denote the sets of outgoing and inbound edges, respectively, of s. Let $s^{\mathrm{start},\nu}$ and $s^{\mathrm{end},\nu}$ denote the unique start and end node, respectively, of G_ν. We can then define a feasible flow $\mathbf{f}^\nu$ in the trellis G_ν by the system

$$\sum_{e \in \mathrm{out}(s^{\mathrm{start},\nu})} f_e^\nu = 1, \tag{8.3a}$$

$$\sum_{e \in \mathrm{in}(s^{\mathrm{end},\nu})} f_e^\nu = 1, \tag{8.3b}$$

$$\sum_{e \in \mathrm{out}(s)} f_e^\nu = \sum_{e \in \mathrm{in}(s)} f_e^\nu \qquad \text{for all } s \in S_\nu \setminus \{s^{\mathrm{start},\nu}, s^{\mathrm{end},\nu}\}, \tag{8.3c}$$

$$f_e^\nu \in \{0, 1\} \qquad \text{for all } e \in E_\nu. \tag{8.3d}$$

The integrality conditions (8.3d) ensure that the solution is actually a path, i. e., no fractional flow values can occur. For the related problem of LP decoding, these constraints are relaxed to $0 \leq f_e^\nu \leq 1$.

Let I_j^ν and O_j^ν denote the set of edges in G_ν whose corresponding input and output bit, respectively, is a 1 (both being subsets of the j-th segment of G_ν), the following constraints relate the bits of the codeword $\mathbf{x} = (\mathbf{x}^s, \mathbf{x}^a, \mathbf{x}^b)$ to the flow variables:

$$x_j^\nu = \sum_{e \in O_j^\nu} f_e^\nu \qquad \text{for } j = 1, \dots, K \text{ and } \nu \in \{a, b\}, \tag{8.4a}$$

$$x_j^s = \sum_{e \in I_j^a} f_e^a \qquad \text{for } j = 1, \dots, K, \tag{8.4b}$$

$$x_{\pi(j)}^s = \sum_{e \in I_j^b} f_e^b \qquad \text{for } j = 1, \dots, K. \tag{8.4c}$$

We can now state the network-flow based turbo code formulation (TC-NFP) as

$$\begin{aligned} \min \quad & \sum_{v \in \{a,b\}} (y^v)^T x^v + (y^s)^T x^s \qquad \text{(TC-NFP)} \\ \text{s.t.} \quad & \text{(8.3)–(8.4) hold.} \end{aligned}$$

where the LLR vector $\mathbf{y}$ is split in the same way as $\mathbf{x}$.

Since all x variables in TC-NFP are auxiliary, we can replace each occurrence by the sum of flow variables defining it. In doing so, (8.4b) and (8.4c) break down to the condition

$$\sum_{e \in I^a_{\pi(j)}} f^a_e = \sum_{e \in I^b_j} f^b_e \qquad \text{for } j = 1, \dots, K. \tag{8.5}$$

This shows that TC-NFP constitutes a shortest path problem (in the LP relaxation: a minimum cost flow problem) plus the K additional side constraints (8.5).

Due to the presence of equations (8.3d), TC-NFP is an integer program and hard to solve in general. However, commercial IP solvers like CPLEX are able to detect the network flow sub-structure and utilize it to speed up the solution process.

8.6 Simulation Results

In this section we present simulation results obtained by the IP formulations described above. The corresponding IP problems have been solved using the commercial state-of-the-art IP solver CPLEX [16]. The codes are investigated with respect to ML performance, error floor and performance of reliability decoding. The following codes are considered:

- WiMAX LDPC codes ($R = 5/6$),
- WiMAX-like LDPC codes ($R = 1/2$),
- BCH and RS codes,
- LTE turbo codes.

In the following, we use the union bound

$$P_B \le \sum_{i=1}^{n} A_i Q\left(\sqrt{2iR\frac{E_b}{N_0}}\right)$$

for an error floor estimation, where E_b is the energy per information bit, $Q(x)$ is the Q-function and R is the code rate [9]. The weight distribution coefficients A_w have been obtained by the IP formulations from (8.2).

block length	A_5	A_6	A_7	A_8
$N = 576$	24	0	312	
$N = 672$	0	0	168	924

Table 8.1: Weight distribution for the WiMAX LDPC codes with $R = 5/6$.

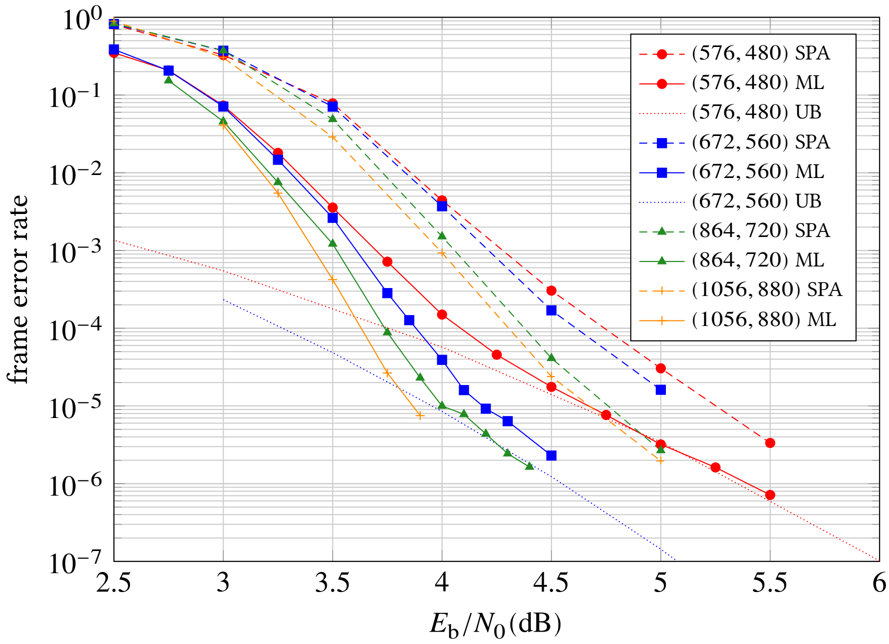


Figure 8.2: ML vs. SPA decoding of four WiMAX LDPC codes with $R = 5/6$. The dotted lines show the union bound estimation of the two smallest codes.

8.6.1 WiMAX-Compliant LDPC, $R = 5/6$

First we consider some high-rate LDPC codes from the WiMAX standard [7] with code rate $R = 5/6$ (Figure 8.2). ML decoding is done with the parity-check formulation (left hand side of (8.1)). Furthermore, the union bound for error-floor estimation based on the weight distributions in Table 8.1 is given.

The ML results are compared with SPA decoding using 40 iterations with layered scheduling. The loss of SPA ranges from 0.5 to 1 dB. A significant improvement in the error floor region can also be observed. Note that in hardware implementations of WiMAX LDPC decoders usually fewer iterations of the SPA are carried out. Moreover, often suboptimal-check node implementations are used that introduce additional loss.

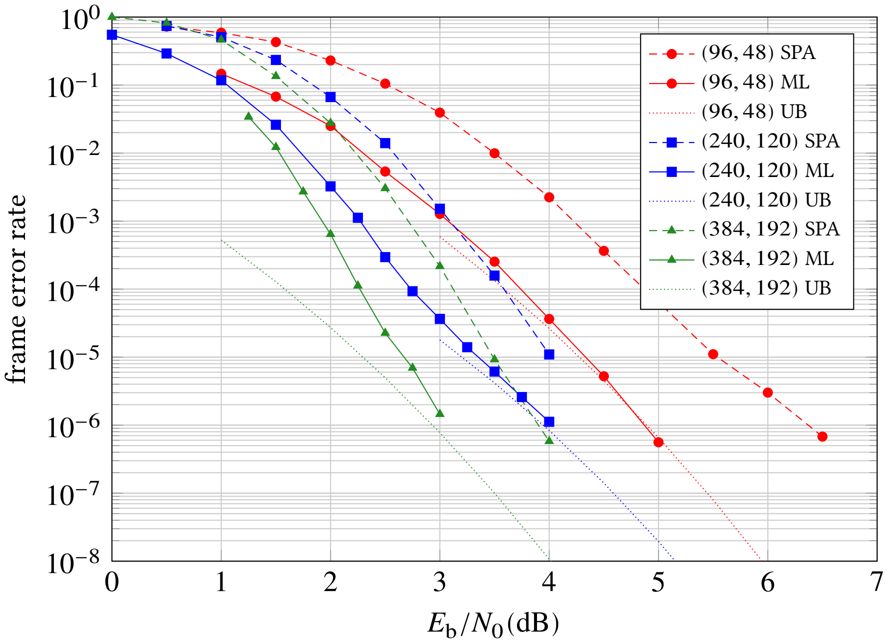


Figure 8.3: ML vs. SPA decoding of four WiMAX-like LDPC codes with $R = 1/2$. The dotted lines show the union bound estimation.

8.6.2 WiMAX-like LDPC, $R = 1/2$

The block lengths of the standard-compliant WiMAX LDPC codes with rate 1/2 are too large to be tackled by the IP solver. Therefore we designed some non-standard LDPC codes with shorter block lengths that have a similar structure as the WiMAX-compliant codes. During code design, we follow the design process of the WiMAX compliant codes. We take the macro matrix from the WiMAX standard and apply a lifting process to obtain a parity-check matrix [7]. In contrast to the (longer) WiMAX-compliant codes, we use smaller matrices for lifting and apply a progressive edge-growth technique [17] to determine the shift values.

The results are LDPC codes with shorter block lengths than specified in the WiMAX standard, but they have the same underlying structure. Therefore, we call these codes “WiMAX-like” LDPC codes. Figure 8.3 shows the simulation results under ML decoding and SPA. The SPA performance is obtained by using 40 iterations and layered scheduling.

8.6.3 BCH and RS Codes

Decoding of BCH and RS codes is considered using soft information (Figure 8.4). We present results for ML decoding, Chase decoding and OSD for the BCH(1023,993) code. Furthermore, the RS(63,55) code is investigated under OSD for its binary image [13].

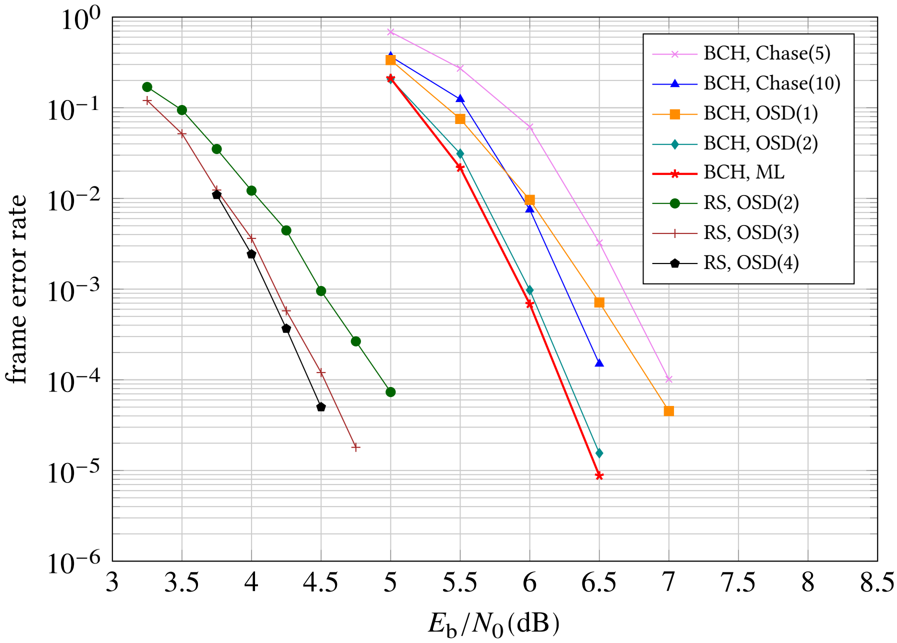


Figure 8.4: Decoding performance of the BCH(1023,993) and RS(63,55) codes under Chase, OSD and ML decoding.

8.6.4 LTE Compliant Turbo Codes

ML decoding of turbo codes is done using the network-flow-based IP formulation (Section 8.5). This formulation exploits the turbo code structure and therefore has a higher simulation speed.

In Figure 8.5, simulation results for LTE turbo codes [8] are given. The ML performance is compared to the turbo decoding algorithm using Log-MAP and 8 iterations. Note that in hardware implementations usually the Max-Log-MAP is used, which introduces an additional loss of about 0.1 dB.

ML as Lower Bound for New Decoding Algorithms

The newly obtained ML decoding simulation results are lower bounds on the FER for decoding algorithms. This is very useful to evaluate the correction capability of (new) decoding algorithms.

Figure 8.6 shows different decoding algorithms for an exemplary (96,48) LDPC code. Besides the SPA, we apply OSD and SPA with *information correction* (6 bits corrected) proposed in [18]. It can easily be read off that OSD(3) nearly achieves ML performance, OSD(2) approaches the ML performance by 0.3 dB. The SPA with information correction of 6 bits performs within 0.5 dB.

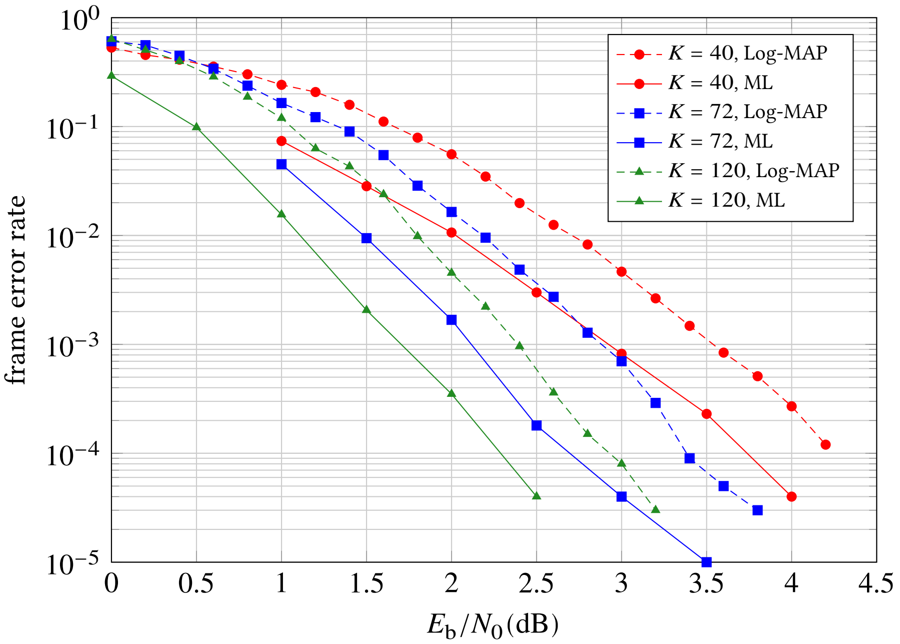


Figure 8.5: Comparison of ML and turbo decoding (Max-Log-MAP with 8 iterations) using LTE turbo codes of three different lengths with $R = 1/3$.

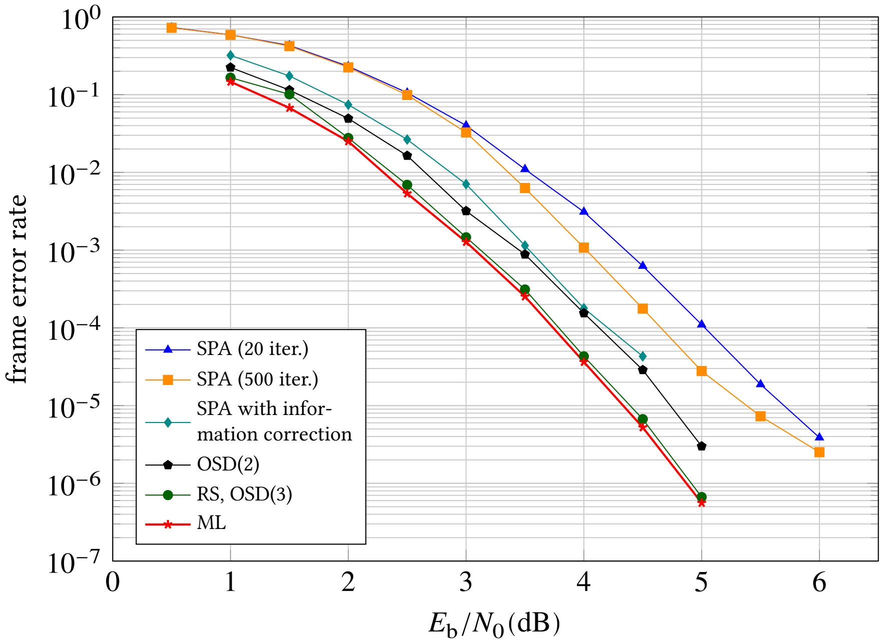


Figure 8.6: Comparison of different decoding algorithm for an exemplary (96,48) LDPC code.

8.7 Conclusion

This paper provides a framework to analyze linear block codes using integer programming. Several IP formulations have been presented that allow to analyze ML decoding performance, minimum-distance properties, the error floor and reliability-based decoding algorithms. Especially, new IP formulations have been presented for the Chase algorithm and OSD as well as for evaluating the weight distributions. It has been shown that ML decoding via IP can be very efficient. Block codes of lengths up to 1000 bits have been successfully simulated in a reasonable amount of time. New simulation results for ML decoding of various linear block codes have been shown. The presented as well as additional ML simulation results can be found online [19].

Acknowledgment

We gratefully acknowledge partial financial support by the DFG (project-ID: KI 1754/1-1) as well as the DAAD project *MathTurbo* (project-ID: 54565400).

References

[1] R. G. Gallager. *Low-Density Parity-Check Codes.* Cambridge, Massachusetts: M.I.T. Press, 1963.

[2] D. J. C. MacKay. “Good error-correcting codes based on very sparse matrices”. *IEEE Transactions on Information Theory* 45.2 (Mar. 1999), pp. 399–431. DOI: 10.1109/18.748992.

[3] R. M. Tanner. “A recursive approach to low complexity codes”. *IEEE Transactions on Information Theory* 27.5 (Sept. 1981), pp. 533–547. ISSN: 0018-9448. DOI: 10.1109/TIT.1981.1056404.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. “Near shannon limit error-correcting coding and decoding: turbo-codes”. In: *IEEE International Conference on Communications.* May 1993, pp. 1064–1070. DOI: 10.1109/ICC.1993.397441.

[5] D. Chase. “Class of algorithms for decoding block codes with channel measurement information”. *IEEE Transactions on Information Theory* 18.1 (Jan. 1972), pp. 170–182. DOI: 10.1109/TIT.1972.1054746.

[6] M. P. C. Fossorier and S. Lin. “Soft-decision decoding of linear block codes based on ordered statistics”. *IEEE Transactions on Information Theory* 41.5 (Sept. 1995), pp. 1379–1396. DOI: 10.1109/18.412683.

[7] *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1.* Feb. 2006. DOI: 10.1109/IEEESTD.2006.99107.

[8] *Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding (Release 8).* Technical Specification. 3$^{\text{rd}}$ Generation Partnership Project; Group Radio Access Network, Dec. 2008.

[9] S. Lin and D. Costello Jr. *Error Control Coding.* 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2004. ISBN: 0130426725.

[10] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience series in discrete mathematics and optimization, John Wiley & Sons, 1988.

[11] M. Breitbach et al. “Soft-decision decoding of linear block codes as optimization problem”. *European Transactions on Telecommunications* 9 (1998), pp. 289–293.

[12] A. Tanatmis et al. “A separation algorithm for improved LP-decoding of linear block codes”. *IEEE Transactions on Information Theory* 56.7 (July 2010), pp. 3277–3289. ISSN: 0018-9448. DOI: 10.1109/TIT.2010.2048489.

[13] M. El-Khamy and R. J. McEliece. “Iterative algebraic soft-decision list decoding of Reed-Solomon codes”. *IEEE Journal on Selected Areas in Communications* 24.3 (Mar. 2006), pp. 481–490. DOI: 10.1109/JSAC.2005.862399.

[14] M. Punekar et al. “Calculating the minimum distance of linear block codes via integer programming”. In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing.* Brest, France, Sept. 2010, pp. 329–333. DOI: 10.1109/ISTC.2010.5613894.

[15] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows.* Prentice-Hall, 1993.

[16] *IBM ILOG CPLEX Optimization Studio.* Software Package. Version 12.4. 2011.

[17] X.-Y. Hu, E. Eleftheriou, and D. Arnold. “Regular and irregular progressive edge-growth tanner graphs”. *IEEE Transactions on Information Theory* 51.1 (Jan. 2005), pp. 386–398. DOI: 10.1109/TIT.2004.839541.

[18] N. Varnica and M. Fossorier. “Belief-propagation with information correction: improved near maximum-likelihood decoding of low-density parity-check codes”. In: *Proceedings of IEEE International Symposium on Information Theory.* Chicago, IL, June–July 2004. DOI: 10.1109/ISIT.2004.1365380.

[19] M. Helmling and S. Scholl. *Database of ML Simulation Results.* Ed. by University of Kaiserslautern. 2014. URL: http://www.uni-kl.de/channel-codes.

Chapter 9

Paper IV: Towards an Exact Combinatorial Algorithm for LP Decoding of Turbo Codes

Michael Helmling and Stefan Ruzika

The following chapter is a reformatted and revised copy of a preprint that is publicly available online (http://arxiv.org/abs/1301.6363). A reduced version of the paper (due to the limitation of 5 pages for conference submissions, all proofs were omitted) was presented at the 2013 ISIT conference and appeared in the following refereed conference proceedings:

M. Helmling and S. Ruzika. "Towards combinatorial LP turbo decoding". In: *Proceedings of IEEE International Symposium on Information Theory.* Istanbul, Turkey, July 2013, pp. 1491–1495. DOI: 10.1109/ISIT.2013.6620475. arXiv: 1301.6363 `[cs.IT]`

Towards an Exact Combinatorial Algorithm for LP Decoding of Turbo Codes

Michael Helmling Stefan Ruzika

We present a novel algorithm that solves the turbo code LP decoding problem in a finite number of steps by Euclidean distance minimizations, which in turn rely on repeated shortest-path computations in the trellis graph representing the turbo code. Previous attempts to exploit the combinatorial graph structure only led to algorithms which are either of heuristic nature or do not guarantee finite convergence. A numerical study shows that our algorithm clearly beats the running time, up to a factor of 100, of generic commercial LP solvers for medium-sized codes, especially for high SNR values.

9.1 Introduction

Since its introduction by Feldman *et al.* in 2002 [1], Linear Programming-based channel decoding has gained tremendous interest because of its analytical power—LP decoding exhibits the maximum-likelihood (ML) certificate property [2], and the decoding behavior is completely determined by the explicitly described “fundamental” polytope [3]—combined with noteworthy error-correcting performance and the availability of efficient decoding algorithms.

Turbo codes, invented by Berrou *et al.* in 1993 [4], are a class of concatenated convolutional codes that, together with a heuristic iterative decoding algorithm, feature remarkable error-correcting performance.

While the first paper on LP decoding [1] actually dealt with turbo codes, the majority of publications in the area of LP decoding now focusses on LDPC codes [5] which provide similar performance (cf. [6] for a recent overview). Nevertheless, turbo codes have some analytical advantages, most importantly the inherent combinatorial structure by means of the trellis graph representations of the underlying convolutional encoders. ML Decoding of turbo codes is closely related to shortest-path and minimum-network-flow problems, both being classical, well-studied topics in optimization theory for which plenty efficient solution methods exist. The hardness of ML decoding is caused by additional conditions on the path through the trellis graphs (they are termed *agreeability constraints* in [1]) posed by the turbo code’s interleaver. Thus ML (LP) decoding is equivalent to solving a (LP-relaxed) shortest-path problem with additional linear side constraints.

So far, two methods for solving the LP have been proposed: General purpose LP solvers like CPLEX [7] are based on the matrix representation of the LP problem. They utilize either the simplex method or interior point approaches [8], but do not exploit any structural properties of the specific problem. Lagrangian relaxation in conjunction with subgradient optimization [1, 9], on the other hand, utilizes this structure, but has practical limitations, most notably it usually converges very slowly.

This paper presents a new approach to solve the LP decoding problem exactly by an algorithm that exploits its graphical substructure, thus combining the analytical power of the LP approach with the running-time benefits of a combinatorial method which seems to be a necessary requirement for practical implementation. Our basic idea is to construct an alternative polytope in the space defined by the additional constraints (called *constraints space*) and show how the LP solution corresponds to a specific point $z_{\mathrm{LP}}^{\mathcal{Q}}$ of that polytope. Then, we show how to computationally find $z_{\mathrm{LP}}^{\mathcal{Q}}$ by a geometric algorithm that relies on a sequence of shortest-path computations in the trellis graphs.

The reinterpretation of constrained optimization problems in constraints space was first developed in the context of multicriteria optimization in [10], where it is applied to minimum spanning tree problems with a single side constraint. In 2010, Tanatmis [11] applied this theory to the turbo decoding problem. His algorithm showed a drastic speedup compared to a general purpose LP solver, however it only works for up to two constraints, while in real-world turbo codes the number of constraints equals the information length.

By adapting an algorithm by Wolfe [12] to compute in a polytope the point with minimum Euclidean norm, we are able to overcome these limitations and decode turbo codes with lengths of practical interest. The algorithm is, compared to previous methods, advantageous not only in terms of running time, but also gives valuable information that can help to improve the error-correcting performance. Furthermore, branch-and-bound methods for integer programming-based ML decoding depend upon fast lower bound computations, mostly given by LP relaxations, and can often be significantly improved by dedicated methods that evaluate combinatorial properties of the LP solutions. Since our LP decoder contains such information, it could also be considered a step towards IP-based algorithms with the potential of practical implementation.

9.2 Background and Notation

9.2.1 Definition of Turbo Codes

A k-dimensional subspace C of the vector space $\mathbb{F}_2^n$ (where $\mathbb{F}_2 = \{0, 1\}$ denotes the binary field), is called an (n, k) binary linear block code, where n is the block length and k the information (or input) length. One way to define a code is by an appropriate encoding function e_C, for which any bijective linear mapping from $\mathbb{F}_2^k$ onto C qualifies. This paper deals with turbo codes [4], a special class of block codes built by interconnecting (at least) two convolutional codes (see e.g. [13]). For the sake of clear notation, we focus on turbo codes as used in the 3GPP LTE standard [14]—i.e., systematic, parallely concatenated turbo codes with two identical terminated rate-1 constituent encoders—despite the fact that our approach is applicable to arbitrary turbo coding schemes. An in-depth covering of turbo code construction can be found in [15].

An (n, k) turbo code $\mathrm{TC} = \mathrm{TC}(C, \pi)$ is defined by a rate-1 convolutional (n_C, k) code C with constraint length d and a permutation $\pi \in \mathbb{S}_k$ such that $n = k + 2 \cdot n_C$. Because we consider terminated convolutional codes only (i.e., there is a designated terminal state of the encoder), the final d bits of the information sequence (also called the *tail*) are not free to choose and thus can not carry any information. Consequently, those bits together with the corresponding d output bits are considered part of the output, which yields $n_C = k + 2 \cdot d$ and a code rate slightly below 1. Let $e_C : \mathbb{F}_2^k \longrightarrow \mathbb{F}_2^{n_C}$ be the associated encoding function. Then, the encoding function of TC is defined as

$$\begin{aligned} e_{\mathrm{TC}} \colon \mathbb{F}_2^k &\longrightarrow \mathbb{F}_2^{k+2\cdot n_C} \\ e_{\mathrm{TC}}(x) &= (x \mid e_C(x) \mid e_C(\pi(x))) \end{aligned} \tag{9.1}$$

where $\pi(x) = (x_{\pi(1)}, \ldots, x_{\pi(k)})$. In other words, the codeword for an input word x is obtained by concatenating

- a copy of x itself,
- a copy of x encoded by C, and
- a copy of x, permuted by π and encoded by C afterwards.

Figure 9.1 shows a circuit-type visualization of this definition.

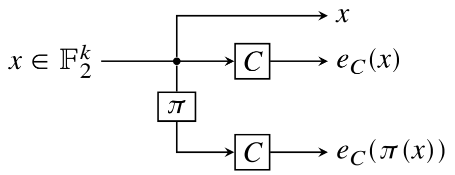


Figure 9.1: Turbo encoder with two convolutional encoders C and interleaver π.

9.2.2 Trellis Graphs of Convolutional Codes

A convolutional code with a specific length is represented naturally by its trellis graph, which is obtained by unfolding the code-defining finite state machine in the time domain: Each vertex of the trellis represents the state at a specific point in time, while edges correspond to valid transitions between two subsequent states and exhibit labels with the corresponding input and output bit, respectively. The following description of convolutional codes loosely follows [6, Section V.C], albeit the notation slightly differs.

We denote a trellis by $T = (V, E)$ with vertex set V and edge set E. Vertices are indexed by time step and state; i.e., $v_{i,s}$ denotes the vertex corresponding to state $s \in \{0, \dots, 2^d - 1\}$ at time $i \in \{1, \dots, k + d + 1\}$. An edge in turn is identified by the time and state of its tail vertex plus its input label, so $e_{i,s,b}$ denotes the edge outgoing from $v_{i,s}$ with input bit $b \in \{0, 1\}$. We call vertical "slices", i.e., the subgraphs induced by the edges of a single time step, *segments* of the trellis. Formally, the segment at time i is

$$\begin{aligned} & S_i = (V_i, E_i), \\ \text{where} \quad & V_i = \left\{v_{j,s} \in V \colon j \in \{i, i+1\}\right\} \\ \text{and} \quad & E_i = \left\{e_{j,s,b} \in E \colon j = i\right\}. \end{aligned}$$

Because the initial and final state of the convolutional encoder are fixed, the leading as well as the trailing d segments contain less than 2^d vertices. Figure 9.2 shows the first few segments of a trellis with $d = 2$.

By construction, the paths from the starting node to the end node in a trellis of a convolutional code C are in one-to-one correspondence with the codewords of C: Let $I_j \subset E_j$ and $O_j \subset E_j$ be those edges of S_j whose input label and output label, respectively, is a 1. The correspondence between a codeword $y \in \mathbb{F}_2^{k+2\cdot d}$ and the corresponding path $P = (e_1, \dots, e_{k+d})$ in T is given by

$$y_i = 1 \Leftrightarrow \begin{cases} e_{k+i} \in I_{k+i} & \text{for } 1 \le i \le d, \\ e_{i-d} \in O_{i-d} & \text{for } d < i \le k + 2 \cdot d, \end{cases} \tag{9.2}$$

where the first part accounts for the d "input" tail bits that are prepended by convention. From (9.2), for each $e \in E$ an index set $J_C(e)$ can be computed with the property that $e \in P \Rightarrow y_j = 1$ for all $j \in J_C(e)$. In our case, $|J_C(e)|$ varies from 0 (for edges in S_i, $i \le k$, with output label 0) to 2 (for edges in S_i, $k + 1 \le i \le k + d$, with both input and output label 1).

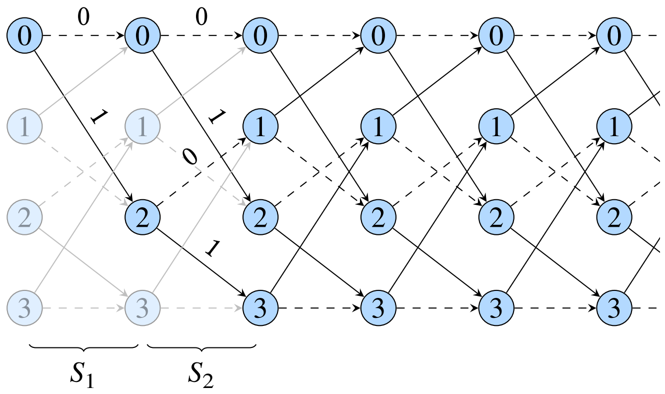


Figure 9.2: Excerpt from a trellis graph with four states and initial state 0. The style of an edge indicates the respective information bit, while the labels refer to the single parity bit.

The path-codeword relation can be exploited for maximum-likelihood (ML) decoding, if the codewords are transmitted through a memoryless binary-input output-symmetric (MBIOS) channel: Let $\lambda \in \mathbb{R}^{k+2\cdot d}$ be the vector of LLR values of the received signal. If we assign to each edge $e \in E$ the cost

$$c(e) = \sum_{j \in J_C(e)} \lambda_j,$$

it can be shown [2] that the shortest path in T corresponds to the ML codeword.

9.2.3 Trellis Representation of Turbo Codes

For turbo codes, we have two isomorphic trellis graphs, T^1 and T^2, according to the two component convolutional encoders. Let formally $T = (G^1 \cup G^2, E^1 \cup E^2)$, and by $P = P^1 \circ P^2$ denote the path that consists of P^1 in T^1 and P^2 in T^2. Only certain paths, called *agreeable*, actually correspond to codewords; namely, an agreeable path $P_1 \circ P_2 = (e^1_1, \dots, e^1_{k+d}, e^2_1, \dots, e^2_{k+d})$ must obey the k *consistency constraints*

$$e^1_i \in I^1_i \iff e^2_{\pi(i)} \in I^2_{\pi(i)} \quad \text{for } i = 1, \dots, k \tag{9.3}$$

because both encoders operate on the same information word, only that it is permuted for the second encoder. Consequently, ML decoding for turbo codes can be formulated as finding the shortest agreeable path in T. If an agreeable path contains $e^1_i \in I^1_i$, it must also contain $e^2_{\pi(i)} \in I^2_{\pi(i)}$, and thus $i \in J_C(e)$ for both $e = e^1_i$ and $e = e^2_{\pi(i)}$. To avoid counting the LLR values λ_i of the systematic bits ($1 \le i \le k$) twice in the objective function, we use the modified cost

$$c(e) = \sum_{j \in J_C(e)} \hat{\lambda}_j \text{ with } \hat{\lambda}_j = \begin{cases} \frac{\lambda_j}{2} & \text{if } 1 \le j \le k, \\ \lambda_j & \text{otherwise.} \end{cases} \tag{9.4}$$

Then, the ML decoding problem for turbo codes can be stated as the combinatorial optimization problem

$$\begin{aligned}
&\text{(TC-ML)} \quad && \min && \sum_{e \in P = P^1 \cup P^2} c(e) && \text{(9.5a)}\\
& && \text{s.t.} && P^1 \text{ is a path in } T^1 && \text{(9.5b)}\\
& && && P^2 \text{ is a path in } T^2 && \text{(9.5c)}\\
& && && P \text{ is agreeable.} && \text{(9.5d)}
\end{aligned}$$

The codeword variables y_i can be included into (TC-ML) by the constraints

$$y_i = \begin{cases} \sum\limits_{J_C(e) \ni i} \frac{f_e}{2} & \text{for } 1 \leq i \leq k, \\ \sum\limits_{J_C(e) \ni i} f_e & \text{for } i > k, \end{cases} \tag{9.6}$$

where the factor $1/2$ is analogical to (9.4). However, these variables are purely auxiliary in the LP and thus not needed.

