Mathematical Programming Approaches for Decoding of Binary Linear Codes

Vom Fachbereich Mathematik
der Technischen Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte Dissertation

vorgelegt von

Akin Tanatmis

Gutacher: Jun.-Prof. Dr. Stefan Ruzika
Dr. Pascal O. Vontobel

Datum der Disputation: 21. Januar 2011

D 386

Contents

Chapter 1

Introduction

Wireless communication has fundamentally changed our information based society in the last decade. Error-correcting codes are used in wireless communication systems for correcting transmission errors which occur due to noise in the transmission medium. In systems using error-correcting codes, at the transmitter end, an assembly of original and redundant information is constructed in form of codewords and sent through the transmission medium. At the receiver end, decoding of codewords is performed. The original information is regenerated with the help of redundant information. Maximum likelihood decoding (MLD) is the optimal decoding approach for any channel code and it is known to be NP-hard.

In practice, channel codes, for example low density parity-check (LDPC) codes or turbo codes, are generally decoded by heuristic approaches called iterative message passing decoding (IMPD), subsuming sum-product algorithm decoding (SPD) (see [3], [47]) and min-sum algorithm decoding (MSD) (see [74]). 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 certificate property, i.e., if the output is a codeword, it is not necessarily the maximum likelihood (ML) codeword. Furthermore, performance of IMPD is poor for arbitrary linear block codes with a dense parity-check matrix.

Mathematical programming is a branch of applied mathematics and has recently been used to derive alternative decoding approaches. Introducing integer programming (IP) formulations of the MLD problem for binary linear codes, Breitbach et al. [9] and Feldman et al. [27] have done pioneering work in this field. A standard approach to deal with difficult IP problems is to relax the integrality constraints and study the relaxation. Linear programming decoding (LPD) was introduced in [27]. LPD offers some advantages and thus it

has become an alternative decoding technique. First, this approach provides analytical statements on convergence, complexity, and correctness of decoding algorithms. Second, LPD is not limited to sparse parity-check matrices. Third, performance improvements can be achieved by enhancing LPD with integer programming techniques, e.g., branch and bound algorithms, cut generation algorithms. Since the work of Feldman et al. [27], LPD has been intensively studied in a variety of articles (see Chapter 5) especially dealing with low density parity-check (LDPC) codes.

The content of this thesis has a very strong interdisciplinary background. MLD of binary linear codes is a real world problem from the field of electrical engineering which can be solved by mathematical programming methods. Our solution approach combines concepts from coding theory as well as mathematical optimization.

The mathematical modeling cycle in general is depicted in Figure 1.1. For a practical problem, first, a mathematical model is built. Second, solution approaches are developed. At this step, it is explored if the results found can be generalized to other optimization problems. Mathematical innovation and discovery is motivated in this way. The cycle is completed with the evaluation and interpretation of results. The mathematical modeling cycle can be repeated until satisfactory results are acquired.

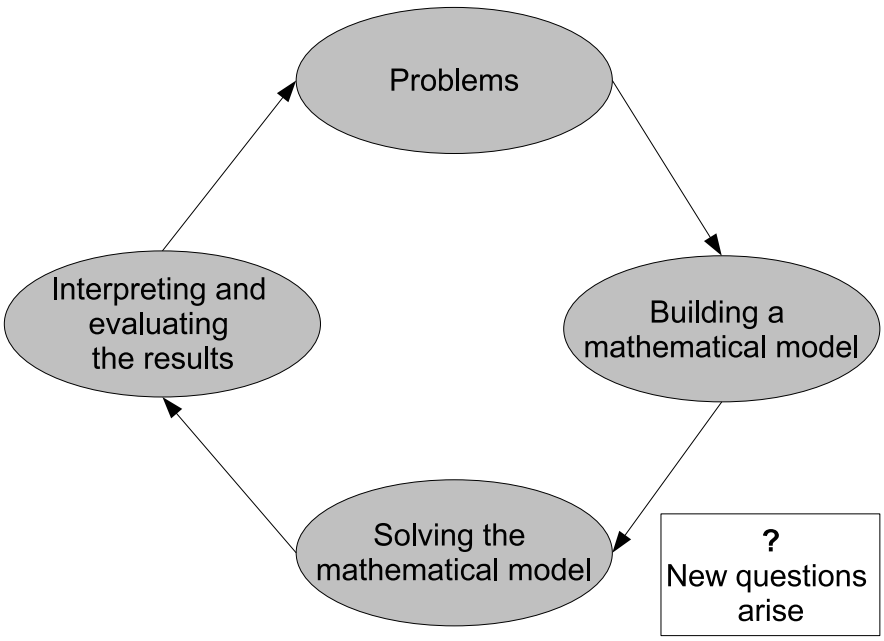


Figure 1.1: Mathematical modeling cycle

1.1 Contributions of this thesis

Based on the mathematical modeling cycle, in this thesis, we aim at finding appropriate integer programming models and associated solution approaches for the MLD problem of several binary linear code classes. Studying the polyhedral properties of the problem is a crucial step of tackling IP problems. Many interesting properties of the codeword polytope, i.e., the convex hull of the codewords of a binary linear code, are known from matroid theory. In particular, polyhedral properties of the cycle polytope, i.e., the convex hull of the incidence vectors of the cycles of a binary matroid, directly applies to codeword polytope of a binary linear code. For example, in this thesis we show the relationship between the codeword polytope of simplex codes and the cycle polytope of the complete binary matroids. We also present some results on decomposition of Hamming codes and simplex codes.