It is straightforward to formulate TC-ML as an integer linear program by introducing a binary flow variable $f_e \in \{0,1\}$ for each $e \in E^1 \cup E^2$. The constraints (9.5b) and (9.5c) can be restated in terms of flow conservation and capacity constraints [16] which define the path polytopes $\mathcal{P}^1_{\text{path}}$ and $\mathcal{P}^2_{\text{path}}$, respectively. By also transforming (9.3) and (9.5a), we obtain

$$\begin{aligned}
&\text{(TC-IP)} \quad && \min && \sum_{e \in E^1 \cup E^2} c(e) \cdot f_e && && \text{(9.7a)}\\
& && \text{s.t.} && f^1 \in \mathcal{P}^1_{\text{path}} && && \text{(9.7b)}\\
& && && f^2 \in \mathcal{P}^2_{\text{path}} && && \text{(9.7c)}\\
& && && \sum_{e \in I^1_i} f_e = \sum_{e \in I^2_{\pi(i)}} f_e && i = 1, \ldots, k && \text{(9.7d)}\\
& && && f_e \in \{0,1\},\, e \in E. && && \text{(9.7e)}
\end{aligned}$$

9.2.4 Polyhedral Theory Background

Besides coding theory, this paper requires some bits of polyhedral theory. A *polytope* is the convex hull of a finite number of points: $\mathcal{P} = \operatorname{conv}(v_1, \ldots, v_n)$. It can be described either by its *vertices* (or *extreme points*), i.e., the unique minimal set fulfilling this defining property, or as the intersection of a finite number of halfspaces: $\mathcal{P} = \bigcap_{i=1}^m \{x \colon a_i^T x \leq b_i\}$. An inequality $a^T x \leq b$ is called *valid* for $\mathcal{P}$ if it is true for all $x \in \mathcal{P}$. In that case, the set $F_{a,b} = \{x \in \mathcal{P} : a^T x = b\}$ is called the *face induced by* the inequality. For any r satisfying $a^T r \geq b$ ($a^T r > b$) we say that the inequality *separates* (*strongly separates*) r from $\mathcal{P}$.

9.3 The LP Relaxation and Conventional Solution Methods

ML decoding of general linear block codes is known to be NP-hard [17]. While the computational complexity of TC-IP is still open, it is widely believed that this problem is NP-hard as well, which would imply that no polynomial-time algorithm can solve TC-IP unless P=NP.[1] By relaxing (9.7e) to $f_e \in [0,1]$, we get the LP relaxation (referred to as TC-LP) of the integer program TC-IP, which in contrast can be solved efficiently by the simplex method or interior point approaches [8]. Feldman *et al.* [1] were the first to analyze this relaxation and attested it reasonable decoding performance.

A general purpose LP solver, however, does not make use of the combinatorial substructure contained in TC-IP via (9.7b) and (9.7c) and thus wastes some potential of solving the problem more efficiently—while LPs are solvable in polynomial time, they do not scale too well, and the number of variables (about $2 \cdot |V| = (k+d) \cdot 2^{d+2}$) and constraints ($|V| + k$) in TC-LP is very large (practical values of d range roughly from 3 to 8).

Note that without the consistency constraints (9.7d), we could solve TC-LP by simply computing shortest paths in both trellis graphs, which is possible in time $\mathcal{O}(d \cdot (k+d))$ (as d is usually considered fixed, this amounts to a linear time complexity), even in the presence of negative weights, because the graphs are acyclic [19]. A popular approach for solving optimization problems that comprise "easy" subproblem plus some "complicating" additional constraints is to solve the Lagrangian dual [20] by subgradient optimization. If we define $g_i(f) = \sum_{e \in I_i^1} f_e - \sum_{e \in I_{\pi(i)}^2} f_e$, the constraints (9.7d) can be compactly rewritten as

$$g_i(f) = 0 \quad \text{for } i = 1, \dots, k. \tag{9.8}$$

The *Lagrangian relaxation with multiplier* $\mu \in \mathbb{R}^k$ is defined as

$$\begin{aligned}
\text{(TC-LR)} \qquad z(\mu) = \min \quad & \sum_{e \in E^1 \cup E^2} c(e) \cdot f_e + \sum_{i=1}^{k} \mu_k \cdot g_i(f) && (9.9\text{a})\\
\text{s.t.} \quad & f^1 \in \mathcal{P}^1_{\text{path}} && (9.9\text{b})\\
& f^2 \in \mathcal{P}^2_{\text{path}} && (9.9\text{c})\\
& f_e \in \{0,1\},\ e \in E. && (9.9\text{d})
\end{aligned}$$

For all $\mu \in \mathbb{R}^k$, the objective value of TC-LR is smaller or equal to that of TC-LP. The *Lagrangian dual problem* is to find multipliers μ that maximize this objective, thus minimizing the gap to the LP solution. It can be shown that in the optimal case both values coincide. Note that the feasible region of TC-LR is the combined path polytope of both T_1 and T_2, so it can be solved by a shortest-path routine in both trellises with modified costs, and the integrality condition on f is fulfilled automatically. Applying Lagrangian relaxation to turbo decoding was already proposed by Feldman *et al.* [1] and further elaborated by Tanatmis *et al.* [9]; the latter reference

---

[1] Note that with state-of-the-art software and prohibitive computational effort, ML turbo decoding *can* be simulated off-line on desktop computers at least for small and medium code sizes [18].

combines the approach with a heuristic to tighten the integrality gap between TC-LP and TC-IP.

The Lagrangian dual is typically solved by a subgradient algorithm that iteratively adjusts the multipliers μ, converging (under some mild conditions) to the optimal value [20]. However, the convergence is often slow in practice and the limit is not guaranteed to be ever reached exactly. Additionally, the dual only informs us about the objective value; recovering the actual solution of the problem requires additional work. In summary, subgradient algorithms suffer from three major flaws. The main result of this paper is an alternative algorithm which exhibits none of these.

9.4 An Equivalent Problem in Constraints Space

Like Lagrangian dualization, our algorithm also uses a relaxed formulation of TC-IP with modified objective function that resembles TC-LR. However, via geometric interpretation of the image of the path polytope in the "constraints space", as defined below, the exact LP solution is found in finitely many steps.

9.4.1 The Image Polytope $\mathcal{Q}$

Let $\mathcal{P}_{\text{path}} = \mathcal{P}^1_{\text{path}} \times \mathcal{P}^2_{\text{path}}$ be the feasible region of TC-LR. We define the map

$$\begin{aligned} \mathfrak{D}\colon \mathcal{P}_{\text{path}} &\to \mathbb{R}^{k+1} \\ f &\mapsto (g_1(f), \dots, g_k(f), c(f))^T \end{aligned}$$

where $c(f) = \sum_{e \in E^1 \cup E^2} c(e) \cdot f_e$ is a short hand for the objective function value of TC-LP. For a path f, the first k coordinates v_i, $i = 1, \dots, k$, of $v = \mathfrak{D}(f)$ tell if and how the conditions $g_i(f) = 0$ are violated, while the last coordinate v_{k+1} equals the cost of f. Let $\mathcal{Q} = \mathfrak{D}(\mathcal{P}_{\text{path}})$ be the image of the path polytope under $\mathfrak{D}$. The following results are immediate:

9.1 Lemma:

(1) *$\mathcal{Q}$ is a polytope.*

(2) *If f represents an agreeable path in T, then $\mathfrak{D}(f)$ is located on the $(k+1)$st axis (henceforth called c-axis or A_c).*

(3) *If v is a vertex of $\mathcal{Q}$ and $v = \mathfrak{D}(f)$ for some $f \in \mathcal{P}_{\text{path}}$, then f is also a vertex of $\mathcal{P}_{\text{path}}$.* ◁

In the situation that $v = \mathfrak{D}(f)$ we will also write $f = \mathfrak{D}^{-1}(v)$ with the meaning that f is *any* preimage of v, which need not be unique.

We consider the auxiliary problem

$$\begin{aligned}(\text{TC-LP}_{\mathcal{Q}}) \qquad z_{\text{LP}}^{\mathcal{Q}} = \min \quad & v_{k+1} && (9.10\text{a})\\ \text{s.t.} \quad & v \in \mathcal{Q} && (9.10\text{b})\\ & v \in A_c && (9.10\text{c})\end{aligned}$$

the solution of which is the lower "piercing point" of the axis A_c through $\mathcal{Q}$. Note that due to (9.10c), k of the $k+1$ variables in TC-LP($\mathcal{Q}$) are fixed to zero, thus the problem is in a sense one-dimensional, the feasible region being the (one-dimensional) projection of $\mathcal{Q}$ onto A_c. Nevertheless, the following theroem shows that TC-LP$_{\mathcal{Q}}$ and TC-LP are essentially equivalent.

9.2 Theorem: *Let v_{LP} be an optimal solution of TC-LP$_{\mathcal{Q}}$ with objective value $z_{\text{LP}}^{\mathcal{Q}}$ and $f_{\text{LP}} = \mathfrak{D}^{-1}(v_{\text{LP}}) \in \mathcal{P}_{\text{path}}$ the corresponding flow. Then $z_{\text{LP}}^{\mathcal{Q}} = z_{\text{LP}}$, the optimal objective value of TC-LP, and f_{LP} is an optimal solution of TC-LP.* ◁

Proof. First we show $z_{\text{LP}}^{\mathcal{Q}} \leq z_{\text{LP}}$. Let f_{LP} be an optimal solution of TC-LP with cost $c(f_{\text{LP}}) = z_{\text{LP}}$. Then $\mathfrak{D}(f_{\text{LP}}) = (0, \dots, 0, z_{\text{LP}})$ by definition of $\mathfrak{D}$, since f_{LP} is feasible and thus $g_1(f_{\text{LP}}) = \cdots = g_k(f_{\text{LP}}) = 0$. Hence $\mathfrak{D}(f_{\text{LP}}) \in A_c \cap \mathcal{Q}$ with $\mathfrak{D}(f_{\text{LP}})_{k+1} = z_{\text{LP}}$, from which it follows that $z_{\text{LP}}^{\mathcal{Q}} \leq z_{\text{LP}}$.

If we assume on the other hand that $z_{\text{LP}}^{\mathcal{Q}} < z_{\text{LP}}$, there must be a $v \in A_c \cap \mathcal{Q}$ such that $v_{k+1} < z_{\text{LP}}$. By definition of $\mathfrak{D}$ this implies the existence of a flow $f = \mathfrak{D}^{-1}(v)$ with $g_1(f) = \cdots = g_k(f) = 0$, hence a feasible one, and $c(f) = v_{k+1} < z_{\text{LP}}$, contradicting optimality of z_{LP}. □

While we do not have an explicit representation of $\mathcal{Q}$ (by means of either vertices or inequalities) at hand, we can easily minimize linear functionals over $\mathcal{Q}$:

9.3 Observation: *The problem*

$$\begin{aligned}(\mathit{LP}_{\mathcal{Q}}) \qquad \min \quad & \gamma^T v\\ \mathit{s.t.} \quad & v \in \mathcal{Q}\end{aligned}$$

can be solved by first computing an optimal solution f^* of the weighted sum problem

$$\begin{aligned}(\mathit{TC\text{-}WS}) \qquad \min \quad & \sum_{i=1}^{k} \gamma_i \cdot g_i(f) + \gamma_{k+1} \cdot c(f)\\ \mathit{s.t.} \quad & f \in \mathcal{P}_{\text{path}}\end{aligned}$$

and then taking the image of f^* under $\mathfrak{D}$. As noted before, this can be achieved within running time $\mathcal{O}(n)$. ◁

Note that TC-WS is closely related to TC-LR: as long as $\gamma_{k+1} \neq 0$, we get the same problem by setting $\mu_i = \gamma_i/(\gamma_{k+1})$ in TC-LR.

9.4.2 Solving TC-LP$_{\mathcal{Q}}$ with Nearest-Point Calculations

Our algorithm solves TC-LP$_{\mathcal{Q}}$ by a series of nearest-point computations between $\mathcal{Q}$ and reference points r^i on A_c, the last of which gives a face of $\mathcal{Q}$ containing the optimal solution v_{LP}.

For each $r \in \mathbb{R}^{k+1}$, we denote by

$$\mathrm{NP}(r) = \arg\min_{v \in \mathcal{Q}} \|v - r\|_2$$

the nearest point to r in $\mathcal{Q}$ with respect to Euclidean norm and define $a(r) = r - \mathrm{NP}(r)$ and $b(r) = a(r)^T \mathrm{NP}(r)$. The following well-known result will be used frequently below.

9.4 Lemma: *The inequality*

$$a(r)^T v \le b(r) \tag{9.11}$$

is valid for $\mathcal{Q}$ and induces a face containing $\mathrm{NP}(r)$*, which we call* $\mathrm{NF}(r)$*. If $r \notin \mathcal{Q}$,* (9.11) *strongly separates r from $\mathcal{Q}$.* ◁

The following theorem is the foundation of our algorithm.

9.5 Theorem: *There exists an $\varepsilon > 0$ such that $v_{\mathrm{LP}} \in \mathrm{NF}(r)$ holds for all r inside the open line segment $\left(v_{\mathrm{LP}}, v_{\mathrm{LP}} - (0, \dots, 0, \varepsilon)^T\right)$.* ◁

Our constructive proof of Theorem 9.5 shows how find a point inside the interval mentioned in the theorem. The outline is as follows: At first, start with a reference point $r \in A_c$ that is guaranteed to be located below v_{LP}. Then, we iteratively compute $\mathrm{NF}(r)$ and update r to be the intersection of A_c with the hyperplane defining $\mathrm{NF}(r)$. The following lemmas show that this procedure, which is illustrated in Figure 9.3 for the two-dimensional case, is valid and finite.

The first result is that the hyperplane defining $\mathrm{NF}(r)$ is always oriented "downwards".

9.6 Lemma: *Let $r = (0, \dots, 0, \rho)^T$ with $\rho < z_{\mathrm{LP}}^{\mathcal{Q}}$ and let $a(r)^T v \le b(r)$ be the inequality defined in* (9.11)*. Then $a(r)_{k+1} < 0$.* ◁

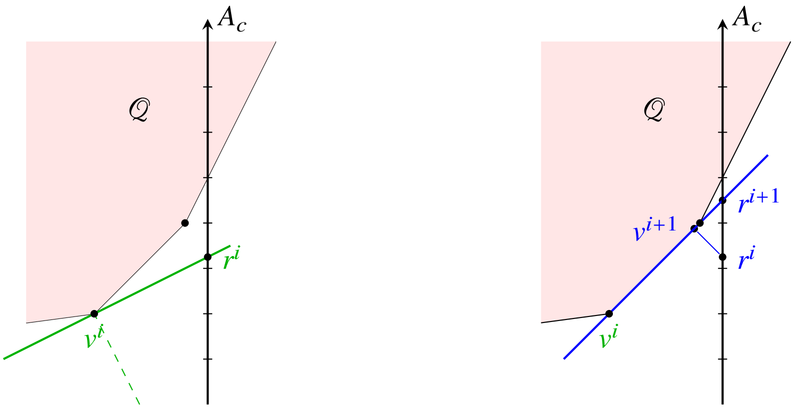
Proof. Assuming $a(r)_{k+1} \ge 0$, we obtain $a(r)^T v_{\mathrm{LP}} = a(r)_{k+1} z_{\mathrm{LP}}^{\mathcal{Q}} \ge a(r)_{k+1} \rho = a(r)^T r > b(r)$, which contradicts $v_{\mathrm{LP}} \in \mathcal{Q}$ by Lemma 9.4. Note that the equalities hold because both v_{LP} and r are elements of A_c, the first inequality stems from the assumptions on $a(r)_{k+1}$ and ρ, and the second follows from Lemma 9.4. □

Next we show that updating r leads to a different nearest face, unless we have arrived at the optimal solution.

9.7 Lemma: *Under the same assumptions as in Lemma 9.6, let $s \in \mathbb{R}^{k+1}$ with*

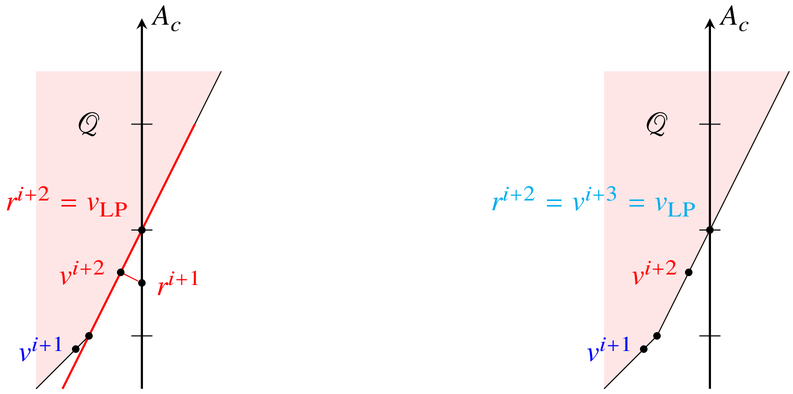
$$s = \left(0, \dots, 0, \frac{b(r)}{a(r)_{k+1}}\right)$$

be the point where the separating hyperplane and A_c intersect. If $\mathrm{NF}(r) = \mathrm{NF}(s)$*, then $s = v_{\mathrm{LP}}$.* ◁



(a) Step i: $v^i = \mathfrak{D}(f^i)$ is found as nearest point to some reference point r^{i-1}. The intersection of the separating hyperplane with the axis A_c, r^1, will be the reference point of the next iteration.

(b) Step $i + 1$: Note that the induced face of $\mathcal{Q}$ here is a facet, while it was a 0-dimensional face in step i.



(c) Step $i + 2$ (zoomed in): The facet induced in this step intersects A_c at v_{LP}, but the algorithm can not yet detect this.

(d) Step $i + 3$: Optimality is detected by $v^{i+3} = r^{i+2}$. The solution $\mathfrak{D}^{-1}(v_{\text{LP}})$ is returned.

Figure 9.3: Schematic execution of Algorithm 9.1 in image space.

Proof. We use contraposition to show that $s \neq v_{\mathrm{LP}}$ implies $\mathrm{NF}(r) \neq \mathrm{NF}(s)$, so assume $s \neq v_{\mathrm{LP}}$. We know that $a(r)^T v \leq b(r)$ is valid for $\mathcal{Q}$ and $a(r)^T s = a(r)_{k+1} s_{k+1} = b(r)$ by construction. This implies that $s \notin \mathcal{Q}$; otherwise we would have $s = v_{\mathrm{LP}}$ because for all $\zeta < s_{k+1}$, $a(r)^T(\zeta e_{k+1}) > b(r)$, so s would really be the lowest point on A_c that is also in $\mathcal{Q}$ and thus optimal.

It follows that $y_N = \mathrm{NP}(s) \neq s$. Since $y_N \in \mathcal{Q}$ and $a(r)^T v \leq b(r)$ is valid for $\mathcal{Q}$, we have $a(r)^T y_N \leq b(r)$.

Case 1: $a(r)^T y_N < b(r)$. Then $y_N \notin \mathrm{NF}(r)$, but $y_N \in \mathrm{NF}(s)$ by definition, which proves the claim for this case.

Case 2: $a(r)^T y_N = b(r)$. From $a(r)^T r > b(r)$ and $a(r)^T s = b(r)$ we obtain

$$a(r)^T r > a(r)^T s \tag{9.12a}$$
$$\Rightarrow a(r)^T (r - s) > 0 \tag{9.12b}$$
$$\Rightarrow a(r)_{k+1}(r_{k+1} - z_{k+1}) > 0 \tag{9.12c}$$
$$\Rightarrow r_{k+1} < s_{k+1}, \tag{9.12d}$$

where we have used again $a(r)_{k+1} < 0$ and the fact that $r, s \in A_c$.

Applying Lemma 9.6 to s as reference point we obtain $a(s)_{k+1} = (s - y_N)_{k+1} < 0$, hence

$$(y_N)_{k+1} > s_{k+1} \tag{9.13a}$$
$$\Rightarrow (y_N)_{k+1}(s_{k+1} - r_{k+1}) > s_{k+1}(s_{k+1} - r_{k+1}) \qquad \text{by (9.12d)} \tag{9.13b}$$
$$\Rightarrow y_N^T(s - r) > s^T(s - r) \tag{9.13c}$$
$$\Rightarrow y_N^T s - y_N^T r + s^T r - s^T s > 0 \tag{9.13d}$$

Plugging the definitions into $a(r)^T y_N = b(r) = a(r)^T s$ yields $(r - \mathrm{NP}(r))^T y_N = (r - \mathrm{NP}(r))^T s$ or $r^T s - r^T y_N = \mathrm{NP}(r)^T s - \mathrm{NP}(r)^T y_N$. Using this we continue from (9.13d) with

$$\Rightarrow y_N^T s + \mathrm{NP}(r)^T s - \mathrm{NP}(r)^T y_N - s^T s > 0 \tag{9.13e}$$
$$\Rightarrow \mathrm{NP}(r)^T(s - y_N) > s^T(s - y_N) \tag{9.13f}$$
$$\Rightarrow a(s)^T \mathrm{NP}(r) > a(s)^T s > b(s) \tag{9.13g}$$

Thus, $\mathrm{NP}(r) \notin \mathrm{NF}(s) = \{v \in \mathcal{Q} : a(s)^T v \leq b(s)\}$, but $\mathrm{NP}(r) \in \mathrm{NF}(r)$ by definition, so those faces must differ. □

Now we show the auxiliary result that if two inequalities induce the same face, then also every convex combination of them does.

9.8 Lemma: *Let $\mathcal{P}$ be a polytope, $x^1, x^2 \in \mathcal{P}$, and $r^1, r^2 \notin \mathcal{P}$. If the inequalities*

$$\begin{aligned} & H^1: (r^1 - x^1)^T x \leq (r^1 - x^1)^T x^1 \\ \textit{and} \quad & H^2: (r^2 - x^2)^T x \leq (r^2 - x^2)^T x^2 \end{aligned}$$

both induce the same face F of $\mathcal{P}$, then also

$$\bar{H}: (\bar{r} - \bar{x})^T x \leq (\bar{r} - \bar{x})^T \bar{x}$$

with $\bar{r} = \lambda r^1 + (1-\lambda) r^2$, $\bar{x} = \lambda x^2 + (1-\lambda) x^2$, $0 \leq \lambda \leq 1$, is valid and induces F. ◁

Proof. We first show that $\bar{H}$ is valid. For $x \in \mathcal{P}$

$$\begin{aligned} (\bar{r} - \bar{x})^T x &= \lambda (r^1 - x^1)^T x + (1-\lambda)(r^2 - x^2)^T x \\ &\leq \lambda (r^1 - x^1)^T x^1 + (1-\lambda)(r^2 - x^2)^T x^2 \\ &= \left(\lambda (r^1 - x^1) + (1-\lambda)(r^2 - x^2)\right)^T \left(\lambda x^1 + (1-\lambda) x^2\right) \\ &= (\bar{r} - \bar{x})^T \bar{x}, \end{aligned}$$

where we have used the fact that H^1 is satisfied with equality for $x = x^2$ and vice versa because of the assumptions. Since we have shown that $\bar{H}$ is valid, it must induce a face $\bar{F}$. It remains to show that $F = \bar{F}$.

$F \subseteq \bar{F}$: $x \in F$ fulfills both H^1 and H^2 with equality, so we can carry out the above calculation with an "=" in the second line to conclude $x \in \bar{F}$.

$\bar{F} \subseteq F$: Let $x \in \bar{F}$ and assume $x \notin F$, which implies $(r^i - x^i)^T x < (r^i - x^i)^T x^i$ for $i \in \{1, 2\}$. Then $\lambda (r^1 - x^1)^T x^1 + (1-\lambda)(r^2 - x^2)^T x^2 > \lambda (r^1 - x^1)^T x + (1-\lambda)(r^2 - x^2)^T x = (\bar{r} - \bar{x})^T x = (\bar{r} - \bar{x})^T \bar{x} = \lambda (r^1 - x^1)^T x^1 + (1-\lambda)(r^2 - x^2)^T x^2$, which is a contradiction.

This concludes the proof. □

The above lemma is used to show that the part of A_c that lies below v_{LP} dissects into intervals such that reference points within one interval yield the same face of $\mathcal{Q}$.

9.9 Lemma: *If $r^1, r^2 \in A_c$ with $r^1_{k+1} < r^2_{k+1} < z^{\mathcal{Q}}_{\mathrm{LP}}$ and $\mathrm{NF}(r^1) = \mathrm{NF}(r^2)$, then $\mathrm{NF}(r) = \mathrm{NF}(r^1)$ for all $r \in [r^1, r^2]$.* ◁

Proof. Let $v^i = \mathrm{NP}(r^i)$ for $i \in \{1, 2\}$. By Lemma 9.8, for each $\lambda \in (0, 1)$ and $\bar{r} = \lambda r^1 + (1-\lambda) r^2$, $\bar{v} = \lambda v^1 + (1-\lambda) v^2$, it holds

$$\{v \in \mathcal{Q} : (\bar{r} - \bar{v})^T v = (\bar{r} - \bar{v})^T \bar{v}\} = \mathrm{NF}(r^1),$$

and applying the converse statement from Lemma 9.4 follows $\bar{v} = \mathrm{NP}(\bar{r})$, so $\mathrm{NF}(\bar{r}) = \mathrm{NF}(r^1)$ as claimed. □

Now we have alle the ingredients at hand to prove our theorem.

Proof (of Theorem 9.5). First we show that there exists at least one r with the desired properties.

Choose some arbitrary $r^0 \in A_c$ with $r^0_{k+1} < z^{\mathcal{Q}}_{\mathrm{LP}}$ (thus $r^0 \notin \mathcal{Q}$). If $v_{\mathrm{LP}} \in \mathrm{NF}(r^0)$, we are done. Otherwise, Lemma 9.7 tells us how to find an r^1 with $r^1_{k+1} > r^0_{k+1}$ such that $\mathrm{NF}(r^1) \neq \mathrm{NF}(r^0)$. Iterating this argument and assuming that v_{LP} is never contained in the induced face results in a sequence $(r^i)_i$ with $r^{i+1}_{k+1} > r^i_{k+1}$ for all i. Because of Lemma 9.9, $\mathrm{NF}(r^{i+1}) \neq \mathrm{NF}(r^i)$ implies $\mathrm{NF}(r^{i+1}) \neq \mathrm{NF}(r^l)$ for all $0 \leq l < i+1$, so that all $\mathrm{NF}(r^i)$ are distinct. But since there are only finitely many faces of $\mathcal{Q}$, this can not be true, so eventually there must be an i^* such that $v_{\mathrm{LP}} \in \mathrm{NF}(r^{i^*})$.

Now let $r^* \in A_c$ be any such point whose existence we have just proven, $v_N = \mathrm{NP}(r^*)$ and $\lambda \in (0,1]$. Let $\bar{r} = \lambda r^* + (1-\lambda)v_{\mathrm{LP}}$ and $\bar{v} = \lambda v_N + (1-\lambda)v_{\mathrm{LP}}$. We use similar arguments as in the proof of Lemma 9.8 to show that

$$(\bar{r} - \bar{v})^T v \leq (\bar{r} - \bar{v})^T \bar{v} \tag{9.14}$$

induces $\mathrm{NF}(r^*)$. For $v \in \mathcal{Q}$,

$$\begin{aligned}
(\bar{r} - \bar{v})^T v &= (\lambda(r^* - v_N) + (1-\lambda)(v_{\mathrm{LP}} - v_{\mathrm{LP}}))^T v \\
&= \lambda(r^* - v_N)^T v \\
&\leq \lambda(r^* - v_N)^T v_N \\
&= \lambda(\lambda(r^* - v_N)^T v_N + (1-\lambda)(r^* - v_N)^T v_{\mathrm{LP}}) \\
&= \lambda(r^* - v_N)^T(\lambda v_N + (1-\lambda)v_{\mathrm{LP}}) \\
&= (\bar{r} - \bar{v})^T \bar{v}.
\end{aligned}$$

So the inequality is valid, and since again for $v \in \mathrm{NF}(r^*)$ equality holds in the third line, we know that the face $\bar{F}$ induced by (9.14) contains $\mathrm{NF}(r^*)$.

Now let $v \in \bar{F}$, i.e., $(\bar{r} - \bar{x})^T v = (\bar{r} - \bar{x})^T \bar{x}$. From the above equations we conclude $\lambda(r^* - v_N)^T v = \lambda(r^* - v_N)^T v_N$, and because $\lambda > 0$ this implies $v \in \mathrm{NF}(r^*)$.

Because the above holds for any $0 < \lambda \leq 1$, we can choose $\bar{r}$ arbitrarily close to v_{LP} on A_c, which completes the proof. □

9.4.3 Solving the Nearest Point Problems

It remains to show how to solve the nearest point problems arising in the discussion above. To that end, we utilize an algorithm by Wolfe [12] that finds in a polytope the point with minimum Euclidean norm. Wolfe's algorithm elaborates on a set of vertices of the polytope that are obtained via minimization of linear objective functions. In our situation, this means that $\mathrm{LP}_{\mathcal{Q}}$ has to be solved repeatedly, which by Observation 9.3 boils down to the linear-time solvable weighted-sum shortest-path problem TC-WS. Note that by subtracting r from the results of $\mathrm{LP}_{\mathcal{Q}}$ and adding r to the final result, the algorithm can be used to calculate the minimum distance between $\mathcal{Q}$ and r also in the case $r \neq 0$.

The algorithm in [12] maintains in each iteration a subset P of the vertex set $V(\mathcal{Q})$ and a point x such that $x = \mathrm{NP}(\mathrm{aff}(P))$ lies in the relative interior of $\mathrm{conv}(P)$, where $\mathrm{aff}(P)$ is the affine hull of P. Such a set is called a *corral*, and we denote the nearest point in $\mathrm{aff}(P)$ by v_P^{aff}.

Initially $P = \{v_0\}$ for an arbitrary vertex v_0 and $x = v_0$. Note that then $v_P^{\mathrm{aff}} = v_0$ and P is indeed a corral. Then the following is executed iteratively (we explain afterwards how the computations are actually carried out):

(1) Solve $p = \arg\min_{v \in \mathcal{Q}} (x^T v)$.

(2) If $p = 0$ (0 is optimal) or $x^T p = x^T x$ (x is optimal), stop. Otherwise, set $P := P \cup \{p\}$ and compute $y = v_P^{\mathrm{aff}}$.

(3) If y is in the relative interior of $\mathrm{conv}(P)$, P is a corral. Set $x := y$ and continue at (1).

(4) Determine $z \in \mathrm{conv}(P) \cap \mathrm{conv}\{x, y\}$ with minimum distance to y; z will be a boundary point of $\mathrm{conv}(P)$.

(5) Remove from P some point that is not on the smallest face of $\mathrm{conv}(P)$ containing z, set $x := z$, and continue at (3)).

The algorithm will eventually find a corral P such that the nearest point of $\mathcal{Q}$ equals v_P^{aff}.

The computations in each step are performed as follows:

(1) This matches the solution of TC-WS.

(2) If we interchangeably use the symbol P for both the set of points and the matrix that contains the elements of P as columns, every $v \in \mathrm{aff}(P)$ can be characterized by some $\lambda \in \mathbb{R}^{|P|}$ such that $v = P\lambda$ and $e^T\lambda = 1$. Thus, the subproblem of determining v_P^{aff} can be written as

$$\begin{aligned} \min \quad & \|P\lambda\|_2^2 = \lambda^T P^T P \lambda \\ \text{s.t.} \quad & e^T \lambda = 1. \end{aligned}$$

It can be shown [12] that this is equivalent to solving the system of linear equations

$$\begin{aligned} (ee^T + P^T P)\mu &= e \\ \lambda &= \frac{1}{\|\mu\|_1}\mu. \end{aligned} \tag{9.15}$$

As an efficient method to solve (9.15), Wolfe suggests to maintain an upper triangular matrix R such that $R^T R = ee^T + P^T P$. Then the solution μ can be found by first solving $R^T \bar{\mu} = e$ for $\bar{\mu}$ and then $R\mu = \bar{\mu}$ for μ; both can be done by a simple backward substitution. When P changes, R can be updated relatively easily without the necessity of a complete recomputation [12].

(3) y is in the relative interior of $\mathrm{conv}(P)$ if and only if all coefficients λ_i in the convex representation of y satisfy $\lambda_i > 0$.

(4) By construction $x \in \operatorname{conv}(P)$. Let $x = \sum_{v \in P} \lambda_v v$ and $y = \sum_{v \in P} \mu_v v$, where $\sum_{v \in P} \lambda_v = \sum_{v \in P} \mu_v = 1$, but $\mu_p \leq 0$ for at least one p. The goal can then be restated as finding the minimal $\theta \in [0,1]$ such that $z_\theta = \theta x + (1-\theta) y \in \operatorname{conv}(P)$. Substituting the above expressions yields
$$z_\theta = \sum_{v \in P} \left(\theta \lambda_v + (1-\theta)\mu_v\right) v,$$
and the condition is that all coefficients are nonnegative. Thus, for all v with $\mu_v \leq 0$,
$$\theta \geq \frac{\mu_p}{\mu_p - \nu_p}$$
must hold. In summary, θ can be computed by the rule
$$\theta = \min\left\{1, \max\left\{\frac{\mu_p}{\mu_p - \nu_p} : \mu_p < 0\right\}\right\}.$$

(5) A point not contained in the smallest face of $\operatorname{conv}(P)$ containing z is not needed for the convex description of $z = \sum_{v \in P} \lambda_v v$; thus it can be identified by $\lambda_v = 0$.

9.4.4 Recovering the Optimal Flow and Pseudocodeword

So far we have shown how to compute the optimal *objective value*, but not the LP *solution*, i. e. the flow $f_{\mathrm{LP}} \in \mathcal{P}_{\mathrm{path}}$ and the (pseudo)codeword y. The algorithm yields its solution v_{LP} by means of a convex combination of extreme points of $\mathcal{Q}$:
$$v_{\mathrm{LP}} = \sum_{i=1}^{t} \lambda_i v_i, \quad \lambda_i \geq 0, \quad \sum_{i=1}^{t} \lambda_i = 1.$$
During its execution the preimage paths $f_i = \mathfrak{D}^{-1}(v_i)$ can be stored alongside with the v_i. Then, the LP-optimal flow f_{LP} is obtained by summing up the paths with the same weight coefficients λ, i.e.,
$$f_{\mathrm{LP}} = \sum_{i=1}^{t} \lambda_i \mathfrak{D}^{-1}(v_i).$$
In order to get the corresponding pseudocodeword, a simple computation based on (9.6) suffices. For most applications, however, the values of y are of interest only in the case that the decoder has found a valid codeword, i.e., $t = 1$ in the above sum. In such a case, the most recent solution of (TC-WS) is an agreeable path that immediately gives the codeword. No intermediate paths have to be stored, which can save a substantial amount of space and running time.

9.4.5 Efficient Reference Point Updates

As suggested by the proof of Theorem 9.5, the nearest point algorithm is run iteratively, and between two runs the $k+1$st component of r is increased by means of the rule $r_{k+1} =$

$b(r)/a(r)_{k+1}$. This section describes how some information from the previous iteration can be re-used to provide an efficient warm start for the next nearest point run.

Assume that in iteration i the point $\mathrm{NP}(r^i) = v^{i+1}$ has been found, inducing the face $\mathrm{NF}(r^i)$ defined by $a(r^i)^T v \leq b(r^i)$ of $\mathcal{Q}$. Recall that NPA internally computes the minimum l_2 norm of $\mathcal{Q} - r^i$. Thus, it outputs $\bar{v}^{i+1} = v^{i+1} - r^i$ as the convex combination of $t \leq k+1$ points $\bar{v}_j = v_j - r^i$, all of which are located on the corresponding face $\widehat{\mathrm{NF}}(r^i)$ of $\mathcal{Q} - r^i$:

$$\bar{v}^{i+1} = v^{i+1} - r^i = \sum_{j=1}^{t} \lambda_j (v_j - r^i) = \sum_{j=1}^{t} \lambda_j \bar{v}_j.$$

In the subsequent nearest point calculation, the norm of $\mathcal{Q} - r^{i+1}$ is minimized. Obviously $\widehat{\mathrm{NF}}(r^i)$ corresponds to a face $\widehat{\mathrm{NF}}(r^{i+1})$ of $\mathcal{Q} - r^{i+1}$, and we can initialize the algorithm with that face by simply adding $r^i - r^{i+1}$ to $\bar{v}$ and each $\bar{v}_j, j = 1, \ldots, t$, which yields

$$\bar{v}^{i+1} + r^i - r^{i+1} = v^{i+1} - r^{i+1} = \sum_{j=1}^{t} \lambda_j (v_j - r^{i+1})$$

and all $v_j - r^{i+1}$ are vertices of $\mathcal{Q} - r^{i+1}$. Note that $r^i - r^{i+1}$ is zero in all but the last component, so this update takes only $t \leq k+1$ steps.

In order to warm-start the nearest point algorithm, the auxiliary matrix R has to be recomputed as well. Using its definition $R^T R = ee^T + V^T V$, we can efficiently compute R by Cholesky decomposition. After these updates we can directly start the nearest point algorithm in Step 2. Numerical experiments have shown that this speeds up LP decoding by a factor of two. In particular, the computation time of the Cholesky decomposition is negligible.

9.5 The Complete Algorithm

Algorithm 9.1 formalizes the procedure developed in Section 9.4 in pseudocode. The initial reference point r^0 is generated by first minimizing $c(f)$ on $\mathcal{P}_{\mathrm{path}}$ (thus solving TC-WS with $\gamma = (0, \ldots, 0, 1)$ and projecting the result in constraints space onto A_c (Line 4). Thereby we ensure that either $r^0 \notin \mathcal{Q}$ or it is located on the boundary of $\mathcal{Q}$, in which case it already is the optimal LP solution. The solution of the nearest point problem and the recovery of the original flow are encapsulated in Lines 7 and 8.

9.6 Numerical Results

9.6.1 Running Time Comparison

To evaluate the computational performance of our algorithm, we compare its running time with the commercial general purpose LP solver CPLEX [7] which is said to be one of the most competitive implementations available.

Algorithm 9.1 Combinatorial Turbo LP Decoder (CTLP)

```
Initialize edge cost $c(f)$ by (9.4).
$f^0 \leftarrow \arg\min\{c(f) : f \in \mathscr{P}_{\text{path}}\}$.
$v^0 \leftarrow \mathfrak{D}(f^0)$.
$r^0 \leftarrow (0, \dots, 0, v^0_{k+1})^T$
$i \leftarrow 0$
while $v^i \neq r^i$ do
    $v^{i+1} \leftarrow \text{NP}(r^i) = \arg\min_{v \in \mathcal{Q}} \|v - r^i\|_2$
    $f^{i+1} = \mathfrak{D}^{-1}(v^{i+1})$
    $a^{i+1} \leftarrow v^{i+1} - r^i$
    $b^{i+1} \leftarrow a^{(i+1)T} v^{i+1}$
    $r^{i+1} \leftarrow \left(0, \dots, 0, b^{i+1} \big/ a^{i+1}_{k+1}\right)^T$
    $i \leftarrow i + 1$
end while
return $f^i$
```

SNR (dB)	0	1	2	3	4	5
time CPLEX (s × 10^{-2})	9.1	9.5	9.6	9.6	9.6	9.8
time CTLP (s × 10^{-2})	1.4	0.9	0.5	0.29	0.24	0.22
ratio	6.5	10.6	19	33	40	45

Table 9.4: Average CPU time per decoded instance (in 1/100s of seconds) for the $(132, 40)$ LTE turbo code, and the ratio by which CPLEX takes longer than our algorithm.

Simulations were run using LTE turbo codes with block lengths 132, 228, and 396, respectively, and a three-dimensional turbo code with block length 384 (taken from [21]) with various SNR values. For each SNR value, we have generated up to 10^5 noisy frames, where the computation was stopped when 200 decoding errors occured. This should ensure sufficient significance of the average results shown in Tables 9.4 and 9.6 to 9.8. The ML performance results are taken from [22].

As one can see, the benefit of using the new algorithm is larger for high SNR values. This becomes most eminent for the 3-D code for which the dimension of $\mathcal{Q}$ is the highest, where the new algorithm is slower than CPLEX for SNRs up to 2. The reason for this behavior can

SNR (dB)	0	1	2	3	4	5
time CPLEX (s × 10^{-1})	3.1	3.4	4.2	4.6	4.7	4.7
time CTLP (s × 10^{-1})	0.7	0.4	0.15	0.05	0.04	0.04
ratio	4.4	8.5	28	92	118	118

Table 9.6: Average decoding CPU time per instance for the $(228, 72)$ LTE turbo code.

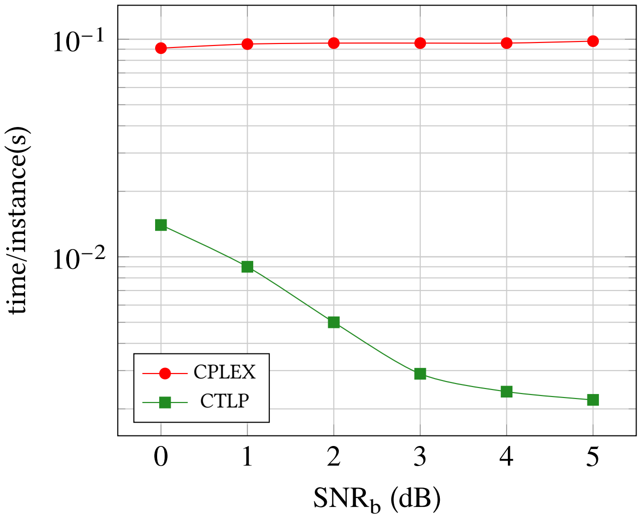


Figure 9.5: CPU time comparison for the $(132, 40)$ LTE Turbo code depending on the SNR value (note the logarithmic time scale).

SNR (dB)	0	1	2	3	4
time CPLEX ($s \times 10^{-1}$)	4.4	4.2	3.6	3.3	3.2
time CTLP ($s \times 10^{-1}$)	6.3	4.1	0.6	0.09	0.08
ratio	0.7	1	6	37	40

Table 9.7: Average decoding CPU time per instance for the $(396, 128)$ LTE turbo code.

SNR (dB)	0	1	2	3	4
time CPLEX (s)	1.4	1.2	0.9	0.72	0.57
time CTLP (s)	4.5	3.1	0.8	0.04	0.014
ratio	0.31	0.39	1.1	18	41

Table 9.8: Average decoding CPU time per instance for a $(384, 128)$ 3-D turbo code.

SNR (dB)	0	2	4
optimal face dimension	25.2	3.6	0.01
integral LP solution	0.26	0.89	0.9995
trivial instances	0	0.13	0.64
major nearest-point cycles	221	53	4
main loops of Algorithm 9.1	4.36	1.9	0.7

Table 9.9: Statistical data for the $(132, 40)$ LTE turbo code; average per-instance counts.

be explained by analyzing statistical information about various internal parameters of the algorithm when run with different SNR values:

- The average dimension of the optimal nearest face, found in the last iteration of the algorithm, drops substantially with increasing SNR. Intuitively, it is not surprising that finding a face that needs less vertices to describe can be found more efficiently.
- In particular, the share of instances for which the LP solution is integral (and thus, the face dimension is zero) increases with the SNR.
- There are some trivial instances where the initial shortest path among both trellis graphs is already a valid codeword. This occurs more often for low channel noise and allows for extremely fast solution (no nearest point calculations have to be carried out).
- The average number of major cycles of the nearest point algorithm performed per instance is seen to drop rapidly with increasing SNR.
- Likewise, the the average number of main loops (Line 6 of Algorithm 9.1) drops, reducing the required calls to CTLP.

Table 9.9 exemplarily contains the average per-instance values of these parameters for the $(132, 40)$ LTE code and SNRs 0, 2, and 4.

9.6.2 Numerical Stability

For larger codes, the dimension of $\mathcal{Q}$ becomes very large which leads to numerical difficulties in the nearest point algorithm: the equation systems solved during the execution sometimes have rank “almost zero” which leads to division by very small numbers, resulting in the floating-point value `NaN`. Careful adjustment of the tolerance values for equivalence checks help to eliminate this problem at least for the block lengths presented in this numerical study.

In addition, it has proven beneficial to divide all objective values by 10 in advance. Intuitively, this compresses $\mathcal{Q}$ along the c-axis, evening out the extensiveness of the polytope in the different dimensions (note that for all axes other than c, the values only range from -1 to 1).

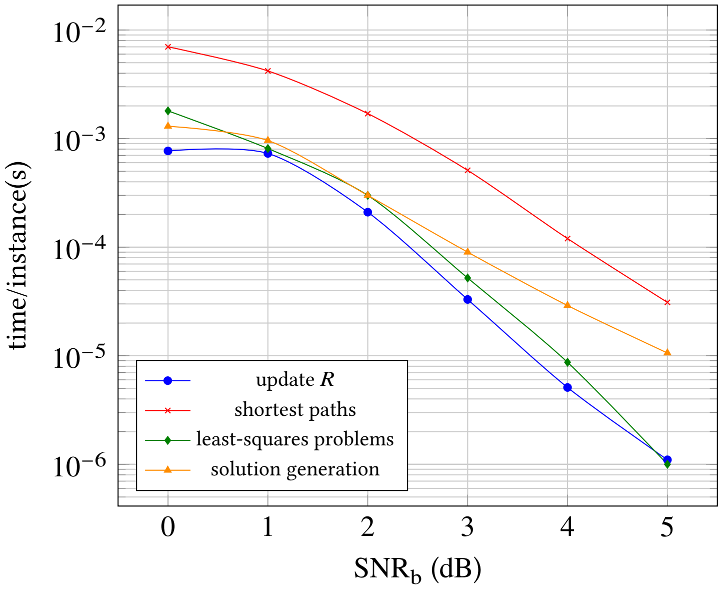


Figure 9.10: Average per-instance CPU time spent on various subroutines of the algorithm decoding the $(132, 40)$ LTE code: update of R, shortest-path computations, solution of the least squares problems, and generation of solutions in path space.

9.7 Improving Error-Correcting Performance

As discussed above, Algorithm 9.1 can be easily modified to return a list of paths f_i, $i = 1, \dots, t$, such that the LP solution is a convex combination of that paths. Each f_i can be split into a paths f_i^1 and f_i^2 through trellis T^1 and T^2, respectively. A path in a trellis, in turn, can uniquely be extended to a codeword. Thus, we have a total of $2t$ candidate codewords. By selecting among them the codeword with minimum objective function value, we obtain a heuristic decoder (*Heuristic A* in the following) that always outputs a valid codeword, and has the potential of a better error-correcting performance than pure LP decoding.

A slightly better decoding performance, at the cost of once more increased running time, is reached if we consider not only the paths that constitute the final LP solution but rather *all* intermediate solutions of TC-WS. We call this modification *Heuristic B*.

Simulation results for the $(132, 40)$ LTE code are shown in Figure 9.11. As one can see, the frame error rate indeed drops notably when using the heuristics, but for low SNR values there still remains a substantial gap to the ML decoding curve. At 5 dB, Heuristic B empirically reaches ML performance; for lower SNR values it is comparable to a Log-MAP turbo decoder with 8 iterations.

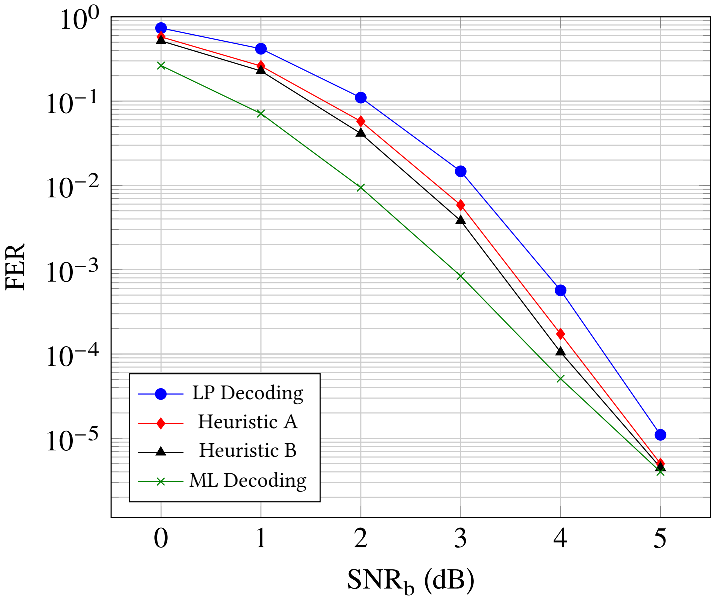


Figure 9.11: Decoding performance of the proposed heuristic enhancements on the $(132, 40)$ LTE turbo code.

9.8 Conclusion and Outlook

We have shown how the inherent combinatorial network-flow structure of turbo codes in form of the trellis graphs can be utilized to construct a highly efficient LP solver, specialized for that class of codes. The decrease in running time, compared to a general purpose solver, is dramatic, and in contrast to classical approaches based on Lagrangian dualization, the algorithm is guaranteed to terminate after a finite number of steps with the exact LP solution.

It is still an open question, however, if and how the LP can be solved in a *completely* combinatorial manner. The nearest point algorithm suggested in this paper introduces a numerical component, which is necessary at this time but rather undesirable since it can lead to numerical problems in high dimension.

Another direction for further research is to examine the usefulness of our decoder as a building block of branch-and-bound methods that solve the integer programming problem, i.e., ML decoders. Several properties of the decoder suggest that this might be a valuable task. For instance, the shortest paths can be computed even faster if a portion of the varibales is fixed, or the algorithm could be terminated prematurely if the reference point exceeds a known upper bound at the current node of the branch-and-bound tree.

Finally, the concepts presented here might be of inner-mathematical interest as well. Optimization problems that are easy to solve in principle but have some complicating constraints are very common in mathematical optimization. Being able to efficiently solve their LP relaxation is a key component of virtually all fast exact or approximate solution algorithms.

Acknowledgments

We would like to acknowledge the German Research Council (DFG), the German Academic Exchange Service (DAAD), and the Center for Mathematical and Computational Modelling (CMCM) of the University of Kaiserslautern for financial support.

References

[1] J. Feldman, D. R. Karger, and M. Wainwright. "Linear programming-based decoding of turbo-like codes and its relation to iterative approaches". In: *Proceedings of the 40th Annual Allerton Conference on Communication, Control and Computing*. Monticello, IL, 2002, pp. 467–477.

[2] J. Feldman. "Decoding error-correcting codes via linear programming". PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 2003.

[3] P. O. Vontobel and R. Koetter. *Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes*. 2005. arXiv: cs/0512078 `[cs.IT]`.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near shannon limit error-correcting coding and decoding: turbo-codes". In: *IEEE International Conference on Communications*. May 1993, pp. 1064–1070. DOI: 10.1109/ICC.1993.397441.

[5] R. G. Gallager. "Low-density parity-check codes". *IRE Transactions on Information Theory* 8.1 (Jan. 1962), pp. 21–28. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057683.

[6] M. Helmling, S. Ruzika, and A. Tanatmis. "Mathematical programming decoding of binary linear codes: theory and algorithms". *IEEE Transactions on Information Theory* 58.7 (July 2012), pp. 4753–4769. DOI: 10.1109/TIT.2012.2191697. arXiv: 1107.3715 `[cs.IT]`.

[7] *IBM ILOG CPLEX Optimization Studio*. Software Package. Version 12.4. 2011.

[8] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[9] A. Tanatmis, S. Ruzika, and F. Kienle. "A Lagrangian relaxation based decoding algorithm for LTE turbo codes". In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing*. Brest, France, Sept. 2010, pp. 369–373. DOI: 10.1109/ISTC.2010.5613906.

[10] S. Ruzika. "On Multiple Objective Combinatorial Optimization". PhD thesis. Kaiserslautern, Germany: University of Kaiserslautern, 2007.

[11] A. Tanatmis. "Mathematical Programming Approaches for Decoding of Binary Linear Codes". PhD thesis. Kaiserslautern, Germany: University of Kaiserslautern, Jan. 2011.

[12] P. Wolfe. "Finding the nearest point in a polytope". *Mathematical Programming* 11 (1 1976), pp. 128–149. ISSN: 0025-5610. DOI: 10.1007/BF01580381.

[13] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. URL: http://www.inference.phy.cam.ac.uk/itprnn/book.html.

[14] *TS 36.212 v11.0.0: LTE E-UTRA Mutliplexing and Channel Coding*. 3rd Generation Partnership Project (3GPP), Oct. 2012. URL: http://www.etsi.org/deliver/etsi_ts/136200_136299/136212/11.00.00_60/ts_136212v110000p.pdf.

[15] S. Lin and D. Costello Jr. *Error Control Coding*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2004. ISBN: 0130426725.

[16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, 1993.

[17] E. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. "On the inherent intractability of certain coding problems". *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 954–972. DOI: 10.1109/TIT.1978.1055873.

[18] A. Tanatmis et al. "Numerical comparison of IP formulations as ML decoders". In: *IEEE International Conference on Communications*. Cape Town, South Africa, May 2010, pp. 1–5. DOI: 10.1109/ICC.2010.5502303.

[19] T. H. Cormen et al. *Introduction to Algorithms*. 2nd ed. MIT Press, 2001.

[20] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization, John Wiley & Sons, 1988.

[21] E. Rosnes, M. Helmling, and A. Graell i Amat. "Pseudocodewords of linear programming decoding of 3-dimensional turbo codes". In: *Proceedings of IEEE International Symposium on Information Theory*. St. Petersburg, Russia, July 2011, pp. 1643–1647. DOI: 10.1109/ISIT.2011.6033823.

[22] M. Helmling and S. Scholl. *Database of ML Simulation Results*. Ed. by University of Kaiserslautern. 2014. URL: http://www.uni-kl.de/channel-codes.

Chapter 10

Paper V: Towards Hardware Implementation of the Simplex Algorithm for LP Decoding

Florian Gensheimer, Michael Helmling, and Stefan Scholl

This chapter is a reformatted revised version of the following publication that was presented at the 2013 (CM)² Young Researcher Symposium and appeared in the refereed conference proceedings:

F. Gensheimer, M. Helmling, and S. Scholl. "Towards hardware implementation of the simplex algorithm for LP decoding". In: *Proceedings of the Young Researcher Symposium.* Center for Mathematical & Computational Modelling (CM)^2. Kaiserslautern, Germany: Fraunhofer, Nov. 2013, pp. 12–17. URL: http://d-nb.info/1044179767

Towards Hardware Implementation of the Simplex Algorithm for LP Decoding

Florian Gensheimer Michael Helmling Stefan Scholl

Combining methods of mathematical optimization theory and applications from communications engineering recently led to new approaches for error correction in data transmission systems. In this new discipline, also called LP decoding, researchers so far focussed on theoretical aspects. In this paper, we study how these new algorithms from mathematical theory can be accelerated using dedicated hardware implementations. We address complexity issues of the widely used simplex algorithms and show how new interesting mathematical problems arise if aspects from hardware design are considered.

10.1 Introduction

Linear programming (LP) is one of the main topics of mathematical optimization. LP problems arise in many economical and industrial areas like production planning, scheduling or logistics. They also play an important role as subproblems in branch-and-bound and cutting-plane algorithms for integer programs, which have a large field of application. The most important algorithm for solving linear programs is the simplex algorithm by G. B. Dantzig (see e.g. [1]). Although the simplex algorithm has an exponential worst-case complexity, it turned out to be very efficient in practice. A rather new application of linear programming is forward error correction or channel coding (see [2, 3] for an introduction).

Channel coding, an engineering discipline, is an essential technique used for correcting errors in digital communication systems that occur during the transmission of data. These transmission errors occur due to bad reception quality, interference between different devices (e.g. mobile phones) or because of physical damage of the devices (storage). Channel coding is used in nearly every communication device today, including smartphones, TV and radio broadcasting, DSL internet access or satellite communications and also in storage devices such as DVDs or USB memory sticks.

A channel coding system consists of two stages: encoder and decoder. The encoder is placed at the transmitter and adds redundant bits before the data is transmitted. The decoder is placed at the receiver and uses the previously added redundancy in order to correct potential transmission errors without the need of retransmission. The decoder is the heart of every channel coding system, since it performs the actual data correction with sophisticated algorithms.

One new approach for error correction decoding is the use of algorithms from the field of mathematical programming: as shown by Feldman *et al.* [4], it is possible to formulate the decoding problem as a linear program which in turn can be solved and analyzed by methods of mathematical optimization. This symbiosis of linear programming from mathematics and channel coding from engineering is called *LP decoding*. LP decoding recently led to new interesting mathematical problems and better algorithms for error correction systems.

Mathematical optimization algorithms are commonly implemented in software and executed on a platform based on a general purpose processor, such as high performance servers, PCs or laptops, that require large space and power. However, many popular communication devices, like smartphones, need to be small and portable and exhibit low power consumption while processing data with high speed.

A general purpose device like a PC processor is designed for great flexibility which allows it to run many different applications. However, if a hardware chip is dedicatedly designed for just one single algorithm, the chip can be highly optimized for exactly this algorithm. In such a case, data operations and memory requirements are well known and the chip architecture can be tailored to the specfic application's requirements.

Implementing algorithms as a dedicated hardware circuit shows several advantages over general purpose hardware:

- speed: speed-ups of several orders of magnitude can be achieved over a PC,
- portability and area: the chip area is often within the region of only several mm^2,
- power: power consumption is often in the order of milliwatts (PC/laptop: several 10 to several 100 Watts).

In the remainder of this paper we consider these dedicated electrical circuits implemented on a chip and call it a *hardware implementation* or simply *hardware*. As we will see later, it is a challenging task to implement an algorithm as hardware, because different aspects of the hardware and the algorithm itself have to be considered.

Hardware implementations for the simplex algorithm and LP decoding have not been investigated deeply so far. In [5], a hardware implementation of the simplex algorithm has been presented. However, this implementation uses the standard simplex method without modifications and does not consider LP decoding. For an efficient hardware implementation it is advantageous to exploit problem specific properties and modifications already at the algorithm level.

As alternatives to the simplex method, interior-point algorithms [6] and a quadratic-programming approach [7] for LP decoding were studied in literature. However, both papers only consider software implementations, and the proposed algorithms are substantially more complex to implement in hardware than the simplex method.

While LP decoding so far has been studied mainly from a theoretical point of view, the aim of the present paper is to investigate how LP decoding algorithms can actually be implemented to work in a communication device. We analyze different variants of the simplex algorithm for LP decoding and reveal large differences in hardware complexity. Additionally, we propose a chip architecture of an LP decoder that can be implemented efficiently.

10.2 The Simplex Algorithm

We start by introducing the fundamentals of linear programming and the simplex method, before reviewing advanced methods based on duality.

A linear program consists of a linear *objective function*, the value of which is to be optimized, and a set of linear *constraints*, i.e. (in)equalities, that limit the values of the variables in the program. Any LP can be written in the standard form

$$\begin{aligned} \min \quad & z = c^T x && (10.1a)\\ \text{s.t.} \quad & Ax = b && (10.1b)\\ & x \geq 0, && (10.1c) \end{aligned}$$

where $x \in \mathbb{R}^n$ are the decision variables, $A \in \mathbb{R}^{m \times n}$ is a matrix with $m \leq n$, and $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ are vectors. A and b describe the so-called functional constraints (10.1b), while (10.1c) defines the nonnegativity constraints. The vector $c \in \mathbb{R}^n$ represents the objective function.

Each row of (10.1b) and (10.1c) defines a *hyperplane* or a *halfspace* of $\mathbb{R}^n$, respectively. Thus, the feasible region $\mathscr{P} = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$ of the LP is a *polyhedron*, i.e., the intersection of a finite number of hyperplanes and halfspaces. For a fixed objective value z, (10.1a) is a hyperplane. Geometrically, the minimization can be interpreted as pushing that hyperplane in direction $-c$ as far as possible without leaving the polyhedron.

The (primal) simplex algorithm is based on the fact that an optimal solution is obtained in a vertex of the polyhedron. The idea of the algorithm is to move iteratively from one vertex of the polyhedron to another in such a way that the sequence of objective function values (10.1a) of the vertices is nonincreasing. Its key observation is that each vertex corresponds to a *basis*, i.e. a linearly independent size-m subset of A's columns. In order to move to an adjacent vertex, exactly one nonbasic column of A is exchanged with a basic column. For each nonbasic column a *reduced cost value* can be computed that tells if adding this column potentially improves the objective value.

In practice, the algorithm operates on the simplex tableau T, a two-dimensional array that contains information about the LP as well as the current basis and reduced costs. Every iteration of the algorithm consists of three main steps:

(1) *computation of the reduced costs:* these determine the column that enters the basis,

(2) *min-ratio rule:* determines the column that leaves the basis,

(3) *basis exchange:* is carried out by a *pivot operation* on $T_{i^*,j}$, i.e., Gaussian elimination step of the form:

```
for $i \in \{1, \dots, m\} \setminus \{i^*\}$ do
    if $T_{i,j} \neq 0$ then
        $T_{i,\bullet} \leftarrow T_{i,\bullet} - \frac{T_{i,j}}{T_{i^*,j}} \cdot T_{i^*,\bullet}$
    end if
end for
```

The pivot operation constitutes the main computational effort of the simplex algorithm. It is therefore crucial to implement this operation efficiently. This operation is repeated in every iteration until the reduced cost vector is nonnegative, which indicates that the vertex of the current basis is optimal.

A second major approach for solving linear programs stems from duality theory. The basis of this theory is the so-called dual program, a special linear program that corresponds to the original LP. For the linear program in standard form (10.1), the dual program (DLP) has the form

$$\max \quad b^T\pi \tag{10.2a}$$

$$\text{s.t.} \quad A^T\pi \leq c, \tag{10.2b}$$

where $\pi \in \mathbb{R}^m$ are the dual variables and $A^T\pi \leq c$ are the dual constraints. Both linear programs (LP) and (DLP) have the same optimal objective value. Hence, one can equivalently solve the dual problem instead of the program (LP).

The *dual simplex algorithm* works on the usual simplex tableau T of (LP). The main difference to the primal simplex algorithm is the fact that the working solution x is infeasible during execution, while the optimality condition is always fulfilled. This is contrary to the primal simplex which always stays feasible, but does not terminate before optimality is established.

In many applications, like e.g. the LP decoding problem, all variables x_i have an upper bound of 1. Both the primal and dual simplex algorithm would have to introduce an artificial variable and an additional side constraint for each such variable because the problem has to be transformed into "standard form" (see e.g. [1]). This leads to a larger simplex tableau and a less efficient simplex algorithm. Special versions of the simplex algorithm (both primal and dual) avoid those extra variables and constraints by implicitly handling upper bounds (see [8, 9]).

Linear programs for practical problems often suffer *degenerate pivot operations*, which means that the step length when moving from one vertex to another reduces to zero, hence the objective function value does not improve. This can dramatically increase the running time and, if occurring frequently, leads to numerical instabilities in the algorithm. Especially for LPs that are highly degenerate—unfortunately, the LP decoding problems was shown to be among those—special care must be taken to avoid degeneracy. We have adapted the so-called *ad-hoc* procedure by P. Wolfe [10] to our case which can be implemented with negligible computational overhead and effectively eliminates the influence of degeneracy for the LP decoding problem.

10.3 Designing Hardware

In this section we present basics and challenges of hardware design. We show how hardware implementations that are highly optimized for one application can outperform general purpose platforms.

As already mentioned in Section 10.1, dedicated hardware implementations are often faster, smaller and consume less power than software implementations running on a general purpose PC. In general, three design goals of hardware implementation can be identified:

- high calculation speed, i.e., high clock frequency,
- small area (leads to reduced production cost and small chip size), and
- small power consumption (to avoid power supply and heat dissipation problems).

To accelerate an algorithm using hardware, it has to be implemented in form of an electrical digital circuit by the hardware designer. Since software (algorithm) and hardware (microchip) development are fundamentally different, this task is not straightforward. In the following we will present some aspects that become important when implementing algorithms in hardware.

10.3.1 Memories for Data Storage

Usually a modern PC provides a large amount of memory (up to several gigabytes) for storing data during the calculations, in order to meet the requirements of a wide range of applications.

In dedicated hardware implementations the memory sizes can be tailored exactly to the application's requirement. This allows for large memory reductions; the resulting memory is often in the order of kilobytes. Additionally, the algorithm itself can be modified to reduce the amount of required storage, finally resulting in a smaller hardware implementation.

10.3.2 Number Representation

Another important aspect is the representation of numerical values that are processed by the algorithm. In a digital system, numbers are represented by vectors of bits. Two fundamental ways to interpret bit vectors as numbers are the *floating-point* and the *fixed-point* representations [11]. On a PC the values are usually represented by double-precision floating-point numbers with a large number of bits (commonly 64). Double-precision floating-point values provide a very good resolution and a wide range. However, floating-point numbers require complex arithmetic hardware, and are therefore avoided in hardware design whenever possible. Instead, it is desired to use fixed-point numbers which lead to a much lower hardware complexity.

A second question arises consequently: How much bits are sufficient to represent the numbers? Usually a low number of bits allows for faster and smaller hardware units and is therefore beneficial (see Figure 10.1). Furthermore, smaller memories can be used for storing data if the number of bits is small. Choosing a low number of bits however leads to a bad resolution of the values and very limited accuracy of the calculations, which may or may not affect the outcome of the algorithm (cf. Table 10.2). There is a clear trade-off between calculation accuracy and hardware complexity (i.e., calculation speed, area and power consumption). This has to be considered when implementing algorithms in hardware. The exact number of required bits in the hardware implementation is mostly determined by extensive simulations.

10.3.3 Arithmetic Units

For arithmetic operations that are carried out in an algorithm, arithmetic hardware units such as adders, multipliers, and so forth have to be implemented. Arithmetic operations have to be used carefully. Assuming fixed-point numbers, additions and subtractions are often cheap to implement. However, multiplications and divisions consume large amounts of hardware area and power and lead to high hardware complexity (see Figure 10.1). Also complex arithmetic operations like logarithms, roots, trigonometric functions etc. exhibit high hardware complexity. Sometimes these functions can be simplified by using approximations such as linearization or simple iterative heuristics. Note that multiplications or divisions by powers of two can be implemented very easily by binary shift operations.

The aspects mentioned so far, although this listing is by no means complete, show important differences between software and hardware design. Therefore, the term "complexity" in software engineering is different from "complexity" in hardware design. When dealing with

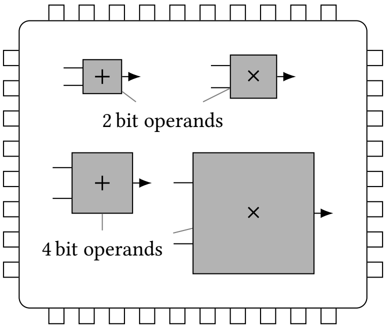


Figure 10.1: Comparison of chip area for adders and multipliers for different numbers of bits of the fixed-point operands.

2-bit represetation	4-bit representation (with decimal point)
10 = −2 11 = −1 00 = 0 01 = 1	100.0 = −4.0 100.1 = −3.5 ⋮ ⋮ 000.0 = 0 000.1 = 0.5 001.0 = 1.0 001.1 = 1.5 ⋮ ⋮ 011.1 = 2.5
4 different levels (resolution = 1)	16 different levels (resolution = 0.5)

Table 10.2: Example for fixed-point numbers using 2 and 4 bits (2s complement), respectively. Note the difference in range and resolution.

hardware implementations, additional and more sophisticated measures of complexity have to be considered.

10.4 Simplex Performance Study

In this section, we evaluate the variants of the simplex algorithm mentioned in Section 10.2:

- the revised primal simplex,
- the revised primal simplex for bounded variables, and
- the dual simplex for bounded variables.

We analyze their running-time performance and their behavior under limited numerical precision with fixed-point arithmetic. The numerical comparisons are done with the $(7, 4)$ Hamming code and a random $(20, 10)$ LDPC code (see [2] for details).

	Primal	Primal (bounded)	Dual (bounded)
		(7,4) Hamming code	
variables	38	31	31
constraints	31	24	24
tableau size	32 × 32	25 × 25	25 × 32
		(20,10) LDPC code	
variables	120	100	100
constraints	100	80	80
tableau size	101 × 101	81 × 81	81 × 101

Table 10.3: Comparison of LP size parameters for the primal simplex and both the primal and dual simplex with bounded variables.

10.4.1 Running-Time Performance

The number of pivot operations, the number of variables and constraints in the LP, and the size of the tableau matrix have major influence on the running-time complexity of the LP solver. The sizes of the LPs for our example decoding problems are shown in Table 10.3. As one can see, the standard primal has the largest size, because it does not handle upper bounds efficiently.