For binary linear codes with short and medium block length, MLD can be achieved by solving the associated IP problem with a general purpose IP solver. IP also offers algorithms for computing the minimum distance of a binary linear code. We present several IP formulations and computationally compare them on various binary linear codes for MLD and minimum distance computation. Moreover, we explore some strategies to facilitate the computation of minimum distance and ML curves.

We review and categorize decoding methods based on mathematical programming approaches for binary linear codes over any binary input memoryless channel. Concepts from mathematical programming used in the context of decoding include linear, integer, and non-linear programming, network flows, notions of duality as well as matroid and polyhedral theory.

As a mathematical programming decoding approach, we propose a new separation algorithm to improve the error-correcting performance of LPD for binary linear block codes. We use an IP formulation with indicator variables that help in detecting the violated parity checks. Gomory cuts from the IP are derived and used in the separation algorithm. An efficient method of finding cuts induced by redundant parity checks (RPCs) is also proposed. Under certain circumstances we can guarantee that these RPC cuts are valid and cut off the fractional optimal solutions of LPD. It is demonstrated on several binary linear codes that our separation algorithm performs significantly better than LPD and SPD in terms of error-correcting performance. Using the minimum distance of a code, a new class of valid inequalities is also introduced.

We focus on developing combinatorial algorithms which use the special structure of the code at hand. Combinatorial algorithms may perform better than algorithms based on solving an LP relaxation in terms of computational complexity and error-correcting performance. Based on a novel IP formulation we propose a two-step algorithm for decoding of LTE turbo codes. Under certain assumptions it can be guaranteed that our two-step algorithm outputs the

ML codeword. We also show that the proposed algorithm outperforms LPD on LTE turbo codes with short block length. For the (132,40) LTE turbo code our algorithm has better error-correcting performance than standard iterative turbo decoding.

The mathematical model and the solution approach developed for decoding of LTE turbo codes initiated our study on equality constraint shortest path problems (ECSPPs). In ECSPP, a shortest path of a directed acyclic graph which satisfy some equality side constraints is explored. ECSPP is NP-hard and it can be considered as a special case of knapsack or subset-sum problems. We develop multiple objective optimization based solution algorithms to find an optimal solution of the LP relaxation of ECSPP with one or two equality side constraints.

1.2 Overview and related publications

Preliminaries on coding theory and polyhedral theory are briefly summarized in Chapter 2.

Chapter 3 is a collobaration with Sebastian Heupel. Our own results and proofs presented in this chapter can partly be found in the Master's thesis: Cycle polytopes and their application in coding theory, S. Heupel, July 2009.

Chapter 4 has partly appeared in: Numerical Comparison of IP Formulations as ML decoders, A. Tanatmis, S. Ruzika, M. Punekar, F. Kienle., In Proceedings of IEEE International Conference on Communications (ICC), May, 2010, Cape Town, South Africa.

Chapter 5 has partly appeared in: Optimization based methods in coding theory, A. Tanatmis and S. Ruzika, submitted to IEEE Transactions on Information Theory.

Chapter 6 has partly appeared in: A Separation Algorithm for Improved LP-Decoding of Linear Block Codes, A. Tanatmis, S. Ruzika, H. W. Hamacher , M. Punekar, F. Kienle and N. Wehn. IEEE Transactions on Information Theory, Vol. 56, Nr. 7 , pages 3277 - 3289, July, 2010 and Valid Inequalities for Binary Linear Codes, A. Tanatmis, S. Ruzika, H. W. Hamacher , M. Punekar, F. Kienle, N. Wehn. In Proceedings of IEEE International Symposium on Information Theory (ISIT), pages 2216 - 2220, July, 2009, Seoul, Korea.

Chapter 7 has partly appeared in: A Lagrangian relaxation based decoding algorithm for LTE turbo codes, A. Tanatmis, S. Ruzika, F. Kienle. In Proceedings of 6th International Symposium on Turbo Codes and Iterative Information Processing, September, 2010, Brest, France.

Chapter 8 is a collaboration with Michael Helmling.

The author of this thesis is the main author of all the papers listed above. This research has been carried on between 2006 - 2010 in the optimization research group, department of mathematics, University of Kaiserslautern.

1.3 Acknowledgements

I gratefully acknowledge financial support by the Center of Mathematical and Computational Modeling of the University of Kaiserslautern.

Foremost, I am grateful to Prof. Dr. Horst W. Hamacher and Prof. Dr. Sven O. Krumke for giving me the opportunity of working in their group. I have learned a lot from their lectures, seminars, and talks. Also I have benefited from their experience, advice, suggestions, and great working atmosphere they created. I would like to thank Prof. Dr. Horst W. Hamacher also for advising me in my career decisions.

I am deeply grateful to Jun. -Prof. Dr. Stefan Ruzika for co-authoring, giving me so many valuable ideas, and making constructive criticism on my work. His door was always open to me for any kind of problem or question I had.

The discussions with my colleagues have improved this work. Special thanks go to my colleagues whom I worked with in the optimization group between the years 2006 - 2010.

I would like to thank Prof. Dr. Ing. Norbert Wehn for the initiation of this work, his support and contributions. Error control coding was a new field for me and I have learned a lot from Dr. Ing. Frank Kienle on codes and decoding algorithms. I want to thank him for his patience and the time he invested in answering my questions. I would also like to thank the members of the microelectronic systems design research group, especially Mayur Punekar for many useful discussions.

I am grateful to Dr. Pascal O. Vontobel from HP Labs in Palo Alto for reading parts of this thesis and giving me very detailed and valuable comments and suggestions. Conversations with him were very informative for me. I also want to thank him for accepting to write the second assessment of this thesis.

For the generous travel support I thank the $(CM)^2$-Nachwuchsring. With the help of the $(CM)^2$-Nachwuchsring I had the chance to meet very important researchers from all over the world and present my research to them.

I want to thank Dr. Rüdiger Stephan for visiting me in Kaiserslautern and sharing his expertise on cardinality constrained combinatorial optimization. This initiated our research of the results which are collected in Chapter 3. I would also like to thank Dr. Eirik Rosnes for our discussions on LTE turbo codes.

I am thankful to Sebastian Heupel for our pleasant collaboration on the relationship between matroid theory and coding theory. I also want to thank Michael Helmling for all the effort he put in programming several algorithms presented in this thesis.

Chapter 2

Preliminaries

2.1 Introduction

In this chapter, notation is fixed and well-known but relevant results from coding theory and polyhedral theory are recalled. With coding theory we refer to the theory of error-control coding. Elementary results on coding theory, binary linear block codes, maximum likelihood decoding and performance measures of codes are covered in Section 2.2. A comprehensive review can be found in [50]. The code families considered in the subsequent chapters are described in Section 2.3. An important aspect of this thesis is the analysis of integer programming (IP) and linear programming (LP) based decoding approaches for binary linear codes. Thus, basics of polyhedral theory are repeated in Section 2.4.

2.2 Coding theory

Error-control coding is concerned with the reliable transmission of information over an unreliable channel. Telephone lines, video broadcasting, etc. are such channels subject to disturbance. For example, on a telephone line thermal noise or interference by others may cause the disturbance.

A simplified abstraction of a communication system is depicted in Figure 2.1. Symbols are used to transfer information. They may belong to any alphabet but in this thesis we consider bits $\{0, 1\}$. An information word u is a k-tuple of binary variables, i.e., $u \in \{0, 1\}^k$. To be robust against the noise in the channel, the sender encodes the information word by adding redundant bits according to an encoding function $E : \{0, 1\}^k \to \{0, 1\}^n$ which is usually injective. The positive integer $n > k$ is called the block length. A codeword x is given by $x = E(u)$ and the set of all codewords $C = \{x : x = E(u),\ u \in \{0, 1\}^k\}$ forms a code C. The cardinality of C is determined by k, $|C| = 2^k$. The rate of a

code is given by $R = \frac{k}{n}$. Codes with high rates are desirable since they contain less redundancy.

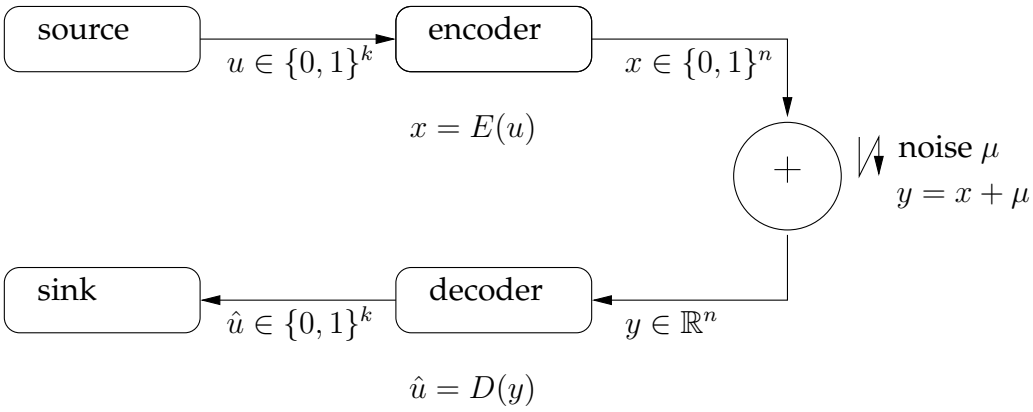


Figure 2.1: Encoding and decoding information.

A channel is a communication medium where the transmission takes place. It is modeled by an input alphabet, output alphabet and the probability of transition. The channels considered in this thesis are the binary symmetric channel (BSC) described in Example 2.1 and the binary input additive white Gaussian noise channel (BIAWGNC) described in Example 2.2. These are binary input memoryless symmetric channels. In a binary input channel only symbols from the alphabet $\{0, 1\}$ are transmitted. The memoryless property implies that bits are affected by the channel independently. Finally the word symmetric is used in the sense that the error probability is not dependent on input, i.e, $P(0|1) = P(1|0)$.