As mentioned in Section 10.2, the simplex iteratively performs pivot steps. The running time is proportional to the number of pivots which may vary for different objective functions. In the channel decoding application, this function is related to the channel noise. Hence, to estimate the running time under realistic conditions, we generate random noise for different signal-to-noise (SNR) ratios; a higher SNR value corresponds to less noise on the channel. In Figure 10.4, the relative frequencies of the iteration counts are shown for the $(7, 4)$ Hamming code. The continuous lines show the values for SNR 0. The performance of the primal simplex variants proved to be independent of the SNR value. Only the dual simplex benefits substantially from a lower noise level, which is indicated by the dotted curve.

As one can see, the dual simplex method dramatically outperforms the primal variants in terms of iteration counts. An explanation for this effect can be found in the initialization procedure of the simplex algorithms: in the primal simplex, always the same, fixed starting basis is used. On average, the corresponding vertex of the polyhedron will be rather far away from the optimal vertex, such that many pivots are necessary. In contrast, the dual simplex for bounded variables [9] chooses an initial solution that would be optimal in the absence of channel noise. Intuitively, this basis is likely to be “closer” to the optimum even if noise is present. Because this starting solution of the dual simplex is not primal feasible, it cannot be used to speed up the primal algorithms.

Figure 10.5 shows for the larger code how the average number of iterations per instance depends on the noise level. Again, the dual simplex is much faster, finishing after 1.66 iterations on

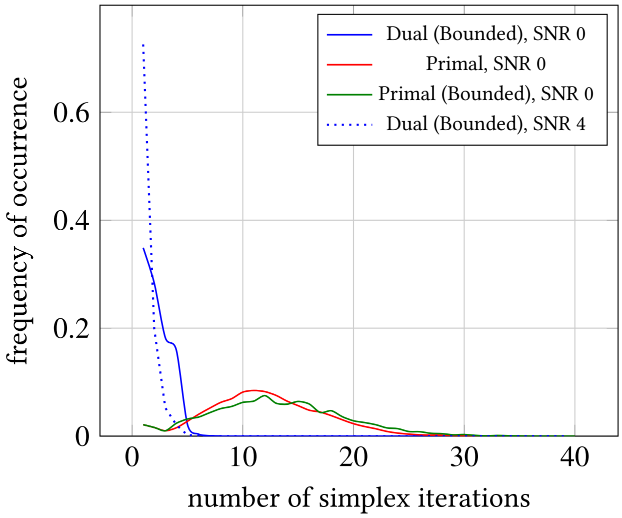


Figure 10.4: Number of iterations until optimality is reached for different simplex variants, using the decoding LP for the $(7, 4)$ Hamming code.

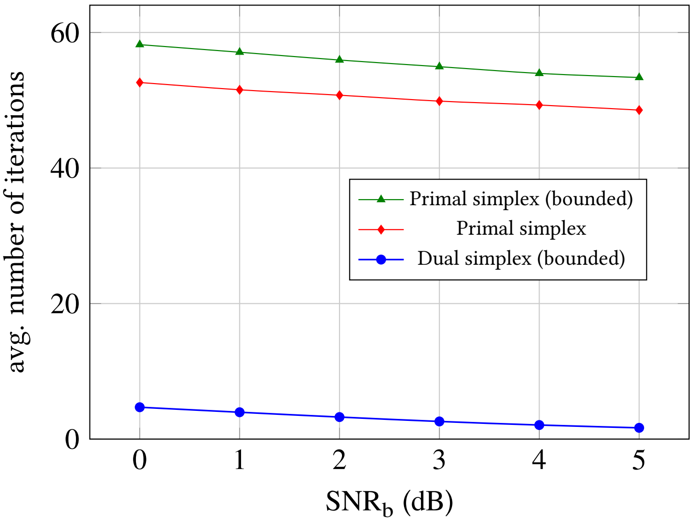


Figure 10.5: Comparison of the average number of iterations per LP between the three different simplex variants on the $(20, 10)$ LDPC code.

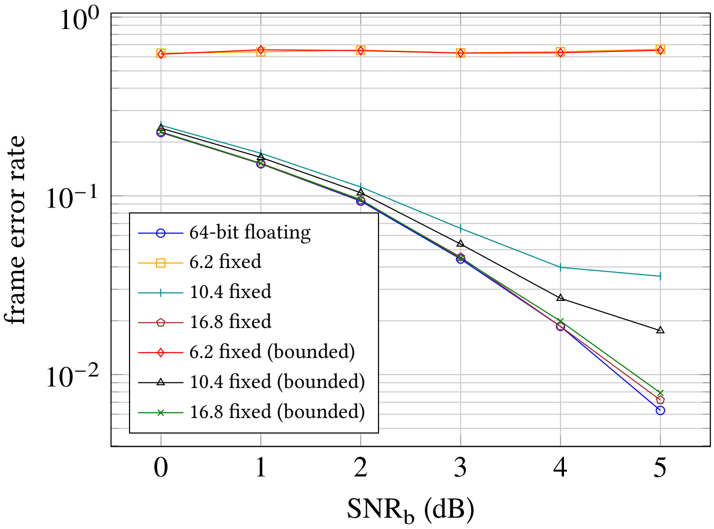


Figure 10.6: Decoding performance of the primal simplex algorithms with fixed-point arithmetics (standard and bounded variables, respectively) on the Hamming code. The dual algorithms are not shown here because their decoding performance did not change compared to floating-point numbers.

average for the highest SNR value.

10.4.2 Comparison of Fixed- and Floating-Point Implementation

As outlined in Section 10.3, using fixed-point numbers with low precision is preferable from the hardware point of view. On the other hand, if the numerical precision is too low, round-off errors can break the algorithm in numerous ways, leading to wrong solutions or even infinite loops. Hence a compromise needs to be found between simplicity and correctness. For the LP decoding application, the latter directly translates into the frame error rate, i.e., the average probability of a decoding error. Depending on the usage scenario, a slightly increased error rate might be tolerable, if this in return allows for low-complexity hardware.

For the $(7,4)$ Hamming code, we compare fixed-point arithmetic with resolutions 16.8, 10.4, and 6.2 with usual double-precision floating-points. Here, the notation $x.y$ means that each number is represented by x bits, y of which constitute the fractional part. This implies that numbers from -2^{x-y-1} to $2^{x-y-1} - 2^{-y}$ can be represented, with a resolution of 2^{-y}.

In our experiments, the dual simplex proves extremely stable, showing practically the same error-correcting performance even with the very poor 6.2 resolution. The primal variants achieve this performance with 16.8; with 10.4 bits the performance drops but is still reasonable, while both primal algorithms are practically useless with the lowest resolution. The resulting error-performance curves for the primal variants are shown in Figure 10.6.

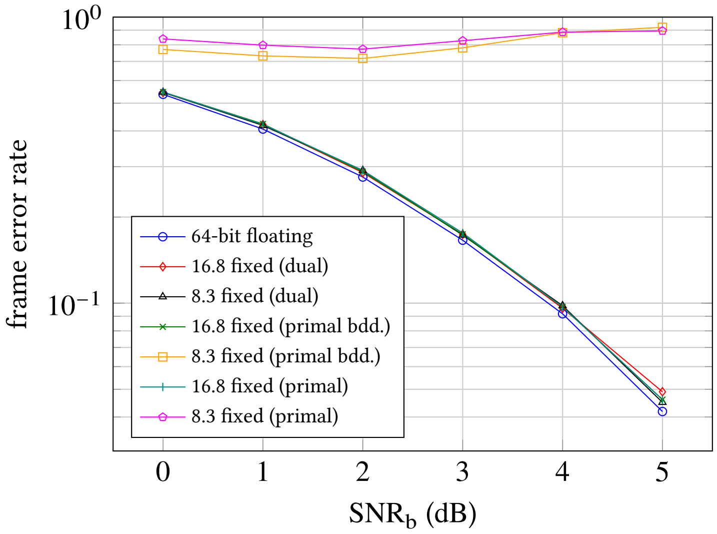


Figure 10.7: Decoding performance of the primal and dual simplex algorithms on the LDPC code with different number represenions.

For the larger code, we compare the floating-point implementation with fixed-point precisions 16.8 and 8.3. Results are shown in Figure 10.7. Again, the dual algorithm is most stable with low precision and practically achieves floating-point performance with 8.3 fixed-point bits. The primal algorithms significantly loose performance with this resolution, but are comparable to the floating-point performance with 16.8.

For both codes, the two primal simplex variants largely show the same behavior, whereas the dual simplex for bounded variables performs clearly superior, in terms of both the number of iterations (and thus overall running time) and the robustness against low-precision numerical resolution.

10.4.3 Proposed Hardware Architecture

In Figure 10.8 we propose a hardware architecture that implements the dual simplex for bounded variables.

It shows the five main parts required by the simplex. The current tableau is stored in the tableau memory. A pivot unit accomplishes pivoting of a column on the fly when it is read. It is followed by an additional unit that performs simple bit flip operations to take care of the variable bounds. The entering variable is determined according to some pricing rule and checks for optimality, i.e., when the simplex has to stop. The leaving variable block is needed to compute the leaving variable using e.g. the min-ratio rule.

Additionally a controller is shown. The controller takes care of the initialization of the other hardware units, keeps track of the basis variable positions, and extracts the values of the

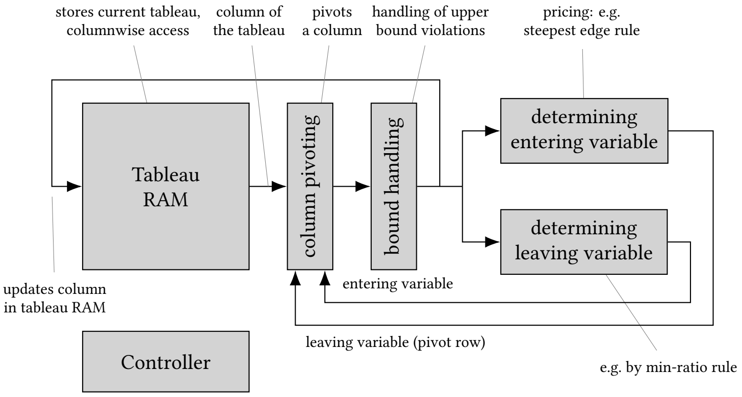

stores current tableau, columnwise access
column of the tableau
pivots a column
handling of upper bound violations
pricing: e.g. steepest edge rule
Tableau RAM
column pivoting
bound handling
determining entering variable
determining leaving variable
updates column in tableau RAM
entering variable
leaving variable (pivot row)
Controller
e.g. by min-ratio rule

Figure 10.8: Proposed hardware architecture for the dual simplex algorithm.

variables when the result is declared optimal.

10.5 Conclusion

In this paper, we have studied the LP decoding problem with three variants of the simplex algorithm. In terms of running time, the dual simplex for bounded variables has shown to outperform the competiting primal variants, finding the optimal solution substantially faster (up to a factor of 50). With respect to hardware implementation complexity, we have also analyzed the behavior of these algorithms under low-precision fixed-point arithmetic. Again, the dual simplex has shown to be most stable, achieving a similar error-correction level as the 64-bit floating-point implementation with 6 and 8 fixed-point bits per number for the Hamming and LDPC code, respectively.

Both the number of iterations and the low size of the number representation are important steps towards a low-power, high-performance hardware LP decoder implementation.

10.6 Outlook and Future Research

Other improvements of the simplex algorithm, specialized for hardware requirements, might further decrease the complexity of hardware LP decoding. Some possible directions are:

10.6.1 Simplex in the Log-Domain

The simplex algorithm usually involves a lot of multiplications and division. However these two operations are costly to implement in hardware, as we have shown in Section 10.3.

One option to transform multiplication and divison into a much simpler operation is to perform calculations in the logarithm domain. If the logarithm of all operands is used, a multiplication transforms into an addition and a division into a simple subtraction, thus avoiding costly operations. However this avoidance comes at additional cost for taking logarithm and the exponential function. Further investigations have to show how far the hardware complexity can be reduced by this approach.

10.6.2 Adaptive LP Decoding

For larger codes, the complete LP decoding formulation contains a large number of constraints, most of which are not actually needed to describe the optimal solution. Taghavi and Siegel [12] have proposed an adaptive method that starts with an empty LP and inserts inequalities on demand, greatly reducing the size of the number of constraints. Two aspects make this approach particularly appealing: Firstly, as shown in [12] it is possible to upper bound the number of necessary constraints to a very small number, compared to the complete LP. Secondly, inserting additional constraints is possible without any extra operations if one uses the dual simplex algorithm. Since the dual simplex algorithm turned out as the most efficient one by our numerical study in Section 10.4, using it for adaptive LP decoding is extremely promising.

10.6.3 Polyhedral Theory for Fixed-Point Numbers

Mathematically, fixed-point numbers constitute a uniform, discrete grid in the Euclidean space. Ordinary polyhedral theory always assumes a continuous space, thus real numbers with infinite precision. A theoretical study of “discrete” polyhedral theory could, besides being an interesting field of research on its own, lead to more efficient algorithms exploiting the grid structure. Additionally, it might be possible to find theoretical results about numerical stability for given fixed-point precision.

Acknowledgment

We gratefully acknowledge the Center of Mathematical and Computational Modelling of the University of Kaiserslautern and the German Research Council (DFG) for financial support.

References

[1] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[2] S. Lin and D. Costello Jr. *Error Control Coding*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2004. ISBN: 0130426725.

[3] M. Helmling, S. Scholl, and A. Tanatmis. “Mathematical optimization based channel coding: current achievements and future research”. In: *Proceedings of the Young Researcher Symposium.* Center for Mathematical & Computational Modelling (CM)^2. Kaiserslautern, Feb. 2011, pp. 16–21. URL: http://nbn-resolving.de/urn:nbn:de:0074-750-0.

[4] J. Feldman, M. J. Wainwright, and D. R. Karger. “Using linear programming to decode binary linear codes”. *IEEE Transactions on Information Theory* 51.3 (Mar. 2005), pp. 954–972. DOI: 10.1109/TIT.2004.842696. URL: www.eecs.berkeley.edu/~wainwrig/Papers/FelWaiKar05.pdf.

[5] S. Bayliss et al. “An FPGA implementation of the simplex algorithm”. In: *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT).* Bangkok, Thailand, Dec. 2006, pp. 49–56. DOI: 10.1109/FPT.2006.270294.

[6] T. Wadayama. “An LP decoding algorithm based on primal path-following interior point method”. In: *Proceedings of IEEE International Symposium on Information Theory.* Seoul, Korea, June 2009, pp. 389–393. DOI: 10.1109/ISIT.2009.5205741.

[7] K. Yang, X. Wang, and J. Feldman. “A new linear programming approach to decoding linear block codes”. *IEEE Transactions on Information Theory* 54.3 (Mar. 2008), pp. 1061–1072. DOI: 10.1109/TIT.2007.915712.

[8] H. W. Hamacher and K. Klamroth. *Lineare und Netzwerk-Optimierung.* 2nd ed. Vieweg, Apr. 2006. ISBN: 978-3-528-03155-8.

[9] H. M. Wagner. “The dual simplex algorithm for bounded variables”. *Naval Research Logistics Quarterly* 5.3 (1958), pp. 257–261. ISSN: 1931-9193. DOI: 10.1002/nav.3800050306.

[10] P. Wolfe. “A technique for resolving degeneracy in linear programming”. *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pp. 205–211. DOI: 10.1137/0111016. eprint: http://epubs.siam.org/doi/pdf/10.1137/0111016.

[11] D. A. Patterson and J. Hennessy. *Computer Organization and Design.* 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004. ISBN: 1558606041.

[12] M. H. Taghavi and P. H. Siegel. “Adaptive methods for linear programming decoding”. *IEEE Transactions on Information Theory* 54.12 (Dec. 2008), pp. 5396–5410. DOI: 10.1109/TIT.2008.2006384. arXiv: cs/0703123 `[cs.IT]`.

Chapter 11

Paper VI: Efficient Maximum-Likelihood Decoding of Linear Block Codes on Binary Memoryless Channels

Michael Helmling, Eirik Rosnes, Stefan Ruzika, and Stefan Scholl

The following chapter is a reformatted and revised copy of a preprint that is publicly available online (http://arxiv.org/abs/1403.4118). Its contents were presented at the 2014 ISIT conference and appeared in the following refereed conference proceedings:

M. Helmling, E. Rosnes, S. Ruzika, and S. Scholl. "Efficient maximum-likelihood decoding of linear block codes on binary memoryless channels". In: *Proceedings of IEEE International Symposium on Information Theory.* Honolulu, HI, USA, June–July 2014, pp. 2589–2593. DOI: 10.1109/ISIT.2014.6875302. arXiv: 1403.4118 `[cs.IT]`

This work was partially funded by the DFG (grant RU-1524/2-1) and by DAAD / RCN (grant 54565400 within the German-Norwegian Collaborative Research Support Scheme).

Efficient Maximum-Likelihood Decoding of Linear Block Codes on Binary Memoryless Channels

Michael Helmling
Eirik Rosnes

Stefan Ruzika
Stefan Scholl

In this work, we consider efficient maximum-likelihood decoding of linear block codes for small-to-moderate block lengths. The presented approach is a branch-and-bound algorithm using the cutting-plane approach of Zhang and Siegel (*IEEE Trans. Inf. Theory*, 2012) for obtaining lower bounds. We have compared our proposed algorithm to the state-of-the-art commercial integer program solver CPLEX, and for all considered codes our approach is faster for both low and high signal-to-noise ratios. For instance, for the benchmark $(155, 64)$ Tanner code our algorithm is more than 11 times as fast as CPLEX for an SNR of 1.0 dB on the additive white Gaussian noise channel. By a small modification, our algorithm can be used to calculate the minimum distance, which we have again verified to be much faster than using the CPLEX solver.

11.1 Introduction

Determining the optimal decoding behavior of error-correcting codes is of significant importance, e. g., to benchmark different coding schemes. When no *a priori* information on the transmitted codeword is known, maximum-likelihood decoding (MLD) is an optimal decoding strategy. It is known that this problem is NP-hard in general [1] such that its complexity grows exponentially in the block length of the code, unless P=NP. Currently, the best known approach for general block codes is to use a state-of-the-art (commercial) integer program (IP) solver (see [2]) like CPLEX [3].

In this work, we present a branch-and-bound approach for efficient MLD of linear block codes. The problem of MLD is closely related to that of calculating the minimum distance of a code, which has attracted some attention recently. For instance, in [4, 5], Rosnes *et al.* proposed an efficient branch-and-bound algorithm to determine all low-weight stopping sets/codewords in a low-density parity-check (LDPC) code. Although the two problems are similar, the bounding step in the algorithm from [4, 5] cannot efficiently be adapted to the scenario of MLD. Conversely, however, the algorithm presented here can also calculate the minimum distance, and our numerical experiments show that this is very efficient compared to CPLEX.

Linear programming (LP) decoding of binary linear codes, as first introduced by Feldman *et al.* in [6], approximates MLD by relaxing the decoding problem into an easier to solve LP problem. The LP problem contains a set of linear inequalities that are derived from the parity-check constraints of a (redundant) parity-check matrix representing the code. As shown in [7], these constraints can iteratively and adaptively be added to the decoding problem, which significantly reduces the overall complexity of LP decoding. Elaborating on this idea, Zhang and Siegel [8] proposed an efficient search algorithm for new *violated* (redundant) parity-check constraints (or "cuts") that tighten the decoding polytope. Depending on the structure of the underlying code, for some codes, this "cutting-plane" LP decoding algorithm performs close to MLD for high signal-to-noise ratios (SNRs) on the additive white Gaussian noise (AWGN) channel, although for most codes, e. g., the $(155, 64)$ Tanner code [9], there is still a gap in decoding performance to MLD [8]. For lower values of the SNR, there could be a significant performance degradation with respect to MLD.

The algorithm proposed in this work closes that gap by using the cutting-plane algorithm for lower bounds and the well-known sum-product (SP) decoder [10] with order-i re-encoding [11] for upper bounds within a sophisticated branch-and-bound framework such that the output always and provably is the maximum-likelihood (ML) codeword. Our numerical study in Section 11.6 shows that it is much faster than CPLEX for all codes under consideration, and moreover is able to decode some of the codes on which CPLEX fails completely.

11.2 Notation and Background

This section establishes some basic definitions and results needed for the rest of the paper.

Let $\mathscr{C}$ denote a binary linear code of length n represented by an $m \times n$ parity-check matrix $\mathbf{H}$. The code is used on a binary-input memoryless output-symmetric channel with input $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathscr{C}$ and channel output denoted by the length-n vector $\mathbf{r} = (r_0, \ldots, r_{n-1})$. The ML decoder can be described by the following optimization problem [12]:

$$\hat{\mathbf{c}}_{\mathrm{ML}} = \operatorname*{arg\,min}_{\mathbf{c} \in \mathscr{C}} \psi_{\boldsymbol{\lambda}}(\mathbf{c}) = \operatorname*{arg\,min}_{\mathbf{c} \in \mathrm{conv}(\mathscr{C})} \psi_{\boldsymbol{\lambda}}(\mathbf{c}) \tag{11.1}$$

where $\psi_{\boldsymbol{\lambda}}(\mathbf{c}) = \boldsymbol{\lambda} \cdot \mathbf{c}^T$ and $(\cdot)^T$ denotes the transpose of its argument, $\boldsymbol{\lambda} = (\lambda_0, \ldots, \lambda_{n-1})$ is a vector of log-likelihood ratios (LLRs) defined by

$$\lambda_i = \log\left(\frac{\Pr(r_i \mid c_i = 0)}{\Pr(r_i \mid c_i = 1)}\right)$$

for all i, $0 \leq i \leq n-1$, and $\mathrm{conv}(\mathscr{C})$ is the *convex hull* of $\mathscr{C}$ in $\mathbb{R}^n$, where $\mathbb{R}$ denotes the real numbers. The MLD problem in (11.1) can be formulated as an IP which in general is an NP-hard problem. As an approximation to MLD, Feldman *et al.* [6] relaxed the codeword polytope $\mathrm{conv}(\mathscr{C})$ in the following way.

Define

$$\mathscr{C}_j = \{\mathbf{c} \in \{0,1\}^n \colon \mathbf{h}_j \cdot \mathbf{c}^T = 0\}$$

where $\mathbf{h}_j = (h_{j,0}, \ldots, h_{j,n-1})$ is the jth row of the parity-check matrix $\mathbf{H}$ and $0 \leq j \leq m-1$. Furthermore, let $\mathrm{conv}(\mathscr{C}_j)$ denote the convex hull of $\mathscr{C}_j$ in $\mathbb{R}^n$. The *fundamental polytope* $\mathscr{P}(\mathbf{H})$ of the parity-check matrix $\mathbf{H}$ is defined as [13]

$$\mathscr{P}(\mathbf{H}) = \bigcap_{j=0}^{m-1} \mathrm{conv}(\mathscr{C}_j). \tag{11.2}$$

The MLD problem in (11.1) can now be relaxed to

$$\hat{\mathbf{p}}_{\mathrm{LP}} = \operatorname*{arg\,min}_{\mathbf{p} \in \mathscr{P}(\mathbf{H})} \psi_{\boldsymbol{\lambda}}(\mathbf{p})$$

where the solution, by definition, is a *pseudocodeword* with fractional entries in general. Note that the LP decoder has the *ML certificate* property, which means that in case $\hat{\mathbf{p}}_{\mathrm{LP}}$ is integral, it is an optimal solution to (11.1).

For each row $\mathbf{h}_j$, $0 \leq j \leq m-1$, in the matrix $\mathbf{H}$ the linear inequalities behind the fundamental polytope in (11.2) are

$$\sum_{i \in \mathscr{V}} p_i - \sum_{i \in \mathscr{N}(j) \setminus \mathscr{V}} p_i \leq |\mathscr{V}| - 1, \text{ for all odd-sized } \mathscr{V} \subseteq \mathscr{N}(j) \tag{11.3}$$

where $\mathscr{N}(j) = \{i \colon h_{j,i} = 1,\ 0 \leq i \leq n-1\}$.

For a given row $\mathbf{h}_j$ of a parity-check matrix $\mathbf{H}$ and a vector $\mathbf{p} \in [0,1]^n$: If there exists an odd set $\mathscr{V} \subseteq \mathscr{N}(j)$ such that the corresponding inequality from (11.3) does not hold, then we say that the jth parity-check constraint induces a *cut* at $\mathbf{p}$.

Central to our branch-and-bound algorithm is the concept of *constraint sets.* A constraint set F is a set

$$\{(\rho_i, c_{\rho_i}) \colon c_{\rho_i} \in \{0,1\} \; \forall \rho_i \in \Gamma\},$$

where $\Gamma \subseteq \{0,\dots,n-1\}$. If (ρ_i, c_{ρ_i}) is a constraint, then position ρ_i is said to be c_{ρ_i}-constrained, which means that position ρ_i is committed to the value c_{ρ_i} in a codeword, while positions not in F are uncommitted.

Let $\mathscr{C}^{(F)}$ denote the subset of codewords from $\mathscr{C}$ consistent with the constraint set F. Then, we can define

$$\psi^{(F)}_{\min,\lambda} = \min_{\mathbf{c} \in \mathscr{C}^{(F)}} \psi_\lambda(\mathbf{c})$$

as the minimum value of the objective function for codewords consistent with F. In the following, $\bar{\psi}^{(F)}_{\min,\lambda}$ will denote any lower bound on $\psi^{(F)}_{\min,\lambda}$. Also, a constraint set F is said to be *valid* if $\mathscr{C}^{(F)}$ is nonempty.

Our proposed branch-and-bound MLD algorithm relies heavily on tight lower bounds on $\psi^{(F)}_{\min,\lambda}$, which are provided by an LP-based decoding algorithm by Zhang and Siegel [8]. We briefly describe this algorithm, denoted as the ZS decoding algorithm, below in Section 11.2.1.

11.2.1 Zhang and Siegel's LP-Based Decoding Algorithm

The ZS decoding algorithm is based on adaptive LP decoding as described in [7] and incorporates an efficient cut-search algorithm, as described in [8]. First the LP problem is initialized with the box constraints. †Solve the LP problem to get an optimal solution $\mathbf{p}^*$. If $\mathbf{p}^*$ is integral, then terminate the algorithm and return the ML codeword $\mathbf{p}^*$. Otherwise, the cut-search algorithm (Algorithm 1 in [8]) is applied to each row of the parity-check matrix of the code. If at least one is found, add all found cuts into the LP problem and repeat the procedure from †. Otherwise, search for cuts from redundant parity-checks. To this end, reduce $\mathbf{H}$ by Gaussian elimination to *reduced row echelon* form, where the columns of $\mathbf{H}$ are processed in the order of the "fractionality" (i. e., closeness to $1/2$) of the corresponding coordinate of $\mathbf{p}^*$. Now, the cut-search algorithm is applied to each row of the obtained modified matrix $\tilde{\mathbf{H}}$. If no cut is found, then terminate. Otherwise, add all found cuts into the LP problem as constraints and repeat the procedure from †. The algorithm above has been detailed in Algorithm 2 in [8], and we refer the interested reader to [8] for further details.

11.3 Basic Branch-and-Bound Algorithm

Our proposed algorithm is a branch-and-bound algorithm on constraint sets and uses the ZS decoding algorithm, as briefly described above, as a basic component in the bounding step.

Thus, there is a one-to-one correspondence between the nodes in the search tree and constraint sets. In the following, when we speak about the *left* and *right* child constraint set, denoted by $F^{\downarrow 0}$ and $F^{\downarrow 1}$, respectively, we mean the constraint set of the left and right child nodes in the search tree.

Now, to each constraint set F, we associate three real numbers

$$\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}},\ \bar{\psi}^{(F)\downarrow 0}_{\min,\boldsymbol{\lambda}},\ \text{and}\ \bar{\psi}^{(F)\downarrow 1}_{\min,\boldsymbol{\lambda}}$$

which are *current* lower bounds on

$$\psi^{(F)}_{\min,\boldsymbol{\lambda}},\ \psi^{(F^{\downarrow 0})}_{\min,\boldsymbol{\lambda}},\ \text{and}\ \psi^{(F^{\downarrow 1})}_{\min,\boldsymbol{\lambda}},$$

respectively. When a constraint set is created, these values are initiated to $-\infty$.

The algorithm maintains a list L of active constraint sets which is initiated with the unconstrained set $\emptyset$. In each iteration, a constraint set F is selected from the list according to the node selection rule (see below). A valid codeword, i. e., a feasible solution of the IP, is generated by decoding the LLR vector (where the constraints imposed by F are enforced by altering the respective LLR values to $\pm\infty$) using the SP algorithm [10] with order-i re-encoding [11], for some integer i, as a post-processing step. The upper bound on the objective function decreases if the decoder output has a lower objective function value than the previous best candidate. Afterwards a lower bound on $\psi^{(F)}_{\min,\boldsymbol{\lambda}}$ is computed by running the ZS algorithm, where the variables contained in F are fixed to their corresponding values. If an integral solution is returned, i. e., a pseudocodeword with no fractional coordinates, it is considered another candidate codeword in the same way as above. Otherwise, and if the computed lower bound is less than the current upper bound τ, the algorithm branches on a fractional position, selected by the branching rule (see below), of the pseudocodeword by adding two constraint sets, namely F augmented by the chosen branching position fixed to 0 and 1, respectively, to the list of active nodes. Then, the next set is chosen from L until one of the termination criteria in Step 2 of Algorithm 11.1 on page 173 (which gives a formal description of the overall algorithm) is fulfilled. Note that the computations to produce a lower bound on $\psi^{(F)}_{\min,\boldsymbol{\lambda}}$ for a given constraint set F are collected into Algorithm 11.2, denoted by LUBD.

11.3.1 Bounding Step

The complexity of Algorithm 11.1 depends heavily on the tightness of the lower bounds computed in Step 5 (from Algorithm 11.2), i. e., on how close $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$ is to the value $\psi^{(F)}_{\min,\boldsymbol{\lambda}}$. To find the best pseudocodeword $\hat{\mathbf{p}}$, we have used the procedure detailed in Algorithm 11.2.

Note that

$$\min\{\psi^{(F\downarrow 0)}_{\min,\boldsymbol{\lambda}}, \psi^{(F\downarrow 1)}_{\min,\boldsymbol{\lambda}}\} = \psi^{(F)}_{\min,\boldsymbol{\lambda}}$$

for any node (constraint set) F. This allows us to update the current lower bound $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}}$ of F (and, recursively, also the ancestors of F), potentially *increasing* its value, once both of its children have been processed (see Steps 18 to 22 of Algorithm 11.1). Tightening the bounds in the search tree is important for decreasing the complexity of the algorithm because nodes whose lower bound exceeds the objective value of the currently best candidate solution (i.e., the current upper bound) can be skipped, thereby reducing the search space.

11.3.2 Branching Step

We have used the following simple branching rule to select the position p in Step 15 of Algorithm 11.1: Take an unconstrained position where the corresponding entry in the decoded pseudocodeword $\hat{\mathbf{p}}$ is closest to $1/2$. This simple procedure seems to work very well in practice.

11.3.3 The Processing Order of the List L

The node selection rule, i. e., the method by which a constraint set F is selected from L in Step 3 of Algorithm 11.1, has great influence on the overall complexity. The most common schemes are *depth-first search*, according to LIFO (last in – first out) processing of L, and *breadth-first search*, where L is processed in FIFO (first in – first out) fashion. Another popular method, called *best-bound search*, selects the next constraint set by $F' = \arg\min_{F \in L} \bar{\psi}^{(F)}_{\min,\lambda}$, with the goal of tightening the overall lower bound as fast as possible.

In our experiments, the following mixed strategy has proven to be most efficient. Apply depth-first processing in general, but every M iterations, for a fixed integer M, and only if $\bar{\psi}^{(F)}_{\min,\lambda} < \tau - \delta$ for a fixed $\delta > 0$, where F is the constraint set from the previous iteration, select the next node by the best-bound rule above.

11.4 Improvements

In this section, we present some improvements to the basic algorithm from Section 11.3.

11.4.1 Tuning the ZS Algorithm for Adaptive LP Decoding

A linear inequality constraint of the general form $\mathbf{a} \cdot \mathbf{x}^T \leq b$, where $\mathbf{a}$ and b are constants, is called *active* at the point $\mathbf{x}^*$ if it holds with equality for $\mathbf{x} = \mathbf{x}^*$. Otherwise, it is called *inactive*. For an LP problem with a set of linear inequality constraints, the optimal solution $\mathbf{x}_{\text{LP}}$ is a *vertex* of the polytope formed by the hyperplanes of all constraints that are active at $\mathbf{x}_{\text{LP}}$. Thus, constraints inactive at $\mathbf{x}_{\text{LP}}$ can be removed without changing the optimal solution.

The ZS decoding algorithm uses adaptive LP decoding, which implies that a lot of linear programs (of increasing size in the number of constraints) are solved successively. Consequently, a simple way to reduce the overall complexity is to remove inactive constraints from time to time.

Algorithm 11.1 Maximum-Likelihood Decoding (MLD)

Input: The received LLR vector $\boldsymbol{\lambda}$ and the order i of re-encoding.
Output: An ML decoded codeword $\hat{\mathbf{c}}_{\mathrm{ML}}$.

```
1:  Initialize $\tau \leftarrow \infty$ and $L \leftarrow \{\emptyset\}$
2:  while $L \neq \emptyset$ and $\bar{\psi}^{(\emptyset)}_{\min,\boldsymbol{\lambda}} < \tau$ do
3:      Choose and remove a constraint set $F$ from $L$.
4:      if $F$ is valid and $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}} < \tau$ then
5:          let $(\hat{\mathbf{c}}, \hat{\mathbf{p}}) \leftarrow \mathrm{LUBD}(\boldsymbol{\lambda}, F)$
6:          if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{c}}) < \tau$ then
7:              let $\hat{\mathbf{c}}_{\mathrm{ML}} \leftarrow \hat{\mathbf{c}}$ and $\tau \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{c}})$
8:          end if
9:          $\bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}} \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$
10:         if $\hat{\mathbf{p}}$ is integral then
11:             if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}}) < \tau$ then
12:                 let $\hat{\mathbf{c}}_{\mathrm{ML}} \leftarrow \hat{\mathbf{p}}$ and $\tau \leftarrow \psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}})$
13:             end if
14:         else if $\psi_{\boldsymbol{\lambda}}(\hat{\mathbf{p}}) < \tau$ then
15:             choose an unconstrained position $p$ based on $\hat{\mathbf{p}}$, construct two new constraint sets $F' = F \cup \{(p, 0)\}$ and $F'' = F \cup \{(p, 1)\}$, and append them to $L$.
16:         end if
17:     end if
18:     if $F \neq \emptyset$ then
19:         determine (the unique) $\tilde{F}$ such that $F = \tilde{F}^{\downarrow i}$, where $i = 0$ or $1$, and update parent bounds as follows:
20:         $\bar{\psi}^{(\tilde{F})\downarrow i}_{\min,\boldsymbol{\lambda}} \leftarrow \max\{\bar{\psi}^{(\tilde{F})\downarrow i}_{\min,\boldsymbol{\lambda}}, \bar{\psi}^{(F)}_{\min,\boldsymbol{\lambda}}\}$
21:         $\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}} \leftarrow \max\{\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}}, \min\{\bar{\psi}^{(\tilde{F})\downarrow 0}_{\min,\boldsymbol{\lambda}}, \bar{\psi}^{(\tilde{F})\downarrow 1}_{\min,\boldsymbol{\lambda}}\}\}$
22:         If $\bar{\psi}^{(\tilde{F})}_{\min,\boldsymbol{\lambda}}$ increased in the previous step, recurse to Step 18 with $F$ replaced by $\tilde{F}$.
23:     end if
24: end while
25: Return $\hat{\mathbf{c}}_{\mathrm{ML}}$.
```

Algorithm 11.2 Lower and Upper Bound Algorithm (LUBD)

```
Input: The received LLR vector $\boldsymbol{\lambda}$ and a constraint set $F$.
Output: The pair $(\hat{\mathbf{c}}, \hat{\mathbf{p}})$.
Perform SP decoding with order-$i$ re-encoding on an LLR vector constrained according to $F$ as follows:
    • $+\infty$ for positions corresponding to 0-constraints.
    • $-\infty$ for positions corresponding to 1-constraints.
    • The original channel LLRs for positions not in $F$.
  The resulting decoded codeword is denoted by $\hat{\mathbf{c}}$.
Perform ZS decoding on the received LLR vector $\boldsymbol{\lambda}$ with equality constraints according to $F$. Denote the decoded pseudocodeword by $\hat{\mathbf{p}}$.
Return the pair $(\hat{\mathbf{c}}, \hat{\mathbf{p}})$.
```

Our implementation uses the *dual simplex method* for solving LP problems, which is very effective in the case of iteratively added constraints by employing a warm-start technique that reuses the basis information of the previously optimal solution (see, e. g., [14] for details). The removal of constraints, however, is expensive because afterwards a new simplex basis has to be computed. Thus, we remove the inactive constrains only when the number of constraints in the current LP problem exceeds T, for some integer T. Note that this differs from the algorithm called MALP-B in [8], where the inactive constraints are removed in each iteration.

Another way to decrease the running-time of the ZS algorithm in some cases is to terminate the ZS decoder prematurely as soon as the objective value exceeds the current upper bound τ in Algorithm 11.1. In that event the objective function value cannot possibly be improved below the current node, and it can be skipped immediately.

11.4.2 Tradeoff Between Tightness and Speed of the ZS Algorithm

The cut-search procedure used in the ZS decoding algorithm yields tight lower bounds on the MLD solution at the cost of a high number of cuts and thus increased processing time spent in the LP solver and for Gaussian elimination (see Section 11.2.1). In two different ways, a tradeoff between speed and tightness can be realized. First, limit the maximum number of times R that the search for redundant parity-check cuts is applied, and secondly, only add a cut if the *cutoff*, i. e., the distance between the current (infeasible) solution and the cutting hyperplane, exceeds a fixed quantity $\gamma > 0$.

Our numerical experiments have shown that both approaches help to significantly reduce the running-time complexity of our algorithm. Additionally, for the first approach, it has proven helpful to use a higher value R^{bb} in those iterations where a best-bound node has been selected (cf. Section 11.3.3).

decoder	1.0 dB		1.5 dB		2.0 dB		2.5 dB		3.0 dB		3.5 dB	
	T_{avg}	N_{avg}	T_{avg}	N_{avg}	T_{avg}	N_{avg}	T_{avg}	N_{avg}	T_{avg}	N_{avg}	T_{avg}	N_{avg}
(155,64) Tanner code [9]												
our alg.	0.81	51	0.24	15	0.05	3.5	0.014	1.4	0.005	1.04	0.004	1.004
CPLEX	9.49	4795	2.95	1816	0.63	370	0.17	61	0.095	5.4	0.086	0.3
(155,64) Tanner code [9] with all-zero decoding												
our alg.	0.24	14	0.11	6.6	0.025	2.1	0.006	1.18	0.001	1.05	0.0005	1.002
CPLEX	3.23	2799	1.16	963	0.28	210	0.06	38	0.02	4.5	0.01	0.3
(204,102) MacKay code [15] with all-zero decoding												
our alg.	2.2	9	0.73	30.5	0.15	6.5	0.02	1.66	0.003	1.04	0.0005	1.003
CPLEX	14.6	12364	4.7	3421	0.83	573	0.12	61	0.03	4.9	0.018	0.19
(127,85) BCH code with all-zero decoding												
our alg.	86	7617	67	5855	33	2549	9	655	2.2	159	0.29	19
CPLEX											3.5	4132

Table 11.1: Numerical comparison of our proposed algorithm and CPLEX for several codes using different values of the SNR on the AWGN channel. The number T_{avg} is the average decoding CPU time per frame in seconds, and N_{avg} is the average number of nodes (per frame) processed by the branch-and-bound algorithms. In all cases but the first we used all-zero decoding as described in Section 11.4.3.

11.4.3 Special Case: MLD Performance Simulation

For benchmarking purposes we are only interested in the actual MLD curve, in which case the MLD algorithm can be simplified. First, since the underlying code is always linear, the error probability of MLD is independent of the actual transmitted codeword, thus we can always, without loss of generality, transmit the all-zero codeword. Furthermore, when the all-zero codeword is transmitted and a codeword $\mathbf{c}$ with objective value $\psi_{\boldsymbol{\lambda}}(\mathbf{c}) < 0 = \psi_{\boldsymbol{\lambda}}(\mathbf{0})$ has been identified, the search can be terminated, since any ML decoder would also fail on this received LLR vector $\boldsymbol{\lambda}$.

11.5 Minimum Distance Computation

The MLD problem is closely related to the computation of the minimum distance $d_{\min}$ of a code as follows. If the all-zero codeword is explicitly forbidden, then an MLD algorithm with the input $\boldsymbol{\lambda} = \mathbf{1}$ will output a codeword of minimum weight:

$$d_{\min}(\mathscr{C}) = \min_{\mathbf{c}\in\mathscr{C}\setminus\{\mathbf{0}\}} \psi_{\mathbf{1}}(\mathbf{c}) = \min_{\mathbf{c}\in\operatorname{conv}(\mathscr{C}\setminus\{\mathbf{0}\})} \psi_{\mathbf{1}}(\mathbf{c}). \tag{11.4}$$

Our proposed decoding algorithm can be modified to exclude the all-zero codeword by the following changes:

(1) Extend the condition in Step 10 of Algorithm 11.1 to "$\hat{\mathbf{p}}$ is integral and $\hat{\mathbf{p}} \neq \mathbf{0}$", which avoids decoding to the all-zero codeword.

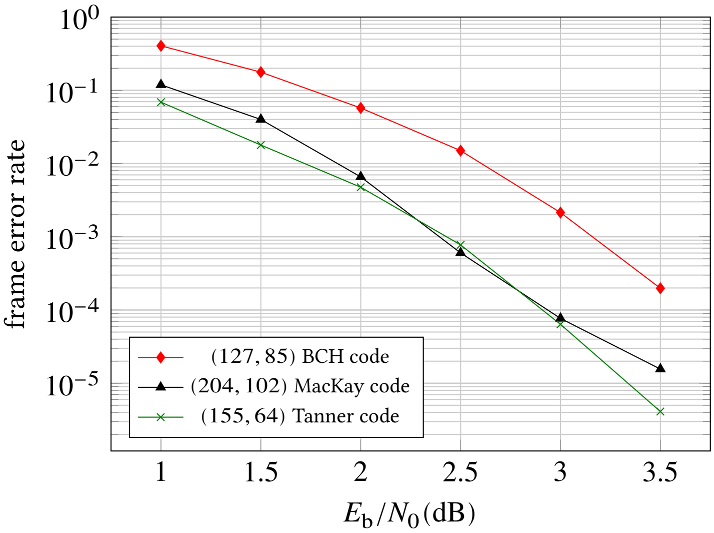


Figure 11.2: MLD performance of the codes considered in this paper.

(2) In the order-i re-encoding performed in Algorithm 11.2, exclude $\mathbf{0}$ from the set of candidate codewords.

Moreover, note that all feasible solutions (i. e., codewords) of (11.4) have an integral objective value. This allows us to change the right hand side in Steps 2, 4, and 14 to $\tau - 1 + \varepsilon$, for a small $\varepsilon > 0$, since $\left\lceil \bar{\psi}^{(F)}_{\min,1} \right\rceil = \tau$ implies that $\psi^{(F)}_{\min,1} \geq \tau$.

11.6 Numerical Results

In this section, we present some numerical results for our proposed MLD algorithm, with all the improvements outlined above in Section 11.4, for several codes on the AWGN channel. We have used order-2 re-encoding ($i = 2$ in Algorithm 11.2), and the open-source GLPK library [16] to solve the LP problems. The following set of parameters was heuristically found to perform well for all codes: $M = 30$, $\delta = 2$, $T = 100$, $R = 5$, $R^{\text{bb}} = 100$, and $\gamma = 0.2$.

As a benchmark, we use the CPLEX IP solver [3], with the IP formulation named IPD1 in [2] which was found to be most efficient in that paper. In case of all-zero decoding, we configured CPLEX to terminate as soon as a codeword with objective value below zero was found, mimicking the adaptions of our algorithm described in Section 11.4.3.

We compare the algorithms with respect to both (single-core) average CPU time T_{avg} and average number N_{avg} of branch-and-bound nodes processed per frame. For our algorithm, N_{avg} equals the number of times the main loop (Step 2 of Algorithm 11.1) is processed, while for CPLEX we report the attribute “number of processed nodes”. Note that the latter drops below one for high SNR, which is probably due to CPLEX’ presolve strategy that establishes optimality in some cases without ever starting the branch-and-bound procedure.

	$d_{\min}$	T^{MLD}	T^{CPLEX}	N^{MLD}	N^{CPLEX}
$(155, 64)$ Tanner code	20	137 s	3682 s	42 785	21842224
$(204, 102)$ MacKay code	8	1.6 s	11.49 s	371	44830
$(408, 204)$ MacKay code	14	152 s	6893 s	9345	936 570

Table 11.3: Numerical results for minimum distance computation.

All calculations were performed on a desktop PC with an Intel Core i5-3470 CPU (3.2 GHz) and 8 GB of RAM.

11.6.1 Maximum-Likelihood Decoding

A comparison of CPLEX and our MLD algorithm, for the different codes outlined below, is given in Table 11.1, both in terms of T_{avg} and N_{avg}. The numbers in the parentheses are for CPLEX; a dash indicates that CPLEX was not able to decode a sufficient number of frames without running out of memory. The corresponding MLD performance curves are plotted in Figure 11.2. For the curves, we have counted 100 erroneous frames for each simulation point.

The $(155, 64)$ Tanner code from [9] is often used as a benchmark code, and was also considered in [8]. For all SNRs, the ZS decoding algorithm showed a performance loss compared to the MLD curve [8].

As can be seen from Table 11.1, our algorithm is more than 11 times as fast as CPLEX for an SNR of 1.0 dB. For higher values of the SNR our proposed algorithm is even faster compared to CPLEX. In the case of all-zero decoding, both algorithms are faster by a factor of 2 to 3, while the relative performance gain by our algorithm remains roughly the same.

The second example is a $(3, 6)$-regular $(204, 102)$ LDPC code taken from the online database of sparse graph codes from MacKay's website [15] (called 204.33.484 there). As can be seen from Table 11.1, also for this code, our algorithm is significantly faster than CPLEX for all simulated SNRs.

In order to evaluate the performance of our algorithm for dense codes, Table 11.1 includes results for the $(127, 85)$ BCH code. CPLEX was not able to decode a significant number of frames for this code, and to our knowledge the MLD curve, as presented in Figure 11.2, was previously unknown.

11.6.2 Minimum Distance Computation

In the case of computing the minimum distance, we used different values for some of the parameters, namely $M = 120$, $R = 1$, $R^{\mathrm{bb}} = 1$, and $\gamma = 0.3$. Results are shown in Table 11.3, which additionally contains the $(408, 204)$ MacKay code (named 408.33.844 at the website [15]) that was used also in [8]. Note that in case of the $(155, 64)$ Tanner code, we can exploit the symmetry and fix $c_0 = 1$ before starting the algorithm. We compare our algorithm to CPLEX with the same formulation as for MLD and the additional constraint $\sum_{i=0}^{n-1} c_i \geq 1$ to exclude the all-zero codeword.[1]

References

[1] E. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. "On the inherent intractability of certain coding problems". *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 954–972. DOI: 10.1109/TIT.1978.1055873.

[2] A. Tanatmis et al. "Numerical comparison of IP formulations as ML decoders". In: *IEEE International Conference on Communications*. Cape Town, South Africa, May 2010, pp. 1–5. DOI: 10.1109/ICC.2010.5502303.

[3] *IBM ILOG CPLEX Optimization Studio*. Software Package. Version 12.6. 2013.

[4] E. Rosnes et al. "Addendum to 'An Efficient Algorithm to Find All Small-Size Stopping Sets of Low-Density Parity-Check Matrices'". *IEEE Transactions on Information Theory* 58.1 (Jan. 2012), pp. 164–171. ISSN: 0018-9448. DOI: 10.1109/TIT.2011.2171531.

[5] E. Rosnes and Ø. Ytrehus. "An efficient algorithm to find all small-size stopping sets of low-density parity-check matrices". *IEEE Transactions on Information Theory* 55.9 (Sept. 2009), pp. 4167–4178. ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2025573.

[6] J. Feldman, M. J. Wainwright, and D. R. Karger. "Using linear programming to decode binary linear codes". *IEEE Transactions on Information Theory* 51.3 (Mar. 2005), pp. 954–972. DOI: 10.1109/TIT.2004.842696. URL: www.eecs.berkeley.edu/~wainwrig/Papers/FelWaiKar05.pdf.

[7] M. H. Taghavi and P. H. Siegel. "Adaptive methods for linear programming decoding". *IEEE Transactions on Information Theory* 54.12 (Dec. 2008), pp. 5396–5410. DOI: 10.1109/TIT.2008.2006384. arXiv: cs/0703123 `[cs.IT]`.

[8] X. Zhang and P. H. Siegel. "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes". *IEEE Transactions on Information Theory* 58.10 (Oct. 2012), pp. 6581–6594. DOI: 10.1109/TIT.2012.2204955. arXiv: 1105.0703 `[cs.IT]`.

[9] R. M. Tanner, D. Sridhara, and T. Fuja. "A class of group-structured LDPC codes". In: *International Symposium on Communication Theory and Applications (ISCTA)*. Ambleside, England, July 2001.

[10] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. "Factor graphs and the sum-product algorithm". *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 498–519. DOI: 10.1109/18.910572. URL: www.comm.utoronto.ca/frank/papers/KFL01.pdf.

---

[1] As a remark, for the (408, 204) MacKay code we have used the previous CPLEX 12.5 instead of 12.6; apparently there is a bug in the latter, causing it to output a $d_{\min}$ of 20 instead of the correct value 14.

[11] M. P. C. Fossorier and S. Lin. “Soft-decision decoding of linear block codes based on ordered statistics”. *IEEE Transactions on Information Theory* 41.5 (Sept. 1995), pp. 1379–1396. DOI: 10.1109/18.412683.

[12] M. Helmling, S. Ruzika, and A. Tanatmis. “Mathematical programming decoding of binary linear codes: theory and algorithms”. *IEEE Transactions on Information Theory* 58.7 (July 2012), pp. 4753–4769. DOI: 10.1109/TIT.2012.2191697. arXiv: 1107.3715 `[cs.IT]`.

[13] P. O. Vontobel and R. Koetter. *Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes.* 2005. arXiv: cs/0512078 `[cs.IT]`.

[14] U. Faigle, W. Kern, and G. Still. *Algorithmic Principles of Mathematical Programming.* Vol. 24. Kluwer Academic Publishers, 2010.

[15] D. J. C. MacKay. *Encyclopedia of sparse graph codes.* 2014. URL: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[16] The GNU Project. *GNU Linear Programming Kit (GLPK).* Software Library. Version 4.52. URL: http://www.gnu.org/software/glpk.

Chapter 12

Paper VII: Minimum Pseudoweight Analysis of 3-Dimensional Turbo Codes

Eirik Rosnes, Michael Helmling, and Alexandre Graell i Amat

The following chapter is a reformatted copy of a preprint that is publicly available online (http://arxiv.org/abs/1103.1559). A paper with the same content appeared in the following publication:

> E. Rosnes, M. Helmling, and A. Graell i Amat. "Minimum pseudoweight analysis of 3-dimensional turbo codes". *IEEE Transactions on Communications* 62.7 (July 2014), pp. 2170–2182. DOI: 10.1109/TCOMM.2014.2329690. arXiv: 1103.1559 `[cs.IT]`

A subset of the results was previously presented at the 2011 ISIT conference and appeared in the conference proceedings:

> E. Rosnes, M. Helmling, and A. Graell i Amat. "Pseudocodewords of linear programming decoding of 3-dimensional turbo codes". In: *Proceedings of IEEE International Symposium on Information Theory*. St. Petersburg, Russia, July 2011, pp. 1643–1647. DOI: 10.1109/ISIT.2011.6033823

The work of E. Rosnes and M. Helmling were supported in part by the DFG under Grant RU-1524/2-1 and in part by the DAAD/RCN under Grant 54565400 within the German–Norwegian Collaborative Research Support Scheme. The work of A. Graell i Amat was supported by the Swedish Research Council under Grant 2011-596.

Minimum Pseudoweight Analysis of 3-Dimensional Turbo Codes

Eirik Rosnes Michael Helmling
Alexandre Graell i Amat

In this work, we consider pseudocodewords of (relaxed) linear programming (LP) decoding of 3-dimensional turbo codes (3D-TCs). We present a relaxed LP decoder for 3D-TCs, adapting the relaxed LP decoder for conventional turbo codes proposed by Feldman in his thesis. We show that the 3D-TC polytope is *proper* and *C-symmetric*, and make a connection to finite graph covers of the 3D-TC factor graph. This connection is used to show that the support set of any pseudocodeword is a stopping set of iterative decoding of 3D-TCs using maximum *a posteriori* constituent decoders on the binary erasure channel. Furthermore, we compute ensemble-average pseudoweight enumerators of 3D-TCs and perform a finite-length minimum pseudoweight analysis for small cover degrees. Also, an explicit description of the fundamental cone of the 3D-TC polytope is given. Finally, we present an extensive numerical study of small-to-medium block length 3D-TCs, which shows that 1) typically (i.e., in most cases) when the minimum distance d_{min} and/or the stopping distance h_{min} is high, the minimum pseudoweight (on the additive white Gaussian noise channel) is strictly smaller than both the d_{min} and the h_{min}, and 2) the minimum pseudoweight grows with the block length, at least for small-to-medium block lengths.

12.1 Introduction

Turbo codes (TCs) have gained considerable attention since their introduction by Berrou *et al.* in 1993 [1] due to their near-capacity performance and low decoding complexity. The conventional TC is a parallel concatenation of two identical recursive systematic convolutional encoders separated by a pseudorandom interleaver. To improve the performance of TCs in the error floor region, hybrid concatenated codes (HCCs) can be used. In [2], a powerful HCC nicknamed 3-dimensional turbo code (3D-TC) was introduced. The coding scheme consists of a conventional turbo encoder and a *patch*, where a fraction λ of the parity bits from the turbo encoder are post-encoded by a third rate-1 encoder. The value of λ can be used to trade off performance in the waterfall region with performance in the error floor region. As shown in [2], this coding scheme is able to provide very low error rates for a wide range of block lengths and code rates at the expense of a small increase in decoding complexity with respect to conventional TCs. In a recent work [3], an in-depth performance analysis of 3D-TCs was conducted. Stopping sets for 3D-TCs were treated in [4]. Finally, we remark that tuned TCs [5] are another family of HCC ensembles where performance in the waterfall and error floor regions can be traded off using a tuning parameter.

The use of linear programming (LP) to decode turbo-like codes was introduced by Feldman *et al.* [6, 7]. They proposed an LP formulation that resembles a shortest path problem, based on the trellis graph representation of the constituent convolutional codes. The natural polytope for LP decoding would be the convex hull of all codewords, in which case LP decoding is equivalent to maximum-likelihood (ML) decoding. However, an efficient description of that polytope is not known in general, i.e., its description length most likely grows exponentially with the block length. The formulation proposed by Feldman *et al.* [6, 7], which grows only linearly with the block length, is a relaxation in the sense that it describes a superset of the ML decoding polytope, introducing additional, fractional vertices. The vertices of the relaxed polytope (both integral and fractional) are what the authors called *pseudocodewords* in [8].

The same authors also introduced a different LP formulation to decode low-density parity-check (LDPC) codes [7, 8] that has been extensively studied since then by various researchers. For that LP decoder, Vontobel and Koetter [9] showed that the set of the polytope's points with rational coordinates (which in particular includes all of its vertices) is equal to the set of all pseudocodewords derived from all finite graph covers of the Tanner graph. In [10], a similar result was established (but with no proof included) for the case of conventional TCs. Here, we prove that statement for the case of 3D-TCs.

In this work, we study (relaxed) LP decoding of 3D-TCs, explore the connection to finite graph covers of the 3D-TC factor graph [11], and adapt the concept of a pseudocodeword. Furthermore, we compute ensemble-average pseudoweight enumerators and perform a finite-length minimum additive white Gaussian noise (AWGN) pseudoweight analysis which shows that the minimum AWGN pseudoweight grows with the block length, at least for small-to-medium block lengths. Furthermore, we show by several examples and by computer search that typically (i.e., in most cases) when the minimum distance $d_{\min}$ and/or the stopping distance $h_{\min}$ is high, the minimum AWGN pseudoweight, denoted by $w_{\min}^{\mathrm{AWGN}}$, is strictly smaller than both the $d_{\min}$ and the $h_{\min}$ for these codes. In [12], Chertkov and Stepanov presented an updated, more efficient (compared to the algorithm from [13]) minimum pseudoweight search algorithm based entirely on the concept of the *fundamental cone* [9] of the LDPC code LP decoder. We will show that such a fundamental cone can be described also for the 3D-TC LP decoder.

Some other work related to the content of this paper is the work on pseudocodeword analysis of binary and nonbinary (protograph-based) LDPC and generalized LDPC codes. See, for instance, [14–16], and references therein. For such codes, the component codes are the trivial repetition code and single parity-check codes, or in the case of generalized LDPC codes, more advanced classical linear block codes. However, not much work has been done on pseudocodeword analysis for turbo-like codes with trellis-based constituent codes. In contrast to these previous works, the trellis-based approach in this paper is different and provides a pseudocodeword analysis of 3D-TCs that can be adapted also to other trellis-based turbo-like codes or concatenated codes based on two or more convolutional codes. We remark that some results on enumerating pseudocodewords for convolutional codes have already been provided by Boston and Conti in [17, 18]. Finally, it is worth mentioning [19] which presents, among several results, a combinatorial characterization of the *Bethe entropy function* in terms of finite graph covers of the factor graph under consideration. In particular, a characterization in terms of the average number of preimages of a *pseudomarginal* vector of rational entries. In contrast to the general framework in [19], this paper discusses techniques to numerically deal with large-degree function nodes representing the indicator function of (long) convolutional codes.

The remainder of the paper is organized as follows. In Section 12.2, we describe the system model and introduce some notation. In Section 12.3, we describe (relaxed) LP decoding of 3D-TCs. The connection to finite graph covers of the 3D-TC factor graph is explored in Section 12.4. Ensemble-average pseudoweight enumerators of 3D-TCs are computed in Section 12.5. These enumerators are subsequently used to perform a probabilistic finite-length minimum AWGN pseudoweight analysis of 3D-TCs. Section 12.6, an efficient heuristic for searching for low AWGN pseudoweight pseudocodewords is discussed. Finally, in Section 12.7, an extensive numerical study is presented, and some conclusions are drawn in Section 12.8.

12.2 Coding Scheme

A block diagram of the 3D-TC is depicted in Figure 12.1. The information data sequence $\mathbf{u}$ of length K bits is encoded by a binary conventional turbo encoder. By a conventional turbo encoder we mean the parallel concatenation of two identical rate-1 recursive convolutional encoders, denoted by C_{a} and C_{b}, respectively. Here, C_{a} and C_{b} are 8-state recursive convolutional encoders with generator polynomial $g(D) = (1 + D + D^3)/(1 + D^2 + D^3)$, i.e., the 8-state constituent encoder specified in the 3GPP LTE standard [20]. The code sequences of C_{a} and C_{b} are denoted by $\mathbf{x}_{\mathrm{a}}$ and $\mathbf{x}_{\mathrm{b}}$, respectively. We also denote by $\mathbf{x}^{\mathrm{TC}}$ the codeword obtained by alternating bits from $\mathbf{x}_{\mathrm{a}}$ and $\mathbf{x}_{\mathrm{b}}$. A fraction λ $(0 \le \lambda \le 1)$, called the *permeability rate*, of the parity bits from $\mathbf{x}^{\mathrm{TC}}$ are permuted by the interleaver Π_{c} (of length $N_{\mathrm{c}} = 2\lambda K$), and encoded by

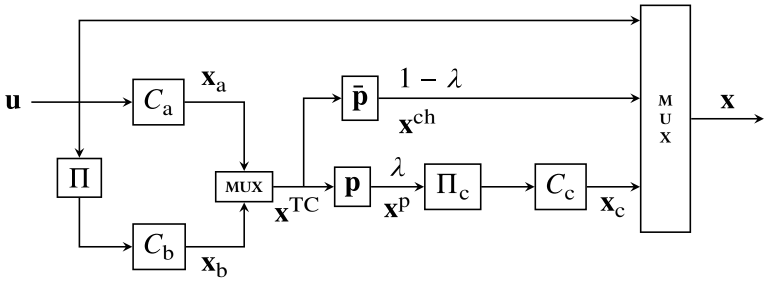


Figure 12.1: 3D turbo encoder. A fraction λ of the parity bits from both constituent encoders C_a and C_b are grouped by a parallel/serial multiplexer, permuted by the interleaver Π_c, and encoded by the rate-1 post-encoder C_c.

an encoder of unity rate C_c with generator polynomial $g(D) = 1/(1 + D^2)$, called the *patch* or the *post-encoder* [2]. This can be properly represented by a puncturing pattern $\mathbf{p}$ applied to $\mathbf{x}^{TC}$ (see Figure 12.1) of period N_p containing λN_p ones (where a one means that the bit is not punctured). Note that the encoder of the patch is like two accumulators, one operating on the even bits and one operating on the odd bits. The fraction $1 - \lambda$ of parity bits which are not encoded by C_c is sent directly to the channel. Equivalently, this can be represented by a puncturing pattern $\bar{\mathbf{p}}$, the complement of $\mathbf{p}$. We denote by $\mathbf{x}_c$ the code sequence produced by C_c. Also, we denote by $\mathbf{x}_a^{ch}$ and $\mathbf{x}_b^{ch}$ the *sub-codewords* of $\mathbf{x}_a$ and $\mathbf{x}_b$, respectively, sent directly to the channel, and by $\mathbf{x}^{ch}$ the codeword obtained by multiplexing (in some order) the bits from $\mathbf{x}_a^{ch}$ and $\mathbf{x}_b^{ch}$. Likewise, we denote by $\mathbf{x}_a^{p}$ and $\mathbf{x}_b^{p}$ the *sub-codewords* of $\mathbf{x}_a$ and $\mathbf{x}_b$, respectively, encoded by C_c, and by $\mathbf{x}^{p}$ the codeword obtained by multiplexing (in some order) the bits from $\mathbf{x}_a^{p}$ and $\mathbf{x}_b^{p}$. Finally, the information sequence and the code sequences $\mathbf{x}^{ch}$ and $\mathbf{x}_c$ are multiplexed to form the code sequence $\mathbf{x}$, of length N bits, transmitted to the channel. Note that the overall nominal code rate of the 3D-TC is $R = K/N = 1/3$, the same as for the conventional TC without the patch. Higher code rates can be obtained either by puncturing $\mathbf{x}^{ch}$ or by puncturing the output of the patch, $\mathbf{x}_c$.

In [2], regular puncturing patterns of period $2/\lambda$ were considered for $\mathbf{p}$. For instance, if $\lambda = 1/4$, every fourth bit from each of the encoders of the outer TC are encoded by encoder C_c. The remaining bits are sent directly to the channel, and it follows that $\mathbf{p} = [11000000]$ and $\bar{\mathbf{p}} = [00111111]$. Note that with this particular puncturing pattern and even with the generator polynomial $g(D) = 1/(1 + D^2)$ for C_c (which is like two separate accumulators operating on even and odd bits, respectively), the bit streams $\mathbf{x}_a$ and $\mathbf{x}_b$ are in general intermingled because of the interleaver Π_c.

12.3 LP Decoding of 3D-TCs

In this section, we consider relaxed LP decoding of 3D-TCs, adapting the relaxation proposed in [7] for conventional TCs to 3D-TCs.

Let $T_\mathrm{x} = T_\mathrm{x}(V_\mathrm{x}, E_\mathrm{x})$ denote the *information bit-oriented trellis* of C_x, $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$, where the vertex set V_x partitions as $V_\mathrm{x} = \bigcup_{t=0}^{I_\mathrm{x}} V_{\mathrm{x},t}$, which also induces the partition $E_\mathrm{x} = \bigcup_{t=0}^{I_\mathrm{x}-1} E_{\mathrm{x},t}$ of the edge set E_x, where I_x is the trellis length of T_x. In the following, the encoders C_x (and their corresponding trellises T_x) are assumed (with some abuse of notation) to be systematic, in the sense that the output bits are prefixed with the input bits. Thus, C_x is regarded as a rate-$1/2$ encoder, and the trellis T_x has an output label containing two bits, for $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$. Now, let $e \in E_{\mathrm{x},t}$ be an arbitrary edge from the tth trellis section. The ith bit in the output label of e is denoted by $c_i(e)$, $i = 0, \dots, n_{\mathrm{x},t} - 1$, the starting state of e as $s^S(e)$, and the ending state of e as $s^E(e)$, where $n_{\mathrm{x},t}$ is the number of bits in the output label of an edge $e \in E_{\mathrm{x},t}$.

For $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$, we define the path polytope $\mathcal{Q}_\mathrm{x}$ to be the set of all $\mathbf{f}^\mathrm{x} \in [0,1]^{|E_\mathrm{x}|}$ satisfying

$$\sum_{e \in E_{\mathrm{x},0}} f_e^\mathrm{x} = 1 \tag{12.1a}$$

$$\sum_{\substack{e \in E_\mathrm{x}:\\ s^S(e)=v}} f_e^\mathrm{x} = \sum_{\substack{e \in E_\mathrm{x}:\\ s^E(e)=v}} f_e^\mathrm{x} \quad \text{for all } v \in V_{\mathrm{x},t} \text{ and } t = 1, \dots, I_\mathrm{x} - 1 \tag{12.1b}$$

and let $\mathcal{Q} = \mathcal{Q}_\mathrm{a} \times \mathcal{Q}_\mathrm{b} \times \mathcal{Q}_\mathrm{c}$. Note that $\mathcal{Q}$ is the set of all feasible network flows through the three trellis graphs T_x, $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$.

Next, we define the polytope $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$ as the pairs $(\tilde{\mathbf{y}}, \mathbf{f})$, where $\tilde{\mathbf{y}} \in [0,1]^{N+2\lambda K}$ and $\mathbf{f} = (\mathbf{f}^\mathrm{a}, \mathbf{f}^\mathrm{b}, \mathbf{f}^\mathrm{c}) \in [0,1]^{|E_\mathrm{a} \cup E_\mathrm{b} \cup E_\mathrm{c}|}$, meeting the constraints

$$(\mathbf{f}^\mathrm{a}, \mathbf{f}^\mathrm{b}, \mathbf{f}^\mathrm{c}) \in \mathcal{Q} \tag{12.2a}$$

$$\sum_{\substack{e \in E_{\mathrm{x},t}:\\ c_i(e)=1}} f_e^\mathrm{x} = \tilde{y}_{\rho_\mathrm{x}(\phi_\mathrm{x}(t,i))} \quad \begin{array}{l} \text{for } t = 0, \dots, I_\mathrm{x} - 1, \\ i = 0, \dots, n_{\mathrm{x},t} - 1, \text{ and } \mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}. \end{array} \tag{12.2b}$$

Here, $\phi_\mathrm{x}(t,i) = \sum_{j=0}^{t-1} n_{\mathrm{x},j} + i$, and $\rho_\mathrm{x}(\cdot)$ denotes the mapping of codeword indices of the constituent encoder C_x to codeword indices of the overall codeword of the 3D-TC appended with the $2\lambda K$ parity bits from encoders C_a and C_b which are sent to the patch.

Finally, let

$$\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}} = \left\{ \mathbf{y} \in [0,1]^N : \exists \hat{\mathbf{y}} \in [0,1]^{2\lambda K}, \mathbf{f} \in \mathcal{Q} \text{ with } ((\mathbf{y}, \hat{\mathbf{y}}), \mathbf{f}) \in \mathcal{Q}_{\Pi,\Pi_\mathrm{c}} \right\}$$

be the projection of $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$ onto the first N variables.

Relaxed LP decoding (on a binary-input memoryless channel) of 3D-TCs can be described by the linear program

$$\text{minimize} \sum_{l=0}^{N-1} \lambda_l y_l \text{ subject to } \mathbf{y} \in \dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}} \tag{12.3}$$

where

$$\lambda_l = \log\left(\frac{\Pr\{r_l \mid c_l = 0\}}{\Pr\{r_l \mid c_l = 1\}} \right), \quad l = 0, \dots, N-1,$$

c_l is the lth codeword bit, and r_l is the lth component of the received vector. If instead of $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ we use the convex hull of the codewords of the 3D-TC, then solving the linear program in (12.3) is equivalent to ML decoding.

The notion of a *proper* and C-*symmetric* polytope was introduced in [7, Ch. 4] where the author proved that the probability of error of LP decoding is independent of the transmitted codeword on a binary-input output-symmetric memoryless channel when the underlying code is linear and the polytope is proper and C-symmetric.

12.1 Proposition: *Let C denote a given 3D-TC with interleavers Π and Π_{c}. The polytope $\dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}}$ is proper, i.e., $\dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}} \cap \{0,1\}^N = C$ and C-symmetric, i.e., for any $\mathbf{y} \in \dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}}$ and $\mathbf{c} \in C$ it holds that $|\mathbf{y} - \mathbf{c}| \in \dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}}$.* ◁

Feldman proved a similar statement in the context of LDPC codes (Lemma 5.2 and Theorem 5.4 in [7]). However, note that his proof is based explicitly on the inequalities of the LP formulation for LDPC codes, and therefore does not generalize to the polytope $\dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}}$.