Example 2.1. *In the BSC, due to the noise, bits from the channel input are flipped with a crossover probability p. The channel output is a corrupt word $y \in \{0,1\}^n$.*

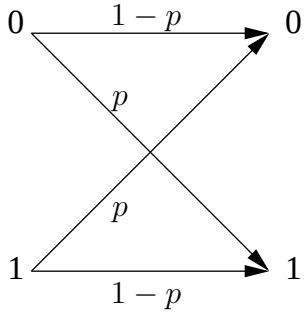


Figure 2.2: Binary symmetric channel.

Example 2.2. *In the BIAWGNC, bits* 0 *and* 1 *are first mapped to* $+1$ *and* -1*, respectively, i.e.,* $x \in \{-1, +1\}^n$*. During the transmission, a Gaussian random variable* $\mu_j \in \mathcal{N}(0, \sigma^2)$*,* $j \in \{1, \ldots, n\}$*, is added independently to each code bit* x_j*, such that* $y \in \mathbb{R}^n$*,* $y_j = x_j + \mu_j$*, is returned as channel output.*

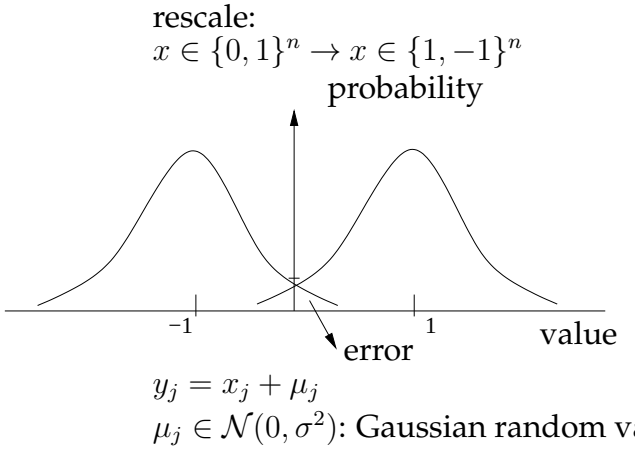


Figure 2.3: Binary input additive white Gaussian noise channel.

The conditional probability density function is given by

$$P(y_j|x_j) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_j - x_j)^2}{2\sigma^2}}.$$

Given a channel output (received word) $y \in \{0,1\}^n$ in the BSC or $y \in \mathbb{R}^n$ in the BIAWGNC, the decoder function $D : \mathbb{R}^n \to \{0,1\}^k$ makes an estimate of the codeword, i.e., $\hat{x} \in \{0,1\}^n$. The estimated information word is given by $\hat{u} = D(y)$. Decoding is successful if $u = \hat{u}$. If $u \neq \hat{u}$, then a decoding error occurs.

The goal in coding theory is to design an encoder/decoder pair such that [50]:

(1) information can be transmitted in a noisy environment as fast as possible,

(2) information can be reliably reproduced at the output of the channel decoder,

(3) cost of implementing the encoder and the decoder falls within acceptable limits.

The difficult part of decoding is the estimation of the transmitted codeword from the received word. The translation of a codeword into an information word is straightforward since there is a one-to-one correspondence between information words and codewords.

2.2.1 Binary linear block codes

In this thesis, we concentrate solely on binary linear block codes.

Definition 2.3. *An (n, k) 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 field $GF(2)$.*

A code $C \subseteq \{0,1\}^n$ can be defined by k basis vectors of length n which are represented by a $(k \times n)$ matrix G called the generator matrix. An example (4×7) generator matrix is given below:

Example 2.4.

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

Given a generator matrix G, it holds that

$$C = \{x \in \{0,1\}^n : x \equiv uG \bmod 2, u \in \{0,1\}^k\}.$$

A binary linear block code can be put into systematic (standard) form.

Definition 2.5. *The generator matrix G is in systematic form if it is written as $G = [I_k|A]$ where I_k is the $(k \times k)$ identity matrix and A is a $(k \times (n-k))$ zero-one matrix.*

The G matrix in systematic form indicates that the first k bits of x are the information bits and that the remaining $n - k$ bits are the parity-check bits which are linear sums (mod 2) of the information bits.

Example 2.6. *Let $u = (u_1, u_2, u_3, u_4)$ and G be the example matrix given in Example 2.4, then the bits of the codeword x are found by*

$$x \equiv uG \ mod\ 2 \equiv (u_1, u_2, u_3, u_4) \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \ mod\ 2,$$

$$x_1 \equiv u_1 \ mod\ 2,\ x_2 \equiv u_2 \ mod\ 2,\ x_3 \equiv u_3 \ mod\ 2,\ x_4 \equiv u_4 \ mod\ 2,$$
$$x_5 \equiv u_1 + u_2 + u_4 \ mod\ 2,$$
$$x_6 \equiv u_1 + u_3 + u_4 \ mod\ 2,$$
$$x_7 \equiv u_2 + u_3 + u_4 \ mod\ 2,$$

Definition 2.7. *Every linear subspace of dimension k has an orthogonal subspace of dimension $n-k$. The orthogonal subspace of binary linear code C, denoted by $C^{\perp}$, is defined as*

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

The subspace $C^{\perp}$ can also be interpreted as a binary linear code of dimension $n-k$ which is referred to as the dual code of C. A set of basis vectors of $C^{\perp}$ constitutes a parity-check matrix $H \in \{0,1\}^{((n-k)\times n)}$. Since every codeword $x \in C$ is orthogonal to each row of H, it holds that $Hx \equiv 0 \bmod 2$. In fact, C is the null space of H and a vector $x \in \{0,1\}^n$ is in C if and only if $Hx \equiv 0 \bmod 2$. If the generator matrix G is in the systematic form, i.e., $G = [I_k|A]$ then an associated parity-check matrix H may be written as $H = \left[A^T|I_{n-k}\right]$.

Example 2.8. *A possible parity-check matrix for the code in Example 2.6 is*

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

A parity-check matrix H is often represented by a Tanner graph $G = (V, E)$. 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 the entry in the i^{th} row and j^{th} column of H is 1. Figure 2.4 is the Tanner graph associated to the H matrix given in Example 2.8. The degree of a check node i is the number of

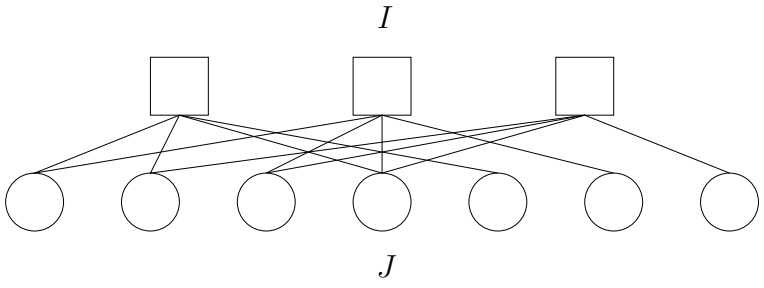


Figure 2.4: Tanner graph.

edges incident to node i in the Tanner graph, or, equivalently, $d_c(i) = \sum_{j=1}^{n} H_{i,j}$. 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.

It should be pointed out that some LPD approaches (see Section 5.5) utilize parity-check matrices being extended by redundant rows, i.e., $H \in \{0,1\}^{(m \times n)}$ with $m > n - k$ and rank$(H) = n - k$.

Before giving the definition of the minimum distance of a code, the Hamming weight of a binary vector and the Hamming distance between two binary vectors are defined in Definitions 2.9 and 2.10. The definitions below can be generalized to all linear codes.

Definition 2.9. *Let $x \in \{0,1\}^n$. The Hamming weight $w(x)$ is given as: $w(x) = |\{j \in \{1,\dots,n\} : x_j \neq 0\}|$.*

Definition 2.10. *Let x^1, $x^2 \in \{0,1\}^n$. $d(x^1,x^2) = \left|\{j \in \{1,\dots,n\} : x_j^1 \neq x_j^2\}\right|$. The measure $d(x^1,x^2)$ is known as the Hamming distance between x^1 and x^2.*

The minimum weight of a binary linear code C is denoted by $w(C) = \min\{w(x) : x \in C, x \neq 0\}$ whereas the minimum distance of a binary linear code $d(C)$ is given by $d(C) = \min\{d(x,x') : x, x' \in C, x \neq x'\}$. The proof of the following result can be found, e.g., in [54, Chapter 3].

Lemma 2.11. *For a binary linear code C, the minimum distance $d(C)$ is equal to minimum weight $w(C)$.*

For the binary symmetric channel, the minimum Hamming distance (minimum distance) between any two distinct codewords is the classical measure for the error-correcting performance of a code. Also for this channel, Hamming spheres are useful to understand the connection between the distance and error-correcting performance of a code. Let $x \in \{0,1\}^n$ and $r \in \mathbb{Z}^+$. A sphere with center x and radius r is defined as

$$S_r(x) := \{y \in \{0,1\}^n | d(x,y) \leq r\}.$$

Example 2.12. *Figure 2.5 is an illustration of pairwise disjoint Hamming spheres (with radius t) of two distinct codewords x and x'. Suppose that during the transmission of a codeword x, less than t bits are flipped in the BSC. Then the received word $y \in \{0,1\}^n$ lies in the sphere $S_t(x)$ and thus, it can be decoded correctly to x.*

Definition 2.13. *Let $C \subseteq \{0,1\}^n$, $t \in \mathbb{N}$, and the transmission channel be the BSC. Then the largest t such that for any distinct x, $x' \in C$, $d(x,x') \geq 2t+1$ (or equivalently $d(C) \geq 2t+1$), is called the random error correction capability of C.*

Remark 2.14. *Similar results and figures can be derived for the binary input additive white Gaussian noise channel.*

By Lemma 2.11, the minimum distance of a binary linear code is the weight of a minimum weight non-zero codeword. Berlekamp et al. [6] conjectured that computing the minimum distance of a binary linear code is NP-hard. This conjecture was first proved by Vardy [68] about two decades later.

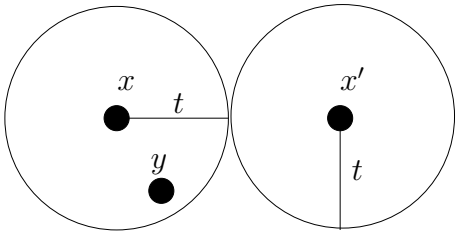


Figure 2.5: Hamming spheres of codewords x, x' with radius r. The received word y can be correctly decoded to codeword x.