In Appendix 12.A, we give a formal proof of both statements in a much more general setting.

12.4 Finite Graph Covers

Let C denote a given 3D-TC with interleavers Π and Π_{c}, and constituent codes C_{x}, $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$. The factor graph [11] of C_{x}, denoted by $\Gamma(C_{\mathrm{x}})$, is composed of *state*, *input*, *parity*, and *check* vertices. The state vertices $s_{\mathrm{x},0}, \dots, s_{\mathrm{x},I_{\mathrm{x}}}$ in $\Gamma(C_{\mathrm{x}})$ represent state spaces of the length-I_{x} information bit-oriented trellis T_{x} of C_{x}. The lth check vertex represents the lth trellis section, i.e., it is an indicator function for the set of allowed combinations of *left* state, input symbol, parity symbol, and *right* state. A factor graph $\Gamma(C)$ of C is constructed as follows.

(1) Remove all the input vertices of $\Gamma(C_{\mathrm{b}})$ by connecting the lth input vertex of $\Gamma(C_{\mathrm{a}})$ to the $\Pi(l)$th check vertex of $\Gamma(C_{\mathrm{b}})$.

(2) Remove all the input vertices of $\Gamma(C_{\mathrm{c}})$ by connecting the parity vertex (from either $\Gamma(C_{\mathrm{a}})$ or $\Gamma(C_{\mathrm{b}})$) corresponding to the lth bit in $\mathbf{x}^{\mathrm{p}}$ to the $\Pi_{\mathrm{c}}(l)$th check vertex of $\Gamma(C_{\mathrm{c}})$.

To construct a degree-m cover of $\Gamma(C)$, denoted by $\Gamma^{(m)}(C)$, we first make m identical copies of $\Gamma(C)$. Now, any permutation of the edges, denoted by $E = E(\Gamma^{(m)}(C))$, connecting the copies of the constituent factor graphs $\Gamma(C_{\mathrm{x}})$ such that the following conditions are satisfied, will give a valid cover of $\Gamma(C)$.

(1) The m copies of the lth input vertex of $\Gamma(C_{\mathrm{a}})$ should be connected by a one-to-one mapping (or permutation) to the m copies of the $\Pi(l)$th check vertex of $\Gamma(C_{\mathrm{b}})$.

(2) The m copies of the parity vertex (from either $\Gamma(C_{\mathrm{a}})$ or $\Gamma(C_{\mathrm{b}})$) corresponding to the lth bit in $\mathbf{x}^{\mathrm{p}}$ should be connected by a permutation to the m copies of the $\Pi_{\mathrm{c}}(l)$th check vertex of $\Gamma(C_{\mathrm{c}})$.

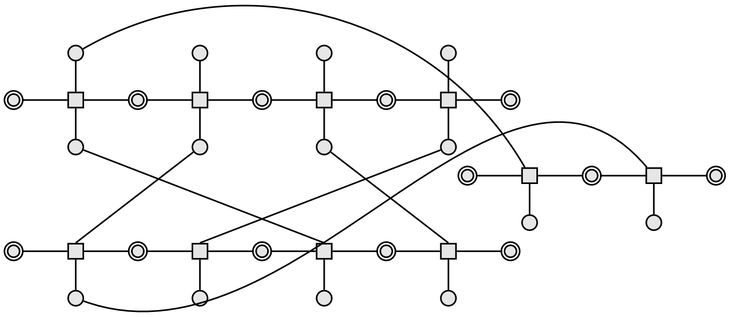


Figure 12.2: Factor graph of a nominal rate-1/3 3D-TC with $\lambda = 1/4$, using the regular puncturing pattern $\mathbf{p} = [11000000]$.

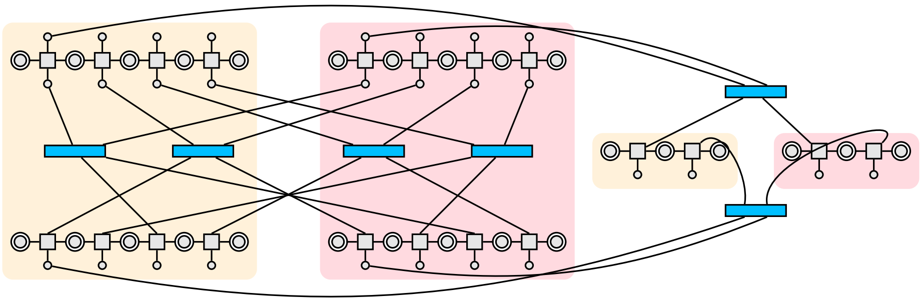


Figure 12.3: Degree-2 cover of the factor graph in Figure 12.2. The six rectangular blue boxes are permutations of size 2 that can be chosen arbitrarily. The nodes highlighted in yellow and pink belong to the first and second cover copy, respectively.

The corresponding code is denoted by $C^{(m)}$. Let $\mathbf{x}^{(m)} = (x_0^{(0)}, \dots, x_{N-1}^{(0)}, \dots, x_0^{(m-1)}, \dots, x_{N-1}^{(m-1)})$ denote a codeword in $C^{(m)}$, define

$$\omega_l(\mathbf{x}^{(m)}) = \frac{|\{i\colon x_l^{(i)} = 1\}|}{m}$$

and let $\boldsymbol{\omega} = \boldsymbol{\omega}(\mathbf{x}^{(m)}) = (\omega_0(\mathbf{x}^{(m)}), \dots, \omega_{N-1}(\mathbf{x}^{(m)}))$. Now, $\boldsymbol{\omega}$ as defined above is said to be a *graph-cover pseudocodeword* of degree m.

12.2 Example: Figure 12.2 depicts the factor graph of a nominal rate-1/3 3D-TC with $\lambda = 1/4$ and input length $K = 4$, using the regular puncturing pattern $\mathbf{p} = [11000000]$. The upper part to the left is the factor graph of C_a, the lower part to the left is the factor graph of C_b, and the right part is the factor graph of C_c. The brown squares in the graph are check vertices corresponding to trellis sections. The single circles are input and parity vertices, and the double circles are state vertices. Figure 12.3 depicts a degree-2 cover of the factor graph from Figure 12.2. The six small blue rectangular boxes are permutations of size 2 that can be chosen arbitrarily. ◁

12.3 Proposition: *The following statements are true:*

(1) The points in $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}} \cap \mathbb{Q}^N$ are in one-to-one correspondence with $\mathcal{P}_{\Pi,\Pi_\mathrm{c}}$, where $\mathbb{Q}$ is the set of rational numbers and $\mathcal{P}_{\Pi,\Pi_\mathrm{c}}$ is the set of all graph-cover pseudocodewords from all finite graph covers of the 3D-TC factor graph.

(2) All vertices of $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ have rational entries. ◁

A similar result was proved in [9] for LDPC codes. We give a formal proof for the case of 3D-TCs in Appendix 12.B. It is inspired by the one in [9], but is a bit more involved because pseudocodewords are defined only indirectly by a linear image of the polytope $\mathcal{Q}$. Note that our proof does not depend on the detailed set-up of 3D-TCs, so it can be extended to all sorts of turbo-like coding schemes.

When decoding 3D-TCs by solving the linear program in (12.3), there is always a vertex of $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ at which the optimum value is attained. We can therefore assume that the LP decoder always returns a vertex and hence, by Proposition 12.3, a (graph-cover) pseudocodeword. Furthermore, the pseudoweight on the AWGN channel of a nonzero pseudocodeword $\boldsymbol{\omega}$ is defined as [9, 21]

$$w^{\mathrm{AWGN}}(\boldsymbol{\omega}) = \frac{\|\boldsymbol{\omega}\|_1^2}{\|\boldsymbol{\omega}\|_2^2} = \frac{\left(\sum_{l=0}^{N-1} \omega_l\right)^2}{\sum_{l=0}^{N-1} \omega_l^2} \tag{12.4}$$

where $\|\cdot\|_q$ is the ℓ_q-norm of a vector.

12.4 Proposition: *Let C denote a given 3D-TC with interleavers Π and Π_c. For any pseudocodeword $\boldsymbol{\omega}$, the support set $\chi(\boldsymbol{\omega})$ of $\boldsymbol{\omega}$, i.e., the index set of the nonzero coordinates, is a stopping set according to [4, Def. 1]. Conversely, for any stopping set $\mathcal{S} = \mathcal{S}(\Pi, \Pi_\mathrm{c})$ of the 3D-TC there exists a pseudocodeword $\boldsymbol{\omega}$ with support set $\chi(\boldsymbol{\omega}) = \mathcal{S}$.* ◁

Proof. This result can be proved in the same manner as the corresponding result for conventional TCs [10, Lem. 2]. The proof given in [10] is based on the linearity of the subcodes $\bar{C}_{\mathrm{a}}$ and $\bar{C}_{\mathrm{b}}$ (from the stopping set definition in [10, Def. 1]). For 3D-TCs the same proof applies using the linearity of all the three subcodes $\hat{C}_{\mathrm{a}}$, $\hat{C}_{\mathrm{b}}$, and $\hat{C}_{\mathrm{c}}$ from [4, Def. 1]. □

As a consequence of Proposition 12.4, it follows that $w_{\min}^{\mathrm{AWGN}}$ of C is upper-bounded by the $h_{\min}$ of C.

We remark that the $d_{\min}$ can be computed exactly by solving the *integer* program in (12.3) with $\lambda_l = 1$ for all l, with integer constraints on all the flow variables $\mathbf{f}$ in (12.1) and (12.2), i. e., $f_l \in \{0,1\}$ for all l, and with the constraint $\sum_{l=0}^{N-1} y_l \geq 1$ to avoid obtaining the all-zero codeword (see [22, Prop. 3.6]). The exact $d_{\min}$ of 3D-TCs has not been computed before in the literature, but will be computed later in this paper for several codes. For instance, in [3], only estimates of $d_{\min}$ were provided. Finally, note that the exact $h_{\min}$ can be computed in a similar manner using *extended trellis modules* in T_{x} (see [23] for details).

12.5 Ensemble-Average Pseudoweight Enumerators

In this section, we describe how to compute the *ensemble-average pseudoweight enumerator* of 3D-TCs for a given graph cover degree m.

Here, we first introduce the concept of a *pseudocodeword vector-weight enumerator (PCVWE)* $\mathcal{P}_{\mathbf{w},\mathbf{h}}^{C_{\mathrm{x}}}$ of the constituent code C_{x}. In particular, $\mathcal{P}_{\mathbf{w},\mathbf{h}}^{C_{\mathrm{x}}}$ is the number of *pseudocodewords* in the constituent code C_{x}, $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$, of Hamming *vector-weight* $\mathbf{h} = (h_1, \dots, h_m)$ corresponding to input sequences of Hamming *vector-weight* $\mathbf{w} = (w_1, \dots, w_m)$. The pseudocodewords of a constituent code C_{x} are obtained as follows. Let $C_{\mathrm{x}}^{(m)}$ denote the degree-m cover of constituent code C_{x}, which is obtained by concatenating C_{x} by itself m times, i.e.,

$$C_{\mathrm{x}}^{(m)} = \left\{ (\mathbf{x}_0, \dots, \mathbf{x}_{m-1}) \colon \mathbf{x}_i \in C_{\mathrm{x}}, \forall i \in \{0, \dots, m-1\} \right\}.$$

Now, let

$$\mathbf{x}^{(m)} = \left(x_0^{(0)}, \dots, x_{N_{\mathrm{x}}-1}^{(0)}, \dots, x_0^{(m-1)}, \dots, x_{N_{\mathrm{x}}-1}^{(m-1)} \right)$$

denote a codeword in $C_{\mathrm{x}}^{(m)}$, where $(x_0^{(i)}, \dots, x_{N_{\mathrm{x}}-1}^{(i)})$ is a codeword in C_{x} for all i, $0 \leq i \leq m-1$, and N_{x} is the block length of C_{x}. The corresponding *unnormalized* pseudocodeword is

$$\boldsymbol{\omega}(\mathbf{x}^{(m)}) = \left(\sum_{i=0}^{m-1} x_0^{(i)}, \dots, \sum_{i=0}^{m-1} x_{N_{\mathrm{x}}-1}^{(i)} \right) \tag{12.5}$$

where addition is integer addition, which means that each component of a pseudocodeword is an integer between 0 and m. The jth component h_j of the vector-weight $\mathbf{h} = (h_1, \dots, h_m)$ of the pseudocodeword in (12.5) is the number of components in the pseudocodeword with value j, i.e.,

$$h_j = \left| \left\{ l \colon \sum_{i=0}^{m-1} x_l^{(i)} = j \text{ and } l \in \{0, \dots, N_{\mathrm{x}} - 1\} \right\} \right|.$$

The PCVWE of constituent code C_{x} can be computed using a nonbinary trellis constructed from the ordinary (information bit-oriented) trellis T_{x}. This trellis will be called the *pseudocodeword trellis* and is denoted by $T^{\mathrm{PC}}_{\mathrm{x},m}$ for constituent code C_{x}. The procedure to construct $T^{\mathrm{PC}}_{\mathrm{x},m}$ from T_{x} is described below.

12.5.1 Constructing $T^{\mathrm{PC}}_{\mathrm{x},m}$ From T_{x}

The pseudocodeword trellis $T^{\mathrm{PC}}_{\mathrm{x},m} = T^{\mathrm{PC}}_{\mathrm{x},m}(V^{\mathrm{PC}}_{\mathrm{x},m}, E^{\mathrm{PC}}_{\mathrm{x},m})$, where $V^{\mathrm{PC}}_{\mathrm{x},m}$ is the vertex set and $E^{\mathrm{PC}}_{\mathrm{x},m}$ is the edge set, can be constructed from the trellis T_{x} in the following way. First, define the sets

$$\begin{aligned}\tilde{V}^{\mathrm{PC}}_{\mathrm{x},m,t} &= \overbrace{V_{\mathrm{x},t} \times V_{\mathrm{x},t} \times \cdots \times V_{\mathrm{x},t}}^{m}\\ \tilde{E}^{\mathrm{PC}}_{\mathrm{x},m,t} &= \left\{((v_{\mathrm{l}}^{(0)},\dots,v_{\mathrm{l}}^{(m-1)}),(v_{\mathrm{r}}^{(0)},\dots,v_{\mathrm{r}}^{(m-1)}))\colon (v_{\mathrm{l}}^{(i)},v_{\mathrm{r}}^{(i)}) \in E_{\mathrm{x},t}, \forall i \in \{0,\dots,m-1\}\right\}\end{aligned}$$

where the time index t runs from 0 to I_{x} (resp. $I_{\mathrm{x}} - 1$) for the vertices (resp. edges). The label of an edge $((v_{\mathrm{l}}^{(0)},\dots,v_{\mathrm{l}}^{(m-1)}),(v_{\mathrm{r}}^{(0)},\dots,v_{\mathrm{r}}^{(m-1)})) \in \tilde{E}^{\mathrm{PC}}_{\mathrm{x},m,t}$ is the integer sum of the labels of its constituent edges $(v_{\mathrm{l}}^{(i)},v_{\mathrm{r}}^{(i)}) \in E_{\mathrm{x},t}$ for all i, $0 \leq i \leq m-1$, which makes the trellis (to be constructed below) nonbinary in general.

Let $\Psi(\cdot)$ denote a permutation that reorders the components of a vertex $(v^{(0)},\dots,v^{(m-1)}) \in \tilde{V}^{\mathrm{PC}}_{\mathrm{x},m,t}$ according to their labels in a nondecreasing order. As an example, for $m = 3$,

$$\Psi(v_1, v_0, v_2) = (v_0, v_1, v_2) \quad \text{and} \quad \Psi(v_2, v_1, v_0) = (v_0, v_1, v_2),$$

assuming that vertex v_i has label i. Now, define the vertex set $V^{\mathrm{PC}}_{\mathrm{x},m,t}$ by expurgating vertices from the vertex set $\tilde{V}^{\mathrm{PC}}_{\mathrm{x},m,t}$ as follows:

$$V^{\mathrm{PC}}_{\mathrm{x},m,t} = \left\{\Psi(v^{(0)},\dots,v^{(m-1)})\colon (v^{(0)},\dots,v^{(m-1)}) \in \tilde{V}^{\mathrm{PC}}_{\mathrm{x},m,t}\right\}.$$

The edge set $E^{\mathrm{PC}}_{\mathrm{x},m,t}$ is defined by expurgating edges from the edge set $\tilde{E}^{\mathrm{PC}}_{\mathrm{x},m,t}$ as follows:

$$\begin{aligned}E^{\mathrm{PC}}_{\mathrm{x},m,t} = \Big\{&(\Psi(v_{\mathrm{l}}^{(0)},\dots,v_{\mathrm{l}}^{(m-1)}),\Psi(v_{\mathrm{r}}^{(0)},\dots,v_{\mathrm{r}}^{(m-1)})),\\ &\qquad\qquad \forall\ ((v_{\mathrm{l}}^{(0)},\dots,v_{\mathrm{l}}^{(m-1)}),(v_{\mathrm{r}}^{(0)},\dots,v_{\mathrm{r}}^{(m-1)})) \in \tilde{E}^{\mathrm{PC}}_{\mathrm{x},m,t}\Big\}\end{aligned}$$

where all duplicated edges (edges with the same left and right vertex and edge label) are expurgated. The final pseudocodeword trellis is constructed by concatenating the trellis sections $T^{\mathrm{PC}}_{\mathrm{x},m,t} = T^{\mathrm{PC}}_{\mathrm{x},m,t}(V^{\mathrm{PC}}_{\mathrm{x},m,t}, E^{\mathrm{PC}}_{\mathrm{x},m,t})$, $t = 0,\dots,I_{\mathrm{x}} - 1$.

As an example, in Figure 12.4, we show both the standard trellis section $T_{\mathrm{x},t}$ (on the left) and the pseudocodeword trellis section $T^{\mathrm{PC}}_{\mathrm{x},m,t}$ for $m = 2$ (on the right), both being invariant of the time index t, of the accumulator code with generator polynomial $1/(1+D)$. Note that for the pseudocodeword trellis section there are two edges with labels $2/1$ and $0/1$, respectively, from the middle vertex to the middle vertex.

12.5 Lemma: *For* $m = 2$, $|V^{\mathrm{PC}}_{\mathrm{x},m,t}| = |V_{\mathrm{x},t}| + \binom{|V_{\mathrm{x},t}|}{2}$ *and* $|E^{\mathrm{PC}}_{\mathrm{x},m,t}| = 2|V_{\mathrm{x},t}|^2 + |V_{\mathrm{x},t}|$. ◁

The proof of this lemma is given in Appendix 12.C.

For a 4-state encoder this means 10 states, and for an 8-state encoder this means 36 states. For an accumulator we only have 3 states as can be seen in Figure 12.4 (the right trellis section). Similar formulas for the number of vertices and edges can be derived for $m > 2$, but are omitted for brevity here.

Now, we define an *equivalence relation* on the set of pseudocodewords as follows. The pseudocodewords $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ are said to be *equivalent* if and only if there exists a positive real number Δ such that $\boldsymbol{\omega}_1 = \Delta \cdot \boldsymbol{\omega}_2$, i.e., they are scaled versions of each other. As a consequence only a single pseudocodeword from an equivalence class can be a vertex of the decoding polytope, which justifies counting equivalence classes only. Running a Viterbi-like algorithm (see Section 12.5.2 below for details) on the pseudocodeword trellis $T^{\mathrm{PC}}_{\mathrm{x},m}$ constructed above, will, in general, count pseudocodewords from the same equivalence class.

However, counting pseudocodewords instead of their equivalence classes does not violate the bounding argument of Section 12.5.5 below, but may lead to a loose bound. For $m = 2$, for instance, these *duplicates* can be removed by a simple procedure which removes all terms of $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}$ with vector-weights $(\mathbf{w}, \mathbf{h}) = ((0, w), (0, h))$.

As a final remark, the issue of counting pseudocodewords from the same equivalence class is not considered in [14, 24] in the context of LDPC code ensembles.

12.5.2 Computing the PCVWE $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}$

The computation of the PCVWE $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}$ for a constituent code C_x can be performed (for a given cover degree m) on the corresponding pseudocodeword trellis $T^{\mathrm{PC}}_{\mathrm{x},m}$, similarly to the computation (on the trellis T_x) of the input-output weight enumerator. The algorithm performs I_x steps in the trellis. At trellis depth t, and for each state s, it computes the partial enumerator $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}(t, s)$ giving the number of paths in the trellis merging to state s at trellis depth t with input vector-weight $\mathbf{w}$ and output vector-weight $\mathbf{h}$. In particular, $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}(t, s)$ is computed from $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}(t-1, s^S(e))$ by considering all edges $e = (s^S(e), s^E(e))$ from starting state $s^S(e)$ (at time $t-1$) to ending state $s^E(e) = s$ (at time t) according to the dynamic programming principle. Finally, $\mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}} = \mathcal{P}^{C_\mathrm{x}}_{\mathbf{w},\mathbf{h}}(I_\mathrm{x}, s_0)$, where s_0 denotes the all-zero state.

The number of required computations per trellis section is $|E^{\mathrm{PC}}_{\mathrm{x},m,t}| \cdot w^m_{\max} \cdot h^m_{\max}$, where $t = 0, \ldots, I_\mathrm{x} - 1$ and $w_{\max}$ (resp. $h_{\max}$) is the maximum entry of $\mathbf{w}$ (resp. $\mathbf{h}$) that we consider. Note that the computational complexity and the memory requirements scale exponentially with the cover degree m ($m = 1$ corresponds to the codeword input-output weight enumerator).

12.5.3 Average Pseudoweight Enumerator With Random Puncturing Pattern $\mathbf{p}$

We assume a random puncturing pattern for $\mathbf{p}$. In particular, the puncturing patterns are sampled uniformly at random from the ensemble of puncturing patterns $\mathbf{p}$ with a fraction of λ

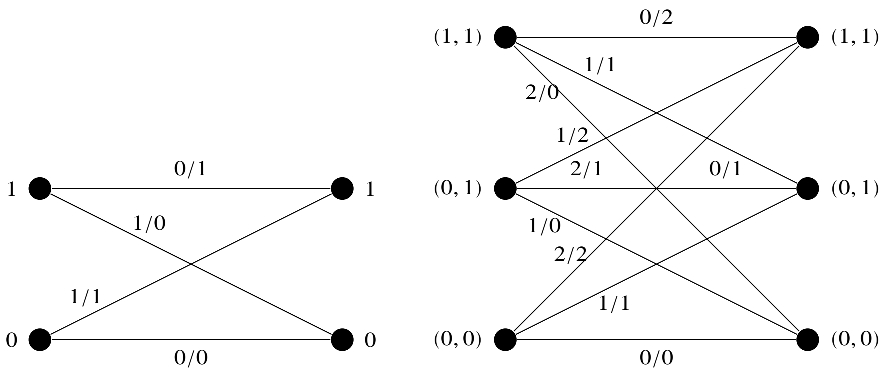


Figure 12.4: The standard trellis section $T_{\mathrm{x},t}$ (on the left) and the pseudocodeword trellis section $T^{\mathrm{PC}}_{\mathrm{x},m,t}$ (on the right), both being invariant of the time index t, of the accumulator code for $m = 2$. Note that for the pseudocodeword trellis section there are two edges with labels $2/1$ and $0/1$, respectively, from the middle vertex to the middle vertex. The vertices of the pseudocodeword trellis section are labeled according to the vertex labeling of the standard trellis section on the left.

ones. Now, using the concept of *uniform interleaver*, the ensemble-average pseudocodeword input-output vector-weight enumerator is

$$\bar{\mathcal{P}}_{\mathbf{w},\mathbf{h}} = \sum_{\mathbf{q},\mathbf{q}_{\mathrm{a}},\mathbf{n}} \frac{\mathcal{P}^{C_{\mathrm{a}}}_{\mathbf{w},\mathbf{q}_{\mathrm{a}}}\mathcal{P}^{C_{\mathrm{b}}}_{\mathbf{w},\mathbf{q}-\mathbf{q}_{\mathrm{a}}}}{\binom{K}{w_1,w_2,\dots,w_m}} \cdot \frac{\left[\prod_{i=1}^{m} \binom{q_i}{n_i}\right]\binom{2K-\sum_{i=1}^{m} q_i}{2\lambda K-\sum_{i=1}^{m} n_i}}{\binom{2K}{2\lambda K}} \cdot \frac{\mathcal{P}^{C_{\mathrm{c}}}_{\mathbf{n},\mathbf{h}-\mathbf{w}-\mathbf{q}+\mathbf{n}}}{\binom{2\lambda K}{n_1,n_2,\dots,n_m}} \tag{12.6}$$

where $\bar{\mathcal{P}}_{\mathbf{w},\mathbf{h}}$ gives the average number (over all interleavers) of unnormalized pseudocodewords of input vector-weight $\mathbf{w}$ and output vector-weight $\mathbf{h}$. In (12.6),

$$\binom{K}{w_1, w_2, \dots, w_m} = \frac{K!}{w_1! \cdots w_m!\,(K - \sum_{i=1}^{m} w_i)!},$$

$\mathbf{q}_{\mathrm{a}}$ is the output vector-weight from the constituent code C_{a}, $\mathbf{q}$ is the total output vector-weight from the outer turbo code, and $\mathbf{n}$ is input vector-weight for the inner constituent code C_{c}.

We remark that (12.6) can be seen as a nonbinary version of [3, Eq. (2)].

Now, the ensemble-average pseudoweight enumerator on channel $\mathscr{H}$ is

$$\bar{\mathcal{P}}^{\mathscr{H}}_{w} = \sum_{\mathbf{w}} \sum_{\substack{\mathbf{h}:\\ w^{\mathscr{H}}(\mathbf{h})=w}} \bar{\mathcal{P}}_{\mathbf{w},\mathbf{h}}$$

where $w^{\mathscr{H}}(\mathbf{h})$ is the weight metric on $\mathscr{H}$. For instance, if $\mathscr{H}$ is the AWGN channel, then

$$w^{\mathscr{H}}(\mathbf{h}) = \left(\sum_{j=1}^{m} j \cdot h_j\right)^2 \Big/ \sum_{j=1}^{m} j^2 \cdot h_j$$

and if $\mathscr{H}$ is the binary erasure channel, then $w^{\mathscr{H}}(\mathbf{h}) = \sum_{j=1}^{m} h_j$.

12.5.4 Average Pseudoweight Enumerator With Regular Puncturing Pattern p

In a similar fashion as for the case with a random puncturing pattern **p**, we can modify [3, Eq. (3)] to arrive at a similar expression (to (12.6)) for the ensemble-average pseudocodeword input-output vector-weight enumerator. Details are omitted for brevity.

12.5.5 Finite-Length Minimum Pseudoweight Analysis

The ensemble-average pseudoweight enumerator $\bar{\mathcal{P}}_w^{\mathcal{H}}$ can be used to bound the minimum pseudoweight on $\mathcal{H}$, denoted by $w_{\min}^{\mathcal{H}}$, of the 3D-TC ensemble in the finite-length regime. In particular, the probability that a code randomly chosen from the ensemble has minimum pseudoweight $w_{\min}^{\mathcal{H}} < \bar{w}$ on $\mathcal{H}$ is upper-bounded by [25]

$$\Pr(w_{\min}^{\mathcal{H}} < \bar{w}) \leq \sum_{w>0}^{<\bar{w}} \bar{\mathcal{P}}_w^{\mathcal{H}}. \tag{12.7}$$

The upper bound in (12.7) can be used to obtain a probabilistic lower bound on the minimum pseudoweight of a code ensemble. For a fixed value of ϵ, where ϵ is any positive value between 0 and 1, we define the probabilistic lower bound with probability ϵ, denoted by $w_{\min,\mathrm{LB},\epsilon}^{\mathcal{H}}$, to be the largest real number $\bar{w}$ such that the right-hand side of (12.7) is at most ϵ. This guarantees that $\Pr(w_{\min}^{\mathcal{H}} \geq \bar{w}) \geq 1 - \epsilon$.

12.6 Searching for the Minimum Pseudoweight

In this section, we present an efficient heuristic to search for low-weight pseudocodewords of 3D-TCs. We use the recently published improved minimum pseudoweight estimation algorithm by Chertkov and Stepanov [12]. In the following, we review that algorithm, restated for 3D-TCs and in a more convenient language.

Recall that the determination of $w_{\min}^{\mathrm{AWGN}}$ amounts to minimizing (12.4) over all nonzero vertices of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$. Some important observations allow us to state an equivalent but simpler problem.

Koetter and Vontobel [26] already noted that

$$w_{\min}^{\mathrm{AWGN}} = \min_{\substack{\boldsymbol{\omega} \in \mathrm{conic}(\dot{\mathcal{Q}}_{\Pi,\Pi_c}) \\ \boldsymbol{\omega} \neq \mathbf{0}}} w^{\mathrm{AWGN}}(\boldsymbol{\omega})$$

where $\mathrm{conic}(\dot{\mathcal{Q}}_{\Pi,\Pi_c})$ is the conic hull of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$ (also termed the fundamental cone). The statement follows immediately from the fact that $w^{\mathrm{AWGN}}(\boldsymbol{\omega}) = w^{\mathrm{AWGN}}(\tau\boldsymbol{\omega})$ for any pseudocodeword $\boldsymbol{\omega}$ and for all $\tau > 0$. The same property allows us to further restrict the search region to the conic section

$$\mathcal{F}_{\mathrm{sec}} = \mathrm{conic}(\dot{\mathcal{Q}}_{\Pi,\Pi_c}) \cap \{\boldsymbol{\omega} \colon \|\boldsymbol{\omega}\|_1 = 1\}$$

because every nonzero pseudocodeword may be scaled to satisfy the normalizing condition without changing the pseudoweight. The benefit of this step is twofold: First, in contrast to (12.6) the domain of optimization now is a polytope that can be stated explicitly by means of (in)equalities. Secondly, minimizing the pseudoweight $w^{\text{AWGN}}(\boldsymbol{\omega}) = \|\boldsymbol{\omega}\|_1^2/\|\boldsymbol{\omega}\|_2^2$ now is equivalent to maximizing $\|\boldsymbol{\omega}\|_2^2$, since the numerator is constant on $\mathscr{F}_{\text{sec}}$.

We are thus in the situation of maximizing a convex function ($\|\cdot\|_2^2$) on a convex polytope. While this is an NP-hard problem in general, the following heuristic proposed in [12] gives very good results in practice.

For $\boldsymbol{\omega} \in \mathscr{F}_{\text{sec}}$, it holds that

$$\|\boldsymbol{\omega}\|_2^2 = \|\boldsymbol{\omega} - \mathbf{1}/N\|_2^2 + \frac{1}{N},$$

where $\mathbf{1}/N = (1, \dots, 1)/N$, i.e., our goal is to maximize, within $\mathscr{F}_{\text{sec}}$, the distance to the central point $\mathbf{1}/N$ (the constant $\frac{1}{N}$ does not affect maximization). Chertkov and Stepanov proposed to first generate a random point $\boldsymbol{\omega}^{(0)} \neq \mathbf{1}/N$ on $\mathscr{F}_{\text{sec}}$, serving as the initial search direction. Then, the linear program

$$\boldsymbol{\omega}^{(i+1)} = \arg\max\, (\boldsymbol{\omega}^{(i)} - \mathbf{1}/N)\boldsymbol{\omega}^T \text{ subject to } \boldsymbol{\omega} \in \mathscr{F}_{\text{sec}}$$

where $(\cdot)^T$ denotes the transpose of its argument, is solved iteratively until the stopping criterion $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)}$ is reached. In each iteration, $\left\|\boldsymbol{\omega}^{(i)} - \mathbf{1}/N\right\|_1$ increases and therefore $\left\|\boldsymbol{\omega}^{(i)} - \mathbf{1}/N\right\|_2$ increases as well, and the result is a local maximum. The search is repeated for an arbitrary number of times in different random directions.

In the case of LDPC codes which are covered in [12], an explicit description of the polytope in question by means of inequalities is available, thus the fundamental cone can be described explicitly as well by omitting those inequalities which are not tight at $\boldsymbol{\omega} = \mathbf{0}$ [26]. This is however not the case for the polytope $\dot{\mathscr{Q}}_{\Pi,\Pi_{\text{c}}}$ which is only implicitly given as the projection of $\mathscr{Q}_{\Pi,\Pi_{\text{c}}}$ onto $\mathbf{y}$. Instead, as we will now show, the cone can be obtained by dropping upper bound constraints on all variables while ensuring that the total flow is equal on all three trellis graphs.

For $\text{x} = \text{a}, \text{b}, \text{c}$, let $\mathscr{Q}_{\text{x}}^{\tau}$ be defined as the set of all $\mathbf{f}^{\text{x}} \in \mathbb{R}_{\geq 0}^{|E_{\text{x}}|}$, where $\mathbb{R}$ is the real numbers, satisfying (12.1b) and the following modified version of (12.1a):

$$\sum_{e \in E_{\text{x},0}} f_e^{\text{x}} = \tau$$

and let

$$\mathscr{F} = \left\{\mathbf{f} = (\mathbf{f}^{\text{a}}, \mathbf{f}^{\text{b}}, \mathbf{f}^{\text{c}}) \colon \exists \tau > 0 \colon \mathbf{f}^{\text{x}} \in \mathscr{Q}_{\text{x}}^{\tau} \text{ for x} = \text{a}, \text{b}, \text{c}\right\}$$

which is, like $\mathscr{Q}$, the set of all network flows in the trellis graphs, but now with an arbitrary positive total flow τ instead of 1. Analogously to $\mathscr{Q}_{\Pi,\Pi_{\text{c}}}$, we define $\mathscr{F}_{\Pi,\Pi_{\text{c}}}$ as the set of $(\tilde{\mathbf{y}}, \mathbf{f})$ where $\tilde{\mathbf{y}} \in \mathbb{R}_{\geq 0}^{N+2\lambda K}$ and $\mathbf{f} = (\mathbf{f}^{\text{a}}, \mathbf{f}^{\text{b}}, \mathbf{f}^{\text{c}}) \in \mathscr{F}$ and additionally (12.2b) is satisfied. The following lemma shows that the projection of $\mathscr{F}_{\Pi,\Pi_{\text{c}}}$ onto $\mathbf{y}$ indeed yields the fundamental cone of 3D-TC LP decoding.