2.2.2 Maximum a posteriori and maximum likelihood decoding problem

To regenerate a codeword from a corrupt channel output, decoding rules are applied. In this section we present the decoding rule which is going to be used in the following chapters. First we give the definition of maximum a posteriori decoding. The maximum likelihood decoding (MLD) rule is derived from the maximum a posteriori decoding rule.

Definition 2.15. *The decoding rule which can be explained as: given a received word y, choose the most likely codeword $x^* \in C$ maximizing the conditional probability $P(x|y)$, i.e., $x^* = \text{argmax}_{x \in C} P(x|y)$, is called block-wise maximum a posteriori (MAP) decoding.*

Using Bayes' theorem the probability $P(x|y)$ can equivalently be written as

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}. \tag{2.1}$$

and it holds that

$$x^* = \underset{x \in C}{\text{argmax}} \frac{P(y|x)P(x)}{P(y)}. \tag{2.2}$$

In (2.2), $P(y)$ is a constant since only x is maximized. Furthermore it is assumed that all codewords are equally likely thus $P(x)$ is the same for all $x \in C$. It follows that maximizing (2.1) on $x \in C$ is equivalent to maximizing $P(y|x)$ on $x \in C$. The latter is called block-wise maximum likelihood decoding.

In a memoryless channel, bits are affected by the channel independently so it holds that

$$P(y|x) = \prod_{j=1}^{n} P(y_j|x_j). \tag{2.3}$$

Taking the logarithm on both sides of (2.3), we obtain

$$\log P(y|x) = \sum_{j=1}^{n} \log P(y_j|x_j). \tag{2.4}$$

Maximizing $P(y|x)$ is equivalent to maximizing $\log P(y|x)$ since $\log P(y|x)$ is a monotone increasing function of $P(y|x)$. A block-wise maximum likelihood (ML) decoder, chooses the codeword maximizing the expression in (2.4) as the estimated codeword.

Example 2.16. *We consider MLD in the BSC with crossover probability p, i.e., $P(0|1) = P(1|0) = p$.*

$$\underset{x \in C}{\operatorname{argmax}} P(y|x) = \underset{x \in C}{\operatorname{argmax}} \sum_{j=1}^{n} log P(y_j|x_j) \tag{2.5}$$

$$\Rightarrow \underset{x \in C}{\operatorname{argmax}} \{d(x,y) log p + [n - d(x,y)] \, log(1-p)\} \tag{2.6}$$

$$\Rightarrow \underset{x \in C}{\operatorname{argmax}} \{d(x,y) log \frac{p}{1-p} + n log(1-p)\}. \tag{2.7}$$

The second term of (2.7) is constant. Moreover, we assume that the probability of correct transmission is larger than the probability of error thus $log \frac{p}{1-p} < 0$. Consequently, the maximum of (2.7) is obtained for the codeword $x \in C$ for which $d(x,y)$ is minimum. It follows that for the BSC, MLD decides for the codeword which has the minimum Hamming distance to the received word.

MLD was shown to be NP-hard by Berlekamp et al. [6]. Let us briefly discuss the proof by Berlekamp et al. The proof was based on the reduction of the decision problem of three-dimensional matching to the decision problem of the MLD problem in polynomial time. Berlekamp et al. [6] first described the decision problem Coset Weights.

Definition 2.17. *The decision problem Coset Weights is defined as:*
Given $H \in \{0,1\}^{(m \times n)}$, $s \in \{0,1\}^m$, and $k \in \mathbb{Z}^+$. Does there exist a vector $e \in \{0,1\}^n$ such that $He \equiv s$ mod 2, $w(e) \leq k$?

The Coset Weights problem is related to the MLD problem. To show this we use the BSC. In the BSC, the received word y is the modulo 2 sum of a codeword $x \in C$ and a binary error vector $e \in \{0,1\}^n$, i.e., $y \equiv (x + e)$ mod 2. If MLD is performed over the BSC, then a codeword which has the minimum Hamming distance to the received word y is searched. In other words, the ML codeword x^* is the codeword such that $y \equiv x^* + e$ mod 2 and $w(e)$ is minimum. If we replace x with $(y + e)$ in $Hx \equiv 0$ mod 2, $He \equiv Hy$ mod 2 is obtained. Let $s \equiv Hy$ mod 2. Now, it can be concluded that the ML codeword is given by the modulo 2 sum of y and e such that $He \equiv s$ mod 2 and $w(e)$ is minimum. Setting $k = 1, 2, \ldots$ and solving the Coset Weights problem, the ML decoding problem can be solved in polynomial time if there exists a polynomial time algorithm for the Coset Weights problem.

	121	132	214	223	311	444
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
1	0	0	1	0	1	0
2	1	0	0	1	0	0
3	0	1	0	0	0	0
4	0	0	0	0	0	1
1	1	0	0	0	1	0
2	0	1	0	0	0	0
3	0	0	0	1	0	0
4	0	0	1	0	0	1

Table 2.1: Binary matrix representation of example set U.