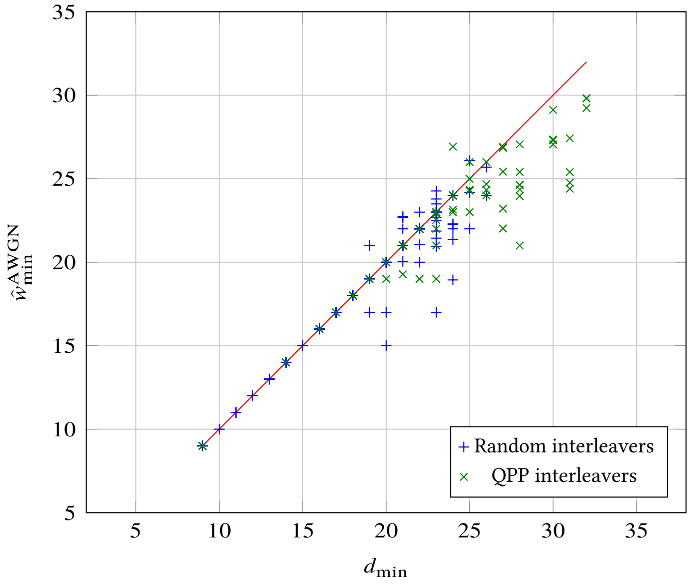


Figure 12.5: Estimated $w_{\min}^{\text{AWGN}}$, using the algorithm from [13] (which is straightforward to apply to 3D-TCs), and exact $d_{\min}$ for 3D-TCs with 100 randomly selected pairs of interleavers (blue plus signs) and with 100 randomly selected pairs of QPP-based interleavers (green x-marks). The diagonal line gives the trivial upper bound of $d_{\min}$ on $w_{\min}^{\text{AWGN}}$ provided by Proposition 12.4. $K = 128$ and $R = 1/3$.

12.6 Lemma: *Let* $\dot{\mathscr{F}}_{\Pi,\Pi_c}$ *be the projection of* $\mathscr{F}_{\Pi,\Pi_c}$ *onto the first* N *variables. Then,* $\dot{\mathscr{F}}_{\Pi,\Pi_c} = \operatorname{conic}(\dot{\mathscr{Q}}_{\Pi,\Pi_c})$. $\triangleleft$

See Appendix 12.D on page 207 for a proof.

12.7 Numerical Results

In this section, we present some numerical results when the interleaver pair (Π, Π_c) is taken from the set of all possible interleaver pairs, and when it is taken from the set of pairs of quadratic permutation polynomials (QPPs) over integer rings. Permutation polynomial based interleavers over integer rings for conventional TCs were first proposed in [27]. These interleavers are fully algebraic and *maximum contention-free* [28], which makes them very suitable for parallel implementation in the turbo decoder. QPP-based interleavers for conventional TCs were also recently adopted for the 3GPP LTE standard [20]. We remark that for the results below, $\lambda = 1/4$ and the regular puncturing pattern $\mathbf{p} = [11000000]$ are assumed. As shown in [3], $\lambda = 1/4$ gives a suitable trade-off between performance in the waterfall and error floor regions. Finally, we emphasize that all the numerical estimates of the $d_{\min}$ and $w_{\min}^{\text{AWGN}}$ given below are actually also upper bounds on the exact values.

12.7.1 Ensemble-Average Results for $K = 128$ and $R = 1/3$

In Figure 12.5, we present the exact $d_{\min}$ and an estimate of $w_{\min}^{\text{AWGN}}$ (which is also an upper bound), denoted by $\hat{w}_{\min}^{\text{AWGN}}$, of unpunctured 3D-TCs with $K = 128$ and with 100 randomly selected pairs of interleavers (Π, Π_c) (blue plus signs). The corresponding results with QPP-based interleaver pairs (and with no constraints on the inverse polynomials) are also displayed (green x-marks). For all codes, except 11, the estimated $w_{\min}^{\text{AWGN}}$ is at most equal to the $d_{\min}$. The values of $w_{\min}^{\text{AWGN}}$ were estimated using the algorithm from [13] (which is straightforward to apply to 3D-TCs) with a signal-to-noise ratio (SNR) of 2.0 dB and 500 evaluations of the algorithm, while the $d_{\min}$ was computed exactly as described in the second paragraph following the proof of Proposition 12.4. Note that when the $d_{\min}$ is strictly smaller than the estimated $w_{\min}^{\text{AWGN}}$ (points above the diagonal line), the estimation algorithm from [13] was unable to provide an estimate that beats the trivial upper bound provided by Proposition 12.4. From the figure, it follows that QPPs give *better codes* (can provide a higher $d_{\min}$ and a higher $\hat{w}_{\min}^{\text{AWGN}}$), and that $w_{\min}^{\text{AWGN}}$ is strictly lower than $d_{\min}$ for most codes when the $d_{\min}$ is large. As a side remark, the algorithm from Section 12.6 gives slightly worse results (the average $\hat{w}_{\min}^{\text{AWGN}}$ increases by approximately 0.05) than with the algorithm from [13] with the same number of runs (500) per instance. However, the algorithm from Section 12.6 is significantly faster.

12.7.2 Exhaustive/Random Search Optimizing $w_{\min}^{\text{AWGN}}$

In this subsection, we present the results of a computer search for pairs of QPPs with a quadratic inverse for $K = 128$, 256, and 320 for unpunctured $R = 1/3$ 3D-TCs. The objective of the search was to find pairs of QPPs giving a large estimated $w_{\min}^{\text{AWGN}}$. To speed up the search, an adaptive threshold on the minimum AWGN pseudoweight $w_{\min}^{\text{AWGN}}$ was set in the search, in the sense that if a pseudocodeword of AWGN pseudoweight smaller than the threshold was found, then this particular candidate pair of QPPs was rejected.

For $K = 128$, we performed an exhaustive search over all 2^{17} pairs of QPPs (with a quadratic inverse). The minimum AWGN pseudoweight was estimated using the algorithm from [13] (which is straightforward to apply to 3D-TCs) with an SNR of 1.7 dB and 500 evaluations of the algorithm.

In Figure 12.6, we plot the exact $d_{\min}$ (red circles), the exact $h_{\min}$ (green x-marks), and $\hat{w}_{\min}^{\text{AWGN}}$ (blue plus signs) of the 75 3D-TCs with the best $\hat{w}_{\min}^{\text{AWGN}}$. For each point in the figure, the x-coordinate corresponds to the sample index (the results are ordered by increasing $d_{\min}$), while the y-coordinate is either the exact $d_{\min}$, the exact $h_{\min}$, or $\hat{w}_{\min}^{\text{AWGN}}$. From the figure, we observe that the best $w_{\min}^{\text{AWGN}}$ (which is at most 30.2139) is strictly smaller than the best possible $d_{\min}$ or $h_{\min}$. The best possible $d_{\min}$ was established to be 38 (exhaustive search), and for this particular code $h_{\min} = 36$, but the estimate of $w_{\min}^{\text{AWGN}}$ is not among the 75 best; it is only 29.6042 (see Table 12.7 which shows the results of an exhaustive/random search optimizing the $d_{\min}$ for pairs of QPPs with a quadratic inverse).

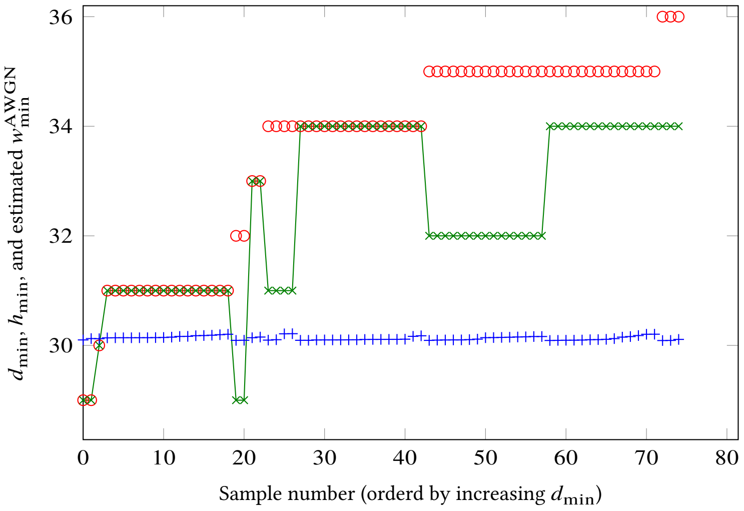


Figure 12.6: Exact $d_{\min}$ (red circles), exact $h_{\min}$ (green x-marks), and $\hat{w}_{\min}^{\mathrm{AWGN}}$ (blue plus signs) of the 75 best (in terms of $\hat{w}_{\min}^{\mathrm{AWGN}}$) QPP-based interleaver pairs for the 3D-TC with input block length $K = 128$ and code rate $R = 1/3$.

For $K = 256$, only a partial search has been conducted. The largest found value for $\hat{w}_{\min}^{\mathrm{AWGN}}$, after taking about 180000 samples, which is close to 17% of the whole space, is 43.0335. As for $K = 128$, the minimum AWGN pseudoweight was estimated using the algorithm from [13] (which is straightforward to apply to 3D-TCs) with an SNR of 1.7 dB and 500 evaluations of the algorithm.

For $K = 320$, we again performed an exhaustive search over the 2^{18} pairs of QPPs with a quadratic inverse. This time we used the algorithm presented in Section 12.6 with 500 iterations per code. The largest estimated minimum pseudoweight $\hat{w}_{\min}^{\mathrm{AWGN}}$ that we found was 46.0612, which is considerably larger than that for the code in Table 12.7 (which shows the results of an exhaustive/random search optimizing the $d_{\min}$ for pairs of QPPs with a quadratic inverse).

12.7.3 Exhaustive/Random Search Optimizing $d_{\min}$

We also performed an exhaustive/random search optimizing the $d_{\min}$ for pairs of QPPs with a quadratic inverse for selected values of K for unpunctured $R = 1/3$ 3D-TCs. For $K = 128$, 160, 192, and 208, the search was exhaustive, in the sense that each pair of interleavers was looked at. In the search, the $d_{\min}$ was estimated using the triple impulse method [29]. The results are given in Table 12.7 for selected values of K, where $f(x) = f_1 x + f_2 x^2 \pmod{K}$ generates the TC interleaver, $\tilde{f}(x) = \tilde{f}_1 x + \tilde{f}_2 x^2 \pmod{N_c}$ generates the permutation in the patch, and $\hat{d}_{\min}$ and $\hat{w}_{\min}^{\mathrm{AWGN}}$ denote the estimated $d_{\min}$ and the estimated $w_{\min}^{\mathrm{AWGN}}$, respectively. The estimates of $w_{\min}^{\mathrm{AWGN}}$ were obtained by using the algorithm from [13] (which is straightforward to apply to 3D-TCs) with SNR and number of evaluations of the algorithm given in the ninth and tenth

K	f_1	f_2	N_{c}	$\tilde{f}_1$	$\tilde{f}_2$	$\hat{d}_{\min}$	$\hat{w}_{\min}^{\mathrm{AWGN}}$	SNR	Evaluations
128 [a]	55	96	64	9	16	38 [b]	29.6042	2.0 dB	2000
160 [a]	131	60	80	9	20	42 [b]	30.0000	1.7 dB	500
192 [a]	35	24	96	11	12	46	32.9046	1.7 dB	500
208 [a]	165	182	104	37	26	49	36.3370	1.7 dB	500
256	239	192	128	37	32	52	42.7816	1.7 dB	500
320	183	280	160	57	20	58	41.3818	1.7 dB	500
512 [c]	175	192	256	15	192	67	45.5872	2.0 dB	500

Table 12.7: Results from an exhaustive/random search for pairs of QPPs with $\lambda = 1/4$, both with a quadratic inverse, in which the first QPP $f(x) = f_1 x + f_2 x^2 \pmod{K}$ generates the TC interleaver and the second QPP $\tilde{f}(x) = \tilde{f}_1 x + \tilde{f}_2 x^2 \pmod{N_{\mathrm{c}}}$ generates the permutation in the patch. Moreover, terms like "SNR" are explained in the text.

---

[a] Exhaustive search, which implies that the corresponding $\hat{d}_{\min}$ is an upper bound on the optimum $d_{\min}$ (the true optimum $d_{\min}$ when the estimate $\hat{d}_{\min}$ is exact) for this input block length.

[b] This is the exact $d_{\min}$, and we can observe a large gap between $d_{\min}$ and $w_{\min}^{\mathrm{AWGN}}$.

[c] The QPPs are taken from [3].

column of the table, respectively. Finally, we remark that the codes in the first and second rows, for $K = 128$ and 160, are $d_{\min}$-optimal, in the sense that there does not exist any pair of QPPs (with a quadratic inverse) giving a $d_{\min}$ strictly larger than 38 and 42, respectively, for the unpunctured 3D-TC.

12.7.4 Ensemble-Average Results for Various K and $R = 1/3$

In Figure 12.8, we present the average estimated (now using the algorithm from Section 12.6) minimum AWGN pseudoweight of 3D-TCs for $K = 128$, 160, 192, 208, 256, 320, 512, 640, 768, 1024, and 1504. Both random interleaver pairs and QPP-based (with a quadratic inverse) interleaver pairs have been considered. In both cases, we generated 40 interleaver pairs of each size. For each code we ran $K/10$ trials of the estimation algorithm described in Section 12.6. From Figure 12.8, we observe that the average $\hat{w}_{\min}^{\mathrm{AWGN}}$ grows with K for both random interleaver pairs and QPP-based interleaver pairs. For all values of K, as expected, the average $\hat{w}_{\min}^{\mathrm{AWGN}}$ is higher for QPP-based interleaver pairs than for random interleaver pairs. As a comparison, we have also plotted the corresponding theoretical values $w_{\min,\mathrm{LB},0.5}^{\mathrm{AWGN}}$ from Section 12.5 (using (12.7)) for graph cover degree 2. Also, for comparison, we have plotted the corresponding lower bounds on the $d_{\min}$ and the $h_{\min}$ using a similar ensemble analysis as the one from Section 12.5. For details, we refer the interested reader to [3, 4]. Note that the curves coincide for small values of K. The reason that the curve for the probabilistic lower bound on $w_{\min}^{\mathrm{AWGN}}$ of the 3D-TC ensemble is higher than the corresponding curve for $h_{\min}$ is that the cover degree is limited to $m = 2$. In general, the pseudocodewords with support set equal to a small-size stopping set which is not a codeword have a cover degree which is quite large. We would expect that the curve for the probabilistic lower bound on $w_{\min}^{\mathrm{AWGN}}$ of the 3D-TC ensemble go further

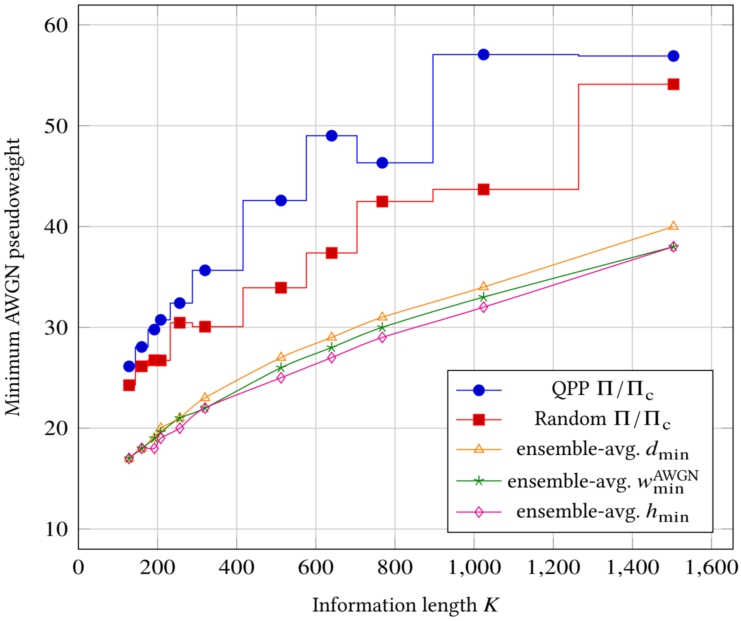


Figure 12.8: The average estimated minimum AWGN pseudoweight for 3D-TCs for different information block lengths K, both for QPP-based (with a QPP inverse) and random interleaver pairs. The lower curves show the probabilistic lower bounds on d_{min}, $w_{\text{min}}^{\text{AWGN}}$, and h_{min} of the 3D-TC ensemble (for cover degrees of at most $m = 2$).

down for m larger than 2. However, it is currently unfeasible to do the actual computations for larger values of m (both the computational complexity and the memory requirements scale exponentially with m). For ease of computation we have used random puncturing patterns $\mathbf{p}$ to compute the curves, while the estimated average values are for regular patterns which in general give better results.

12.8 Conclusion

In this work, we performed a minimum pseudoweight analysis of pseudocodewords of (relaxed) LP decoding of 3D-TCs, adapting the LP relaxation proposed by Feldman in his thesis for conventional TCs. We proved that the 3D-TC polytope is proper and C-symmetric, and made a connection to finite graph covers of the 3D-TC factor graph. This connection was used to show that the support set of any pseudocodeword is a stopping set (as defined in [4, Def. 1]), and enabled a finite-length minimum pseudoweight analysis. Furthermore, an explicit description of the fundamental cone of the 3D-TC polytope was given. Finally, both a theoretical and an extensive numerical study of the minimum AWGN pseudoweight of small-to-medium block length 3D-TCs was presented, which showed that 1) typically (i.e., in most cases) when the d_{min} and/or the h_{min} is high, $w_{\text{min}}^{\text{AWGN}}$ is strictly smaller than both the d_{min} and the h_{min} for these codes, and 2) that $w_{\text{min}}^{\text{AWGN}}$ grows with the block length, at least for small-to-medium block lengths. For instance, the exhaustive search for $K = 128$ over the entire class of QPP-based interleaver pairs (with a quadratic inverse) revealed that the best minimum AWGN pseudoweight is strictly smaller than the best minimum/stopping distance. It is expected that the $w_{\min}^{\mathrm{AWGN}}$ will dominate the decoding performance for high SNRs.

12.A Proof of Proposition 12.1

We first prove a more general result and then show how it applies to our case.

12.7 Lemma: *Let C_δ, $\delta = 1, \dots, \Delta$, be linear block codes of the same length N and let $C = \bigcap_{\delta=1}^{\Delta} C_\delta$. Then,*

(1) $\operatorname{conv}(C_\delta)$ is proper and C_δ-symmetric for all δ, and

(2) $\bigcap_{\delta=1}^{\Delta} \operatorname{conv}(C_\delta)$ is proper and C-symmetric. ◁

Proof. (1) The convex hull $\operatorname{conv}(C_\delta)$ is proper because all codewords are by definition vertices of the polytope. Moreover, because no vertex of the unit hypercube is the convex combination of others, $\operatorname{conv}(C_\delta)$ cannot contain any other integral points. To show C_δ-symmetry, choose $\mathbf{a} \in \operatorname{conv}(C_\delta)$ and $\mathbf{c} \in C_\delta$ arbitrarily. By construction, $\mathbf{a}$ can be written as a convex combination of codewords of C_δ, i.e.,

$$\mathbf{a} = \sum_{i=1}^{|C_\delta|} \lambda_i \mathbf{c}_i \quad \text{where} \sum_{i=1}^{|C_\delta|} \lambda_i = 1 \text{ and } \lambda_i \geq 0.$$

We claim that

$$|\mathbf{a} - \mathbf{c}| = \sum_{i=1}^{|C_\delta|} \lambda_i (\mathbf{c}_i \oplus \mathbf{c}) = \sum_{i=1}^{|C_\delta|} \lambda_i \tilde{\mathbf{c}}_i \tag{12.8}$$

where $\oplus$ denotes integer addition modulo 2 and $\tilde{\mathbf{c}}_i$ is the ith codeword of C_δ using a different ordering. This would imply C_δ-symmetry, i.e., if $\mathbf{a} \in \operatorname{conv}(C_\delta)$ and $\mathbf{c} \in C_\delta$, then $|\mathbf{a} - \mathbf{c}| \in \operatorname{conv}(C_\delta)$.

Let a_j, c_j, and $c_{i,j}$ denote the jth coordinate of $\mathbf{a}$, $\mathbf{c}$, and $\mathbf{c}_i$, respectively. The first equality in (12.8) follows for $c_j = 0$ from

$$\left|a_j - c_j\right| = a_j = \sum_{i=1}^{|C_\delta|} \lambda_i c_{i,j} = \sum_{i=1}^{|C_\delta|} \lambda_i (c_{i,j} \oplus c_j)$$

and for $c_j = 1$ because

$$\left|a_j - c_j\right| = 1 - a_j = \sum_{i=1}^{|C_\delta|} \lambda_i (1 - c_{i,j}) = \sum_{i=1}^{|C_\delta|} \lambda_i (c_{i,j} \oplus c_j).$$

The second part of (12.8) holds because $\mathbf{c}_i \oplus C_\delta = C_\delta$ due to the linearity of C_δ.

(2) Let $\mathscr{P} = \bigcap_{\delta=1}^{\Delta} \operatorname{conv}(C_\delta)$. The properness of $\mathscr{P}$ for C follows immediately from the properness of $\operatorname{conv}(C_\delta)$, $\delta = 1, \dots, \Delta$, and the definition of C. Now, if $\mathbf{a} \in \mathscr{P}$ and $\mathbf{c} \in C$, then for $\delta = 1, \dots, \Delta$ we have $\mathbf{a} \in \operatorname{conv}(C_\delta)$ and $\mathbf{c} \in C_\delta$, so by 1) $|\mathbf{a} - \mathbf{c}| \in \operatorname{conv}(C_\delta)$ and thus $|\mathbf{a} - \mathbf{c}| \in \mathscr{P}$. □

Now, let C be a 3D-TC. By $\tilde{C}$ we denote the code of length $N + 2\lambda K$ obtained by appending the hidden parity bits from C_a and C_b which are sent to the patch. For $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$ we define a supercode $\tilde{C}_\mathrm{x}$ of $\tilde{C}$ by unconstraining all bits that are not connected to the constituent code C_x, i.e., $\tilde{\mathbf{x}} \in \tilde{C}_\mathrm{x}$ if and only if $(\tilde{x}_{\rho_\mathrm{x}(0)}, \dots, \tilde{x}_{\rho_\mathrm{x}(N_\mathrm{x}-1)}) \in C_\mathrm{x}$, where N_x is the block length of C_x and $\rho_\mathrm{x}(\cdot)$ is defined in (12.2b), and $\tilde{x}_i \in \{0, 1\}$ for all remaining i. Observe that $\tilde{C}_\mathrm{a} \cap \tilde{C}_\mathrm{b} \cap \tilde{C}_\mathrm{c} = \tilde{C}$.

Next, define polytopes $\mathcal{Q}^{\mathrm{x}}_{\Pi,\Pi_\mathrm{c}}$ that are obtained from $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$ by dropping in (12.2b) all constraints not corresponding to C_x, and let $\tilde{\mathcal{Q}}^{\mathrm{x}}_{\Pi,\Pi_\mathrm{c}}$ be the projection of $\mathcal{Q}^{\mathrm{x}}_{\Pi,\Pi_\mathrm{c}}$ onto the $\tilde{\mathbf{y}}$ variables. Finally, define $\tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ in analogy to $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ as the projection of $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$ onto the first $N + 2\lambda K$ variables.

Due to the trellis structure, it is easily seen that $\tilde{\mathcal{Q}}^{\mathrm{x}}_{\Pi,\Pi_\mathrm{c}} = \operatorname{conv}(\tilde{C}_\mathrm{x})$ for $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$, and by comparing the polytope definitions we see that $\tilde{\mathcal{Q}}^{\mathrm{a}}_{\Pi,\Pi_\mathrm{c}} \cap \tilde{\mathcal{Q}}^{\mathrm{b}}_{\Pi,\Pi_\mathrm{c}} \cap \tilde{\mathcal{Q}}^{\mathrm{c}}_{\Pi,\Pi_\mathrm{c}} = \tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$. Applying Lemma 12.7 shows that $\tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ is both proper and $\tilde{C}$-symmetric. Now, C is the projection of $\tilde{C}$ onto the first N variables, and $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ the corresponding projection of $\tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$.

To show that $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ is proper, first observe that the projection of any $\tilde{\mathbf{c}} \in \tilde{C}$ onto the first N variables is obviously contained in $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$. Conversely, let $\mathbf{a} \in \dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}} \cap \{0, 1\}^N$. By definition there exists $\tilde{\mathbf{a}} \in \tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ such that $\tilde{\mathbf{a}} = (\mathbf{a}, \hat{\mathbf{a}})$. In order to show that $\hat{\mathbf{a}}$ is integral, note that the systematic part of $\tilde{\mathbf{a}}$ is contained in $\mathbf{a}$ and thus integral. Again by the trellis structure, this implies a unique and integral configuration of the flow variables in T_a and T_b, and consequently also the variables according to the output of those encoders, including the hidden bits sent to the patch, must be integral. Because $\tilde{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$ is proper it follows that $\tilde{\mathbf{a}} \in \tilde{C}$ and thereby also $\mathbf{a} \in C$, which proves properness. Finally, note that C-symmetry is trivially preserved by projections, which concludes the proof.

12.B Proof of Proposition 12.3

Let $\mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_\mathrm{c}} \subset \mathcal{Q}$ be the projection of $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$ onto $\mathbf{f}$. We call a flow $\mathbf{f} \in \mathcal{Q}$ *agreeable* if $\mathbf{f} \in \mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_\mathrm{c}}$. For an agreeable flow $\mathbf{f}$ let $\tilde{\mathbf{y}} = \tilde{\mathbf{y}}(\mathbf{f})$ be the uniquely determined element of $[0, 1]^{N+2\lambda K}$ such that $(\tilde{\mathbf{y}}, \mathbf{f}) \in \mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$. Analogously, $\mathbf{y}(\mathbf{f})$ is the projection of $\tilde{\mathbf{y}}(\mathbf{f})$ onto the first N variables. Note that $\tilde{\mathbf{y}}(\mathbf{f})$ (and $\mathbf{y}(\mathbf{f})$) can be read off from $\mathbf{f}$ by (12.2b).

For $\mathbf{f} = (\mathbf{f}^\mathrm{a}, \mathbf{f}^\mathrm{b}, \mathbf{f}^\mathrm{c}) \in \mathcal{Q}$, but not necessarily $\mathbf{f} \in \mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_\mathrm{c}}$, we can still use (12.2b) to deduce local values of $\tilde{\mathbf{y}}$. More precisely, if we define

$$\Phi(\mathrm{x}) = \{j \colon j = \rho_\mathrm{x}(\phi_\mathrm{x}(t, i)) \text{ for some } (t, i)\}$$

then for all $j \in \Phi(\mathrm{x})$ we can deduce $\tilde{y}_j^{\mathrm{x}}(\mathbf{f}) = \sum_{\substack{e \in E_{\mathrm{x},t}: \\ c_i(e)=1}} f_e^{\mathrm{x}}$ where $t \in \{0, \dots, I_{\mathrm{x}} - 1\}$ and $i \in \{0, \dots, n_{\mathrm{x},t} - 1\}$ are determined by (12.2b). This implies that $\mathbf{f} \in \mathcal{Q}$ is agreeable if and only if

$$\tilde{y}_j^{\mathrm{x}}(\mathbf{f}) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}) \tag{12.9}$$

for all $(j, \mathrm{x}, \mathrm{x}')$ such that $j \in \Phi(\mathrm{x}) \cap \Phi(\mathrm{x}')$ and $(\mathrm{x}, \mathrm{x}') \in \{(\mathrm{a}, \mathrm{b}), (\mathrm{a}, \mathrm{c}), (\mathrm{b}, \mathrm{c})\}$, where the first case amounts to the outer interleaver Π and the remaining cases are due to the connections to the patch from C_{a} and C_{b}, respectively, via Π_{c}. We denote the set of these triples $(j, \mathrm{x}, \mathrm{x}')$ by $\mathcal{A}$.

12.8 Lemma: *The relation* $\mathcal{P}_{\Pi,\Pi_{\mathrm{c}}} \subseteq \dot{\mathcal{Q}}_{\Pi,\Pi_{\mathrm{c}}}$ *holds.* ◁

Proof. Let $\boldsymbol{\omega}(\mathbf{x}^{(m)}) \in \mathcal{P}_{\Pi,\Pi_{\mathrm{c}}}$ be a graph-cover pseudocodeword of C, i.e., there exists a degree-m cover code $C^{(m)}$ of C such that $\mathbf{x}^{(m)}$ is a codeword of $C^{(m)}$. As before, we can extend $\mathbf{x}^{(m)}$ to

$$\tilde{\mathbf{x}}^{(m)} = \left(\tilde{x}_0^{(0)}, \dots, \tilde{x}_{N+2\lambda K-1}^{(0)}, \dots, \tilde{x}_0^{(m-1)}, \dots, \tilde{x}_{N+2\lambda K-1}^{(m-1)}\right)$$

by appending the parity bits of the copies of C_{a} and C_{b} that are sent to copies of C_{c}.

For each $l = 0, \dots, m-1$, $\left(\tilde{x}_0^{(l)}, \dots, \tilde{x}_{N+2\lambda K-1}^{(l)}\right)$ induces via trellis encoding a flow $\mathbf{f}_{\boldsymbol{\omega}}^{(l)}$ in $\mathcal{Q}$ with entries only from $\{0, 1\}$. In general, $\mathbf{f}_{\boldsymbol{\omega}}^{(l)}$ is not agreeable because the connections are mixed with different copies in the cover graph. However, from the definition of a graph cover we can conclude that

$$\tilde{y}_j^{\mathrm{x}}(\mathbf{f}_{\boldsymbol{\omega}}^{(l)}) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_{\boldsymbol{\omega}}^{(\pi_j(l))}) \tag{12.10}$$

for all $(j, \mathrm{x}, \mathrm{x}') \in \mathcal{A}$ and all $l = 0, \dots, m-1$, where π_j is the corresponding permutation introduced by the graph cover, either (in the case $\mathrm{x} = \mathrm{a}$ and $\mathrm{x}' = \mathrm{b}$) on connections from an input vertex of $\Gamma(C_{\mathrm{a}})$ to a check vertex of $\Gamma(C_{\mathrm{b}})$ or (if $\mathrm{x}' = \mathrm{c}$) on connections from a parity vertex of C_{a} or C_{b} to a check vertex of $\Gamma(C_{\mathrm{c}})$.

We claim that

$$\mathbf{f}_{\boldsymbol{\omega}} = \frac{1}{m} \sum_{l=0}^{m-1} \mathbf{f}_{\boldsymbol{\omega}}^{(l)} \tag{12.11}$$

is agreeable and that $\mathbf{y}(\mathbf{f}_{\boldsymbol{\omega}}) = \boldsymbol{\omega}(\mathbf{x}^{(m)})$.

First, note that $\mathbf{f}_{\boldsymbol{\omega}}$ is a convex combination of elements from the convex set $\mathcal{Q}$, so $\mathbf{f}_{\boldsymbol{\omega}} \in \mathcal{Q}$ as well. To prove agreeability, we verify (12.9) for all $(j, \mathrm{x}, \mathrm{x}') \in \mathcal{A}$:

$$\begin{aligned}
\tilde{y}_j^{\mathrm{x}}(\mathbf{f}_{\boldsymbol{\omega}}) &= \sum_{\substack{e \in E_{\mathrm{x},t}: \\ c_i(e)=1}} f_{\boldsymbol{\omega},e}^{\mathrm{x}} &&= \sum_{\substack{e \in E_{\mathrm{x},t}: \\ c_i(e)=1}} \frac{1}{m} \sum_{l=0}^{m-1} f_{\boldsymbol{\omega},e}^{\mathrm{x},(l)} \\
&= \frac{1}{m} \sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_{\boldsymbol{\omega}}^{(l)}) &&= \frac{1}{m} \sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_{\boldsymbol{\omega}}^{(\pi_j(l))}) \\
&= \frac{1}{m} \sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_{\boldsymbol{\omega}}^{(l)}) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_{\boldsymbol{\omega}})
\end{aligned}$$

where we have used (12.2b), (12.11), and (12.10). The second-to-last equality follows since π_j is a permutation of $\{0, \dots, m-1\}$. This shows that $\mathbf{f}_\omega$ is agreeable and thus $\mathbf{y}(\mathbf{f}_\omega)$ is well-defined.

Now, fix $j \in \{0, \dots, N-1\}$ and pick any x such that $j \in \Phi(\mathrm{x})$. Then

$$y_j(\mathbf{f}_\omega) = \frac{1}{m} \sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_\omega^{(l)}) = \frac{1}{m} \sum_{l=0}^{m-1} x_j^{(l)} = \frac{1}{m} \left|\{l : x_j^{(l)} = 1\}\right| = \omega_j(\mathbf{x}^{(m)})$$

which concludes the proof. □

Before proving the other direction for rational points, we first show part (2) of Proposition 12.3.

12.9 Lemma: *All vertices of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$ have rational entries.* ◁

Proof. Let $\mathbf{y}$ be a vertex of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$. Because $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$ is a projection of the polytope $\mathcal{Q}_{\Pi,\Pi_c}$, and $\mathcal{Q}_{\Pi,\Pi_c}$ is the image of $\mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_c}$ under a linear map, there exists some vertex $\mathbf{f}$ of $\mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_c}$ such that $(\tilde{\mathbf{y}}(\mathbf{f}), \mathbf{f})$ is also a vertex of $\mathcal{Q}_{\Pi,\Pi_c}$ and $\mathbf{y}$ is the projection of $\tilde{\mathbf{y}}$ onto the first N variables.

Now, $\mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_c}$ is a rational polyhedron (i.e., it is defined by (in)equalities with rational entries only), so every vertex of $\mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_c}$ is rational [30, p. 123]. Since by (12.2b) each $\tilde{y}_j$ is just a sum of entries of $\mathbf{f}$ for each $j = 0, \dots, N + 2\lambda K - 1$, $\tilde{\mathbf{y}}$ and thus $\mathbf{y}$ must be rational as well. □

12.10 Lemma: *For every $\mathbf{y} \in \dot{\mathcal{Q}}_{\Pi,\Pi_c} \cap \mathbb{Q}^N$ there exists a rational point $(\tilde{\mathbf{y}}, \mathbf{f}) \in \mathcal{Q}_{\Pi,\Pi_c}$ such that $\mathbf{y} = \mathbf{y}(\mathbf{f})$.* ◁

Proof. Let $\mathbf{y}$ be a rational point of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$. Because $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$ is a polytope, $\mathbf{y}$ can be written as a convex combination of vertices of $\dot{\mathcal{Q}}_{\Pi,\Pi_c}$, i.e., $\mathbf{y} = \sum_{k=0}^{d} \lambda_k \mathbf{y}^k$ where $\lambda_k \geq 0$ for $k = 0, \dots, d$ and $\sum_{k=0}^{d} \lambda_k = 1$. Furthermore, by Carathéodory's theorem (e.g., [31, p. 94]), this is even possible with some $d \leq N$ such that the $\mathbf{y}^k$, $k = 0, \dots, d$, are affinely independent. Consequently, λ is the unique solution of the system

$$\begin{pmatrix} \mathbf{y}^0 & \mathbf{y}^1 & \dots & \mathbf{y}^d \\ 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_d \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 1 \end{pmatrix}$$

and by applying Cramer's rule for solving linear equation systems (and Lemma 12.9 which guarantees that all $\mathbf{y}^k$ have rational entries) we see that $\lambda_k \in \mathbb{Q}$ for $k = 0, \dots, d$. Furthermore, the proof of Lemma 12.9 tells us that for each $\mathbf{y}^k$ there is a rational $\mathbf{f}^k \in \mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_c}$ such that $\mathbf{y}^k = \mathbf{y}(\mathbf{f}^k)$. The flow $\mathbf{f} = \sum_{k=0}^{d} \lambda_k \mathbf{f}^k$ satisfies $\mathbf{y} = \mathbf{y}(\mathbf{f})$ (because $\mathbf{y}(\cdot)$ is linear) and $(\tilde{\mathbf{y}}(\mathbf{f}), \mathbf{f}) \in \mathcal{Q}_{\Pi,\Pi_c}$ is rational, which concludes the proof. □

Now, we are able to prove the missing counterpart to Lemma 12.8.

12.11 Lemma: *It holds that $\dot{\mathcal{Q}}_{\Pi,\Pi_c} \cap \mathbb{Q}^N \subseteq \mathcal{P}_{\Pi,\Pi_c}$.* ◁

Proof. Let $\mathbf{y}$ be a rational point of $\dot{\mathcal{Q}}_{\Pi,\Pi_\mathrm{c}}$. By Lemma 12.10, there exist rational $\mathbf{f} \in \mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_\mathrm{c}}$ and rational $\tilde{\mathbf{y}} = (\mathbf{y}, \hat{\mathbf{y}})$ such that $(\tilde{\mathbf{y}}, \mathbf{f}) \in \mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$. Let m be the least common denominator of the entries of $\mathbf{f}$. Then, $\mathbf{f}_m = m\mathbf{f}$ is a flow with integral values between 0 and m. Applying the flow decomposition theorem [32, p. 80] in this context guarantees that $\mathbf{f}_m$ can be split up into m binary flows, i.e.,

$$\mathbf{f}_m = \sum_{l=0}^{m-1} \mathbf{f}_m^{(l)} \tag{12.12}$$

where $\mathbf{f}_m^{(l)}$ has entries from $\{0, 1\}$ and represents a valid path for each trellis T_x, $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$.

Because $\mathbf{f} \in \mathcal{Q}^{\mathbf{f}}_{\Pi,\Pi_\mathrm{c}}$, we conclude from (12.9) that $\tilde{y}_j^{\mathrm{x}}(\mathbf{f}) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f})$ for all $(j, \mathrm{x}, \mathrm{x}') \in \mathcal{A}$. This is equivalent (by linearity) to $\tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_m)$ which by (12.12) means that $\sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m^{(l)}) = \sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_m^{(l)})$. Because all $\mathbf{f}_m^{(l)}$ are $\{0, 1\}$-valued, this last equation implies

$$|\{l \colon \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m^{(l)}) = 1\}| = |\{l \colon \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_m^{(l)}) = 1\}|$$

and consequently for each $(j, \mathrm{x}, \mathrm{x}') \in \mathcal{A}$ a permutation π_j on $\{0, \dots, m-1\}$ can be chosen such that $\tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m^{(l)}) = \tilde{y}_j^{\mathrm{x}'}(\mathbf{f}_m^{(\pi_j(l))})$ for all $l = 0, \dots, m-1$. These π_j define an m-cover $\Gamma^{(m)}(C)$ of $\Gamma(C)$, and by construction

$$\mathbf{x}^{(m)} = (x_0^{(0)}, \dots, x_{N-1}^{(0)}, \dots, x_0^{(m-1)}, \dots, x_{N-1}^{(m-1)})$$

is a codeword of $C^{(m)}$, where we define $x_j^{(l)} = \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m^{(l)})$ for the first x among $(\mathrm{a}, \mathrm{b}, \mathrm{c})$ such that $j \in \Phi(\mathrm{x})$. Finally, we see that

$$\begin{aligned}\omega_j(\mathbf{x}^{(m)}) &= \frac{1}{m}\sum_{l=0}^{m-1} x_j^{(l)} = \frac{1}{m}\sum_{l=0}^{m-1} \tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m^{(l)}) \quad \text{(for some x)}\\ &= \frac{1}{m}\tilde{y}_j^{\mathrm{x}}(\mathbf{f}_m) = \tilde{y}_j^{\mathrm{x}}(\mathbf{f}) = y_j\end{aligned}$$

(by definition of $\mathcal{Q}_{\Pi,\Pi_\mathrm{c}}$) for any $j = 0, \dots, N-1$, which shows that $\boldsymbol{\omega}(\mathbf{x}^{(m)}) = \mathbf{y}$. □

12.C Proof of Lemma 12.5

The number of vertices in $V^{\mathrm{PC}}_{\mathrm{x},2,t}$ will be the number of distinct ordered 2-tuples of integers modulo $|V_{\mathrm{x},t}|$, which is $|V_{\mathrm{x},t}| + \binom{|V_{\mathrm{x},t}|}{2}$.

Now, let us consider the number of edges, and in particular, the edges from vertex $\Psi(v_\mathrm{l}, u_\mathrm{l})$ to vertex $\Psi(v_\mathrm{r}, u_\mathrm{r})$ in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$, where $v_\mathrm{l}, u_\mathrm{l}, v_\mathrm{r}, u_\mathrm{r} \in V_{\mathrm{x},t}$. We have four possible constituent edges to consider, namely the edges $v_\mathrm{l} \xrightarrow{a\,/\,b} v_\mathrm{r}$, $v_\mathrm{l} \xrightarrow{c\,/\,d} u_\mathrm{r}$, $u_\mathrm{l} \xrightarrow{e\,/\,f} u_\mathrm{r}$, and $u_\mathrm{l} \xrightarrow{g\,/\,h} v_\mathrm{r}$ where the labels above the arrows are the input/output labels. Note that $a \neq c$ and $e \neq g$ when $v_\mathrm{r} \neq u_\mathrm{r}$. Also, the (integer) label of a vertex $v \in V_{\mathrm{x},t}$ will be denoted by $\ell(v)$.

- Case $v_\mathrm{l} \neq u_\mathrm{l}$ and $v_\mathrm{r} \neq u_\mathrm{r}$: All four constituent edges are distinct, and there will be four edges between the vertices $\Psi(v_\mathrm{l}, u_\mathrm{l})$ and $\Psi(v_\mathrm{r}, u_\mathrm{r})$ in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$ with labels $a{+}e \;/\; b{+}f$, $c{+}g \;/\; d{+}h$, $g{+}c \;/\; h{+}d$, and $e{+}a \;/\; f{+}b$. Since there are only two distinct labels ($a{+}e \;/\; b{+}f$ and $c{+}g \;/\; d{+}h$ are always distinct for a minimal trellis/encoder, details omitted for brevity), two of the edges can be removed.

- Case $v_\mathrm{l} \neq u_\mathrm{l}$ and $v_\mathrm{r} = u_\mathrm{r}$: In this case there are only two distinct constituent edges to consider, and there will be two edges between the vertices $\Psi(v_\mathrm{l}, u_\mathrm{l})$ and $\Psi(v_\mathrm{r}, u_\mathrm{r})$ in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$ with labels $a{+}g \;/\; b{+}h$ and $g{+}a \;/\; h{+}b$ (or $c{+}e \;/\; d{+}f$ and $e{+}c \;/\; f{+}d$). Since both labels are the same, one of the edges can be removed.

In summary, for the first two cases above, we get a total of $\binom{|V_{\mathrm{x},t}|}{2} \cdot (2 + 1 + 1)$ edges in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$, since there are $\binom{|V_{\mathrm{x},t}|}{2}$ 2-tuples $(v_\mathrm{l}, u_\mathrm{l})$ with $\ell(v_\mathrm{l}) < \ell(u_\mathrm{l})$ and $v_\mathrm{l}, u_\mathrm{l} \in V_{\mathrm{x},t}$, and two possible values for $v_\mathrm{r} = u_\mathrm{r}$ in the second case (the label is either $a{+}g \;/\; b{+}h$ or $c{+}e \;/\; d{+}f$).

- Case $v_\mathrm{l} = u_\mathrm{l}$ and $v_\mathrm{r} \neq u_\mathrm{r}$: In this case there are again only two distinct constituent edges to consider, and there will be two edges between the vertices $\Psi(v_\mathrm{l}, u_\mathrm{l})$ and $\Psi(v_\mathrm{r}, u_\mathrm{r})$ in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$ with labels $a{+}c \;/\; b{+}d$ and $c{+}a \;/\; d{+}b$. Since both labels are the same, one of the edges can be removed.

- Case $v_\mathrm{l} = u_\mathrm{l}$ and $v_\mathrm{r} = u_\mathrm{r}$: In this case there is only one distinct constituent edge to consider, and there will be a single edge between the vertices $\Psi(v_\mathrm{l}, u_\mathrm{l})$ and $\Psi(v_\mathrm{r}, u_\mathrm{r})$ in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$ with label $a{+}a \;/\; b{+}b$ (or $c{+}c \;/\; d{+}d$).

In summary, for the last two cases above, we get a total of $|V_{\mathrm{x},t}| \cdot (1 + 1 + 1)$ edges in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$, since there are $|V_{\mathrm{x},t}|$ 2-tuples $(v_\mathrm{l}, u_\mathrm{l})$ with $v_\mathrm{l} = u_\mathrm{l}$ and $v_\mathrm{l}, u_\mathrm{l} \in V_{\mathrm{x},t}$, and two possible values for $v_\mathrm{r} = u_\mathrm{r}$ in the fourth case (the label is either $a{+}a \;/\; b{+}b$ or $c{+}c \;/\; d{+}d$).

In total, there are $4\binom{|V_{\mathrm{x},t}|}{2} + 3|V_{\mathrm{x},t}| = 2|V_{\mathrm{x},t}|^2 + |V_{\mathrm{x},t}|$ edges in $T^{\mathrm{PC}}_{\mathrm{x},2,t}$, which is the desired result.

12.D Proof of Lemma 12.6

At first we show that $\dot{\mathscr{F}}_{\Pi,\Pi_\mathrm{c}} \subseteq \operatorname{conic}(\dot{\mathscr{Q}}_{\Pi,\Pi_\mathrm{c}})$, so let $\mathbf{y} \in \dot{\mathscr{F}}_{\Pi,\Pi_\mathrm{c}}$. By definition of $\dot{\mathscr{F}}_{\Pi,\Pi_\mathrm{c}}$, this implies the existence of some $\mathbf{f} = (\mathbf{f}^\mathrm{a}, \mathbf{f}^\mathrm{b}, \mathbf{f}^\mathrm{c})$, $\hat{\mathbf{y}} \in \mathbb{R}^{2\lambda K}_{\geq 0}$, and $\tau > 0$ such that $((\mathbf{y}, \hat{\mathbf{y}}), \mathbf{f}) \in \mathscr{F}_{\Pi,\Pi_\mathrm{c}}$ and $\mathbf{f}^\mathrm{x} \in \mathscr{Q}^\tau_\mathrm{x}$ for $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$. We will show that

$$(\tilde{\mathbf{y}}_\tau, \mathbf{f}_\tau) = \left(\frac{1}{\tau}(\mathbf{y}, \hat{\mathbf{y}}), \frac{1}{\tau}\mathbf{f}\right) \in \mathscr{Q}_{\Pi,\Pi_\mathrm{c}},$$

from which the claim follows because then $\mathbf{y} = \tau \mathbf{y}_\tau$ is a positive multiple of an element of $\dot{\mathscr{Q}}_{\Pi,\Pi_\mathrm{c}}$.

Conditions (12.1b) and (12.2b), which hold for $((\mathbf{y}, \hat{\mathbf{y}}), \mathbf{f})$ by definition of $\mathscr{F}_{\Pi,\Pi_\mathrm{c}}$, are invariant to scaling, so they hold for $(\tilde{\mathbf{y}}_\tau, \mathbf{f}_\tau)$ as well. Because $\mathbf{f}^\mathrm{x} \in \mathscr{Q}^\tau_\mathrm{x}$, it also follows that $\mathbf{f}^\mathrm{x}_\tau$ satisfies (12.1a) for all $\mathrm{x} = \mathrm{a}, \mathrm{b}, \mathrm{c}$.

Equation (12.1a) also ensures that the total $\mathbf{f}_\tau$-value in the first segment of each trellis T_{x} equals $\frac{1}{\tau}\tau = 1$, and because of (12.1b) this must hold for all other trellis segments as well. Since $\mathbf{f}_\tau$ is also nonnegative, we can conclude from this that each entry of $\mathbf{f}_\tau$ lies in $[0,1]$. But then also $\tilde{\mathbf{y}}_\tau \in [0,1]^{N+2\lambda K}$ because each $\tilde{y}_j$, $j = 0, \dots, N + 2\lambda K - 1$, is a subset of the total flow through a single segment and thus upper-bounded by 1, which concludes this part of the proof.

Now, we show that $\mathrm{conic}(\dot{\mathscr{Q}}_{\Pi,\Pi_{\mathrm{c}}}) \subseteq \dot{\mathscr{F}}_{\Pi,\Pi_{\mathrm{c}}}$. Let $\mathbf{y} \in \mathrm{conic}(\dot{\mathscr{Q}}_{\Pi,\Pi_{\mathrm{c}}})$. Since $\mathrm{conic}(\dot{\mathscr{Q}}_{\Pi,\Pi_{\mathrm{c}}})$ is the conic hull of the convex set $\dot{\mathscr{Q}}_{\Pi,\Pi_{\mathrm{c}}}$, this implies the existence of some $\tau > 0$ and $\mathbf{y}_{\mathscr{Q}} \in \dot{\mathscr{Q}}_{\Pi,\Pi_{\mathrm{c}}}$ such that $\mathbf{y} = \tau \cdot \mathbf{y}_{\mathscr{Q}}$. To $\mathbf{y}_{\mathscr{Q}}$ then there must exist $\hat{\mathbf{y}}_{\mathscr{Q}}$ and $\mathbf{f}_{\mathscr{Q}}$ such that $((\mathbf{y}_{\mathscr{Q}}, \hat{\mathbf{y}}_{\mathscr{Q}}), \mathbf{f}_{\mathscr{Q}}) \in \mathscr{Q}_{\Pi,\Pi_{\mathrm{c}}}$, from which immediately it follows that $((\mathbf{y} = \tau \cdot \mathbf{y}_{\mathscr{Q}}, \tau\hat{\mathbf{y}}_{\mathscr{Q}}), \tau\mathbf{f}_{\mathscr{Q}}) \in \mathscr{F}_{\Pi,\Pi_{\mathrm{c}}}$, and thus $\mathbf{y} \in \dot{\mathscr{F}}_{\Pi,\Pi_{\mathrm{c}}}$.

Acknowledgment

The authors wish to thank the anonymous reviewers for their valuable comments and suggestions that helped improve the presentation of the paper.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima. "Near shannon limit error-correcting coding and decoding: turbo-codes". In: *IEEE International Conference on Communications.* May 1993, pp. 1064–1070. DOI: 10.1109/ICC.1993.397441.

[2] C. Berrou et al. "Improving the distance properties of turbo codes using a third component code: 3D turbo codes". *IEEE Transactions on Communications* 57.9 (Sept. 2009), pp. 2505–2509. DOI: 10.1109/TCOMM.2009.09.070521.

[3] E. Rosnes and A. Graell i Amat. "Performance analysis of 3-d turbo codes". *IEEE Transactions on Information Theory* 57.6 (June 2011), pp. 3707–3720. ISSN: 0018-9448. DOI: 10.1109/TIT.2011.2133610.

[4] A. Graell i Amat and E. Rosnes. "Stopping set analysis of 3-dimensional turbo code ensembles". In: *Proceedings of IEEE International Symposium on Information Theory.* Austin, TX, June 2010, pp. 2013–2017. DOI: 10.1109/ISIT.2010.5513362.

[5] C. Koller et al. "Analysis and design of tuned turbo codes". *IEEE Transactions on Information Theory* 58.7 (July 2012), pp. 4796–4813.

[6] J. Feldman, D. R. Karger, and M. Wainwright. "Linear programming-based decoding of turbo-like codes and its relation to iterative approaches". In: *Proceedings of the 40th Annual Allerton Conference on Communication, Control and Computing*. Monticello, IL, 2002, pp. 467–477.

[7] J. Feldman. "Decoding error-correcting codes via linear programming". PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 2003.

[8] J. Feldman, M. J. Wainwright, and D. R. Karger. “Using linear programming to decode binary linear codes”. *IEEE Transactions on Information Theory* 51.3 (Mar. 2005), pp. 954–972. DOI: 10.1109/TIT.2004.842696. URL: www.eecs.berkeley.edu/~wainwrig/Papers/FelWaiKar05.pdf.

[9] P. O. Vontobel and R. Koetter. *Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes.* 2005. arXiv: cs/0512078 `[cs.IT]`.

[10] E. Rosnes. “On the connection between finite graph covers, pseudo-codewords, and linear programming decoding of turbo codes”. In: *Proceedings of the 4th International Symposium on Turbo Codes & Related Topics.* Munich, Germany, Apr. 2006, pp. 1–6.

[11] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. “Factor graphs and the sum-product algorithm”. *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 498–519. DOI: 10.1109/18.910572. URL: www.comm.utoronto.ca/frank/papers/KFL01.pdf.

[12] M. Chertkov and M. G. Stepanov. “Polytope of correct (linear programming) decoding and low-weight pseudo-codewords”. In: *Proceedings of IEEE International Symposium on Information Theory.* St. Petersburg, Russia, July 2011, pp. 1648–1652. DOI: 10.1109/ISIT.2011.6033824.

[13] M. Chertkov and M. G. Stepanov. “An efficient pseudocodeword search algorithm for linear programming decoding of LDPC codes”. *IEEE Transactions on Information Theory* 54.4 (Apr. 2008), pp. 1514–1520. ISSN: 0018-9448. DOI: 10.1109/TIT.2008.917682.

[14] S. Abu-Surra, D. Divsalar, and W. E. Ryan. “Enumerators for protograph-based ensembles of LDPC and generalized LDPC codes”. *IEEE Transactions on Information Theory* 57.2 (Feb. 2011), pp. 858–886. DOI: 10.1109/TIT.2010.2094819.

[15] D. Divsalar and L. Dolecek. “Graph cover ensembles of non-binary protograph LDPC codes”. In: *Proceedings of IEEE International Symposium on Information Theory.* Cambridge, MA, July 2012, pp. 2526–2530. DOI: 10.1109/ISIT.2012.6283972.

[16] D. Divsalar and L. Dolecek. “Ensemble analysis of pseudocodewords of protograph-based non-binary LDPC codes”. In: *Proceedings of the IEEE Information Theory Workshop.* Paraty, Brazil, Oct. 2011, pp. 340–344. DOI: 10.1109/ITW.2011.6089475.

[17] N. Boston. “A multivariate weight enumerator for tail-biting trellis pseudocodewords”. In: *Proceedings of the Workshop on Algebra, Combinatorics and Dynamics.* Belfast, Northern Ireland, Aug. 2009.

[18] D. Conti and N. Boston. “Matrix representations of trellises and enumerating trellis pseudocodewords”. In: *Proceedings of the 49th Annual Allerton Conference on Communication, Control and Computing.* Monticello, IL, Sept. 2011, pp. 1438–1445.

[19] P. O. Vontobel. “Counting in graph covers: a combinatorial characterization of the bethe entropy function”. *IEEE Transactions on Information Theory* 59.9 (Sept. 2013), pp. 6018–6048. DOI: 10.1109/TIT.2013.2264715.

[20] *Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding (Release 8).* Technical Specification. 3$^{\text{rd}}$ Generation Partnership Project; Group Radio Access Network, Dec. 2008.

[21] G. D. Forney Jr. “Codes on graphs: normal realizations”. *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 529–548. DOI: 10.1109/18.910573.

[22] A. Tanatmis et al. “Valid inequalities for binary linear codes”. In: *Proceedings of IEEE International Symposium on Information Theory*. Seoul, Korea, June 2009, pp. 2216–2220. DOI: 10.1109/ISIT.2009.5205846.

[23] E. Rosnes and Ø. Ytrehus. “Turbo decoding on the binary erasure channel: finite-length analysis and turbo stopping sets”. *IEEE Transactions on Information Theory* 53.11 (Nov. 2007), pp. 4059–4075. DOI: 10.1109/TIT.2007.907496.

[24] M. F. Flanagan. “Exposing pseudoweight layers in regular LDPC code ensembles”. In: *Proceedings of the IEEE Information Theory Workshop*. Taormina, Italy, Oct. 2009, pp. 60–64. DOI: 10.1109/ITW.2009.5351212. arXiv: 0908.1298 `[cs.IT]`.

[25] H. D. Pfister and P. H. Siegel. “The serial concatenation of rate-1 codes through uniform random interleavers”. *IEEE Transactions on Information Theory* 49.6 (June 2003), pp. 1425–1438. DOI: 10.1109/TIT.2003.811907.

[26] R. Koetter and P. O. Vontobel. “Graph covers and iterative decoding of finite-length codes”. In: *Proceedings of the 3rd International Symposium on Turbo Codes & Related Topics*. Brest, France, Sept. 2003, pp. 75–82.

[27] J. Sun and O. Y. Takeshita. “Interleavers for turbo codes using permutation polynomials over integer rings”. *IEEE Transactions on Information Theory* 51.1 (Jan. 2005), pp. 101–119. DOI: 10.1109/TIT.2004.839478.

[28] O. Y. Takeshita. “On maximum contention-free interleavers and permutation polynomials over integer rings”. *IEEE Transactions on Information Theory* 52.3 (Mar. 2006), pp. 1249–1253. DOI: 10.1109/TIT.2005.864450.

[29] S. Crozier, P. Guinand, and A. Hunt. “Computing the minimum distance of turbo-codes using iterative decoding techniques”. In: *Proceedings of the 22nd Biennial Symposium on Communications*. Kingston, ON, Canada, May–June 2004, pp. 306–308.

[30] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization, John Wiley & Sons, 1988.

[31] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[32] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice-Hall, 1993.

Part III

Closing

Chapter 13

Conclusions and Future Work

In this thesis, we have shown that the use of mathematical optimization is able to contribute substantially to the area of coding theory in such different aspects as fundamental theory, algorithm development, and even hardware implementation. To a certain extent, this is due to the formulation of the ML decoding problem as an integer linear program with a useful LP relaxation by Feldman in 2003 [4]. But even more generally, all sorts of different subdisciplines of mathematical optimization find their application to the decoding problem. In Paper IV, for instance, the graphical trellis structure of turbo codes was used in a very fast combinatorial solution algorithm for the subproblems, while ideas from computational geometry and multicriteria optimization provide the "glue" to find the optimal LP solution from a finite number of those subproblems. More examples for such connections exist that did not play a central role in this thesis, e.g. the interpretation of a linear code as a *matroid* that was mentioned in Paper I (see also [28]) and the recent advances in applying *non-linear* optimization methods to the LP decoding problem [38].

The potential of using mathematical optimization approaches for decoding are out of question. The algorithm developed in Paper VI is, to our knowledge, the fastet available ML decoder for general linear codes, and the outstanding performance of our turbo LP decoder (Paper IV) suggests that, when it is used within a branch-and-cut procedure similar to Paper VI, the same result will appear for the case of turbo and turbo-like codes. Paper III has shown that IP formulations can even be used, in some cases, to simulate the error-correction performance of heuristic decoding methods faster than by running these heuristics themselves. Finally, Paper VII has illustrated that the theory of linear programming decoding, together with the related concepts of LP pseudocodewords and graph covers, is able to evaluate individual codes as well as complete ensembles with respect to their pseudoweight spectrum that determines the decoding behavior of both the LP and belief-propagation decoders.

However, the "transfer of ideas" that is present in the contributions of this thesis is not simply one-directional, going from optimization to coding problems. The findings in Paper IV, while being driven by the specific application of LP decoding of turbo codes, are to a large extent independent of the concrete problem setup and thus applicable to a whole class of combinatorial optimization problems with additional complicating equality constraints. The question how this approach relates and compares to other methods is of great interest for future research—in fact, after publication of Paper IV we have discovered a close relationship to the Dantzig-Wolfe reformulation of the problem (see [39]) that has to be further examined.

Besides that, however, there are several other natural directions in which to continue the studies that were begun in this thesis. The application of the branch-and-cut ML decoder to the case of turbo codes was already mentioned above. Another question includes if and how non-linear LP decoders, in particular the ADMM method [38], can be employed as a high-performance subroutine within the branch-and-cut procedure. The work in Paper V entails an entire new discipline on hardware implementation of optimization-based decoding algorithms, and finally it is an interesting open question how to generalize the algorithms we developed to the cases of non-binary and / or higher-order modulation that were examined in Paper II.

Bibliography for Parts I and III

[1] D. J. C. MacKay. "Good error-correcting codes based on very sparse matrices". *IEEE Transactions on Information Theory* 45.2 (Mar. 1999), pp. 399–431. DOI: 10.1109/18.748992.

[2] C. Berrou and A. Glavieux. "Near optimum error correcting coding and decoding: turbo-codes". *IEEE Transactions on Communications* 44.10 (Oct. 1996), pp. 1261–1271. ISSN: 0090-6778. DOI: 10.1109/26.539767.

[3] J. Feldman, D. R. Karger, and M. Wainwright. "Linear programming-based decoding of turbo-like codes and its relation to iterative approaches". In: *Proceedings of the 40th Annual Allerton Conference on Communication, Control and Computing*. Monticello, IL, 2002, pp. 467–477.

[4] J. Feldman. "Decoding error-correcting codes via linear programming". PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 2003.

[5] J. Feldman, M. J. Wainwright, and D. R. Karger. "Using linear programming to decode binary linear codes". *IEEE Transactions on Information Theory* 51.3 (Mar. 2005), pp. 954–972. DOI: 10.1109/TIT.2004.842696. URL: www.eecs.berkeley.edu/~wainwrig/Papers/FelWaiKar05.pdf.

[6] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Aug. 1963.

[7] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.

[8] U. Faigle, W. Kern, and G. Still. *Algorithmic Principles of Mathematical Programming*. Vol. 24. Kluwer Academic Publishers, 2010.

[9] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization, John Wiley & Sons, 1988.

[10] V. Klee and G. J. Minty. "How good is the simplex algorithm?" In: *Proc. Inequalities III*. Los Angeles, CA: Academic Press, Sept. 1972, pp. 159–175.

[11] C. E. Shannon. "A mathematical theory of communication". *The Bell System Technical Journal* 27 (July 1948), pp. 379–423, 623–656.

[12] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. URL: http://www.inference.phy.cam.ac.uk/itprnn/book.html.

[13] R. G. Gallager. *Information Theory and Reliable Communication*. John Wiley & Sons, 1968.

[14] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. Vol. 16. North-Holland, 1977.

[15] T. J. Richardson and R. L. Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008. ISBN: 978-0-521-85229-6.

[16] D. Costello Jr. and G. D. Forney Jr. “Channel coding: the road to channel capacity”. *Proceedings of the IEEE* 95.6 (June 2007), pp. 1150–1177. DOI: 10.1109/JPROC.2007.895188.

[17] E. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. “On the inherent intractability of certain coding problems”. *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 954–972. DOI: 10.1109/TIT.1978.1055873.

[18] A. Vardy. “The intractability of computing the minimum distance of a code”. *IEEE Transactions on Information Theory* 43.6 (Nov. 1997), pp. 1757–1766. DOI: 10.1109/18.641542.

[19] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. “Factor graphs and the sum-product algorithm”. *IEEE Transactions on Information Theory* 47.2 (Feb. 2001), pp. 498–519. DOI: 10.1109/18.910572. URL: www.comm.utoronto.ca/frank/papers/KFL01.pdf.

[20] P. O. Vontobel and R. Koetter. *Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes.* 2005. arXiv: cs/0512078 `[cs.IT]`.

[21] M. Flanagan et al. “Linear-programming decoding of nonbinary linear codes”. *IEEE Transactions on Information Theory* 55.9 (Sept. 2009), pp. 4134–4154. ISSN: 0018-9448. DOI: 10.1109/TIT.2009.2025571. arXiv: 0804.4384 `[cs.IT]`.

[22] M. Breitbach et al. “Soft-decision decoding of linear block codes as optimization problem”. *European Transactions on Telecommunications* 9 (1998), pp. 289–293.

[23] A. Tanatmis et al. “A separation algorithm for improved LP-decoding of linear block codes”. *IEEE Transactions on Information Theory* 56.7 (July 2010), pp. 3277–3289. ISSN: 0018-9448. DOI: 10.1109/TIT.2010.2048489.

[24] M. Punekar et al. “Calculating the minimum distance of linear block codes via integer programming”. In: *Proceedings of the International Symposium on Turbo Codes and Iterative Information Processing.* Brest, France, Sept. 2010, pp. 329–333. DOI: 10.1109/ISTC.2010.5613894.

[25] A. B. Keha and T. M. Duman. “Minimum distance computation of LDPC codes using a branch and cut algorithm”. *IEEE Transactions on Communications* 58.4 (Apr. 2010), pp. 1072–1079. DOI: 10.1109/TCOMM.2010.04.090164.

[26] A. Tanatmis et al. “Valid inequalities for binary linear codes”. In: *Proceedings of IEEE International Symposium on Information Theory.* Seoul, Korea, June 2009, pp. 2216–2220. DOI: 10.1109/ISIT.2009.5205846.

[27] A. Tanatmis et al. “Numerical comparison of IP formulations as ML decoders”. In: *IEEE International Conference on Communications.* Cape Town, South Africa, May 2010, pp. 1–5. DOI: 10.1109/ICC.2010.5502303.

[28] N. Kashyap. “A decomposition theory for binary linear codes”. *IEEE Transactions on Information Theory* 54.7 (July 2008), pp. 3035–3058. DOI: 10.1109/TIT.2008.924700. URL: http://www.ece.iisc.ernet.in/~nkashyap/Papers/code_decomp_final.pdf.

[29] R. G. Jeroslow. “On defining sets of vertices of the hypercube by linear inequalities”. *Discrete Mathematics* 11.2 (1975), pp. 119–124. ISSN: 0012-365X. DOI: 10.1016/0012-365X(75)90003-5.

[30] R. G. Gallager. "Low-density parity-check codes". PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, Sept. 1960.

[31] M. H. Taghavi and P. H. Siegel. "Adaptive methods for linear programming decoding". *IEEE Transactions on Information Theory* 54.12 (Dec. 2008), pp. 5396–5410. DOI: 10.1109/TIT.2008.2006384. arXiv: cs/0703123 `[cs.IT]`.

[32] M. H. Taghavi, A. Shokrollahi, and P. H. Siegel. "Efficient implementation of linear programming decoding". *IEEE Transactions on Information Theory* 57.9 (Sept. 2011), pp. 5960–5982. DOI: 10.1109/TIT.2011.2161920. arXiv: 0902.0657 `[cs.IT]`.

[33] X. Zhang and P. H. Siegel. "Adaptive cut generation algorithm for improved linear programming decoding of binary linear codes". *IEEE Transactions on Information Theory* 58.10 (Oct. 2012), pp. 6581–6594. DOI: 10.1109/TIT.2012.2204955. arXiv: 1105.0703 `[cs.IT]`.

[34] G. D. Forney Jr. et al. "On the effective weights of pseudocodewords for codes defined on graphs with cycles". In: *Codes, systems, and graphical models*. Ed. by B. Marcus and J. Rosenthal. Vol. 123. The IMA Volumes in Mathematics and its Applications. Springer Verlag, New York, Inc., 2001, pp. 101–112.

[35] R. Koetter and P. O. Vontobel. "Graph covers and iterative decoding of finite-length codes". In: *Proceedings of the 3rd International Symposium on Turbo Codes & Related Topics*. Brest, France, Sept. 2003, pp. 75–82.

[36] M. Chertkov and M. G. Stepanov. "Polytope of correct (linear programming) decoding and low-weight pseudo-codewords". In: *Proceedings of IEEE International Symposium on Information Theory*. St. Petersburg, Russia, July 2011, pp. 1648–1652. DOI: 10.1109/ISIT.2011.6033824.

[37] A. I. Ali, J. Kennington, and B. Shetty. "The equal flow problem". *European Journal of Operational Research* 36.1 (1988), pp. 107–115. ISSN: 0377-2217. DOI: 10.1016/0377-2217(88)90012-4.

[38] S. Barman et al. "Decomposition methods for large scale lp decoding". *IEEE Transactions on Information Theory* 59.12 (Dec. 2013), pp. 7870–7886. DOI: 10.1109/TIT.2013.2281372.

[39] G. B. Dantzig and P. Wolfe. "Decomposition principle for linear programs". *Operations Research* 8.1 (1960), pp. 101–111. ISSN: 0030364X. URL: http://www.jstor.org/stable/167547.

Lebenslauf

Michael Helmling

geboren am 19. Februar 1986 in Kaiserslautern

03/2005	**Allgemeine Hochschulreife (Abitur)**, *Albert-Schweitzer-Gymnasium*, Kaiserslautern.
05/2005–01/2006	**Zivildienst**, *Westpfalz-Klinikum*, Kaiserslautern.
2005	**Früheinstieg ins Physikstudium**, *Technische Universität Kaiserslautern*. Teilnahme am Fernstudienprogramm.
04/2006–01/2011	**Mathematikstudium mit Anwendungsfach Informatik**, *TU Kaiserslautern*
10/2007–09/2008	**Parallelstudium Informatik**, *TU Kaiserslautern*. Erfolgreiche Ablegung von vier Prüfungen (ohne Abschluss).
01/2011	**Diplom in Mathematik mit Anwendungsfach Informatik**, *TU Kaiserslautern*. Titel der Diplomarbeit: *Band Matrix Constrained Optimization Problems with Applications to Coding Theory*.
03/2011–02/2013	**Promotionsstudium in Mathematik**, *TU Kaiserslautern*. Mit einem Stipendium der Landesforschungsinitiative *(CM)² – Center for Mathematical and Computational Modelling*.
seit 03/2013	**Wissenschaftlicher Mitarbeiter**, *Mathematisches Institut der Universität Koblenz-Landau, Campus Koblenz*. Dabei Fortsetzung der in Kaiserslautern begonnenen Promotion.