Definition 2.18. *Let T be a finite set. The decision problem Three Dimensional Matching is defined as:*
Given a subset $U \subseteq T \times T \times T$, such that the elements of U are ordered triples from T. Does there exist $W \subseteq U$ such that $|W| = |T|$, and no two elements of W agree in any coordinate?

Example 2.19. *An example set given in [6] for $T = \{1, 2, 3, 4\}$ is*

$$U = \{(1, 2, 1), (1, 3, 2), (2, 1, 4), (2, 2, 3), (3, 1, 1), (4, 4, 4)\}. \tag{2.8}$$

It can be verified that $U \supset W = \{(1, 3, 2), (2, 2, 3), (3, 1, 1), (4, 4, 4)\}$ is a matching. A key observation used in the reduction of Three Dimensional Matching problem to Coset Weights problem is that the set U can be encoded to a $3\,|T| \times |U|$ binary matrix. An example is illustrated in Table 2.1.

Obviously, a matching in U exists if and only if there are $|T|$ columns in the binary matrix whose mod 2 sum is the all-ones vector.

Theorem 2.20. *The three-dimensional matching problem can be reduced to Coset Weights problem.*

Proof. Given a Three Dimensional Matching instance U, T, and W, U is encoded to a binary matrix H, s is set to all-ones vector and k is set to $|T|$. The fact that three-dimensional matching is NP-complete [44] implies that Coset Weights problem is NP-complete and thus ML decoding is NP-hard. □

Interestingly, ML decoding is closely related to the minimum weight cycle problem in matroid theory (see Chapter 3). More specifically, there is a one-to-one correspondence between binary matroids and binary linear codes. The cycle problem contains as a special case the max-cut problem which is known to be NP-hard [44]. Therefore, the cycle problem is NP-hard and, thus, matroid theory provides an alternative proof for NP-hardness of ML decoding.

2.2.3 ML decoding as integer programming problem

In [23] Feldman shows that $P(y|x)$ can be replaced by a linear cost function. Recall that we denote the maximum likelihood codeword by x^*. From (2.3), (2.4) it holds that

$$x^* = \underset{x \in C}{\operatorname{argmax}} P(y|x) = \underset{x \in C}{\operatorname{argmax}} \sum_{j=1}^{n} \log P(y_j|x_j).$$

Equivalently,

$$x^* = \underset{x \in C}{\operatorname{argmin}} \left(-\sum_{j=1}^{n} \log P(y_j|x_j) \right). \tag{2.9}$$

If a constant term $\sum_{j=1}^{n} \log P(y_j|x_j = 0)$ is added to the right-hand side of (2.9), the optimum does not change and the following holds.

$$\begin{aligned} x^* &= \underset{x \in C}{\operatorname{argmin}} \left(\sum_{j=1}^{n} \log P(y_j|x_j = 0) - \log P(y_j|x_j) \right) \\ &= \underset{x \in C}{\operatorname{argmin}} \left(\sum_{j \in J: x_j = 1} \log \frac{P(y_j|x_j = 0)}{P(y_j|x_j = 1)} \right) \\ &= \underset{x \in C}{\operatorname{argmin}} \left(\sum_{j=1}^{n} \lambda_j x_j \right). \end{aligned}$$

The value $\lambda_j = \log \frac{P(y_j|x_j=0)}{P(y_j|x_j=1)}$ is called the j^{th} log-likelihood ratio (LLR). Consequently the integer programming formulation of ML decoding is given as

$$\min\{\lambda^T x : x \in C\}. \tag{2.10}$$

2.2.4 Performance measures

The main performance measures of an error-correcting code are its rate, encoding/decoding complexity, and its capability of correcting errors caused by

the channel. Higher rate codes are desirable to minimize redundancy. The complexity of the encoder/decoder pair as well as the block length n affect the latency of transmission. Therefore high complexity encoders, decoders are avoided. The error-correcting capability of a code is directly proportional with the minimum distance.

It is common in coding theory to measure the error-correcting performance of a code or decoding method in terms of frame error or bit error rate versus signal-to-noise ratio. Frame error rate (FER) is the rate of decoding errors in a predetermined number of transmissions whereas signal-to-noise ratio (SNR) is given by SNR$= \frac{E_b}{N_0}$ such that E_b is the energy per transmitted bit and N_0 is the one-sided power spectral density of the channel noise. N_0 is usually expressed in decibels (dBs) (see [50]). Graphically, this is depicted by a plot showing the frame error rate in the vertical axis and the signal-to-noise ratio in the horizontal axis. In the subsequent chapters, these curves will be presented as the main performance measure.

Example 2.21. *An example curve for the ML decoding of a linear block code with* $n = 132$ *and* $r \approx \frac{1}{3}$ *((132,40)- LTE turbo code) is given in Figure 2.6. This curve is obtained by solving the IP formulation of ML decoding problem by a commercial IP solver (see Chapter 4). Frame error rates are determined by counting* 100 *decoding errors. Obviously, as the SNR value increases the rate of decoding errors decreases.*

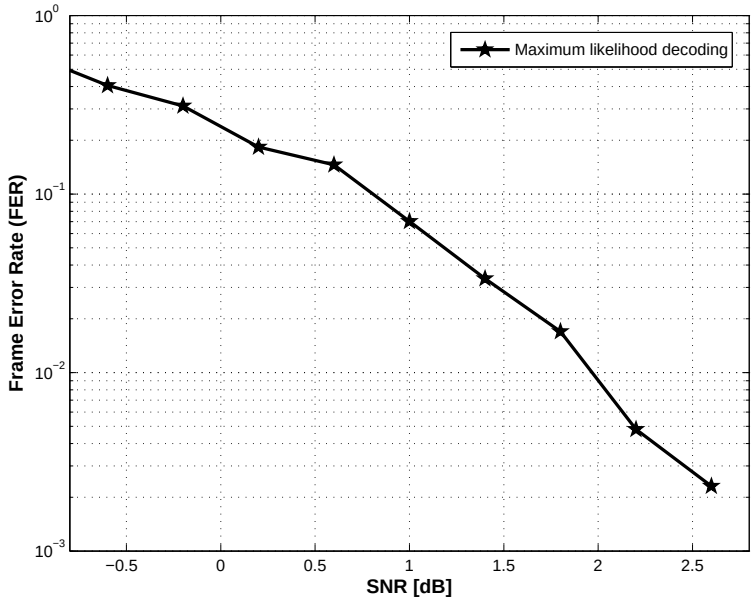


Figure 2.6: ML decoding curve of an irregular (132,40) LTE turbo code.

For a given SNR value, FER should be minimized. Alternatively for a specific FER, the SNR value should be as low as possible. Given a code rate R, the

SNR value for which error free (or an arbitrarily small error probability) communication is achievable, is the Shannon limit (see [50]). The Shannon limit is defined independently of the code structure and relies on an infinite block length.

2.3 Relevant code families

Some binary linear code families considered in this work are turbo codes, low density parity-check (LDPC) codes, expander codes, Hamming codes, and Bose-Chaudhuri-Hocquenghem (BCH) codes. Giving detailed information on the structure of these codes is out of the scope of this thesis. We refer to [50] for a detailed description.

Definition 2.22. *Turbo codes are based on parallel or serial concatenation of at least two convolutional codes (see for example [50]) with interleavers inbetween.*

An interleaver is a device that performs the permutation of input data. The component convolutional codes have individual encoders. Moreover, MLD in polynomial time is possible for convolutional codes. By combining convolutional encoders with interleavers, codes which have very high error-correcting performance can be achieved. This comes at the price that turbo codes are much more complex to decode. There are many different encoding schemes for turbo codes. The standard turbo encoder is explained in Example 2.23

Example 2.23. *In the standard turbo encoder, the first convolutional encoder works directly on the k information bits passed to the encoder and generates k many parity bits we denote by p_1 whereas, the second convolutional encoder receives an interleaved sequence and outputs k many parity bits we denote by p_2. The first k bits, i.e., v, at the output of this standard turbo encoder are the same as k input bits.*

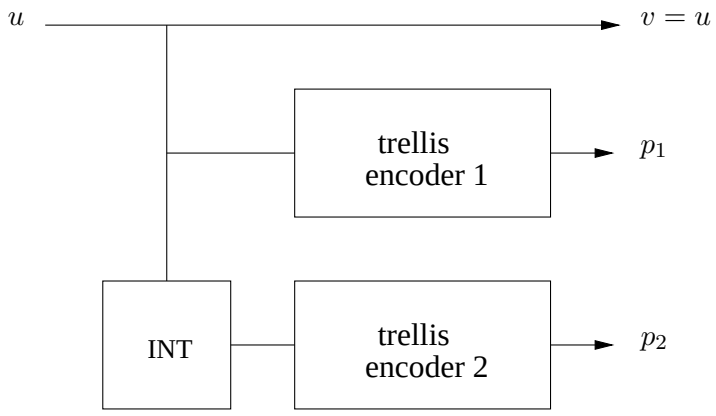


Figure 2.7: Encoder scheme of LTE turbo codes.

The interleaver length used in a turbo code is fixed. Due to this reason, the turbo encoder always generates a fixed number of bits at the output. This fixed block length property makes turbo codes a class of block codes.

Next, we give formal definitions of LDPC codes and expander codes.

Definition 2.24. *[29] A linear code C is a low density parity-check (LDPC) code if it can be described by a Tanner graph whose maximum check node degree d_c^{max} and maximum variable node degree d_v^{max} stay constant as n grows.*

There exist regular and irregular LDPC codes. Let $d_v, d_c \in \mathbb{N}$. In a (d_v, d_c)-regular LDPC code $d_c(i) = d_c$ for all $i \in I$ and $d_v(j) = d_v$ for all $j \in J$. In an irregular LDPC code $d_c(i)$ and $d_v(j)$ vary with $i \in I$ and $j \in J$.

Definition 2.25. *Let G be a (d_v, d_c)-regular bipartite graph where d_v, d_c are variable node and check node degrees respectively. The graph G is called an (α, δ) expander, α, $\delta \in (0, 1)$, if for any subset $|S| \leq \alpha n$ it holds that $|N_S| \geq \delta d_v |S|$.*

Next we discuss Hamming codes.

Definition 2.26. *Let $m \geq 3$ be a positive integer. A Hamming code is a binary linear code with an $(m \times 2^m - 1)$ parity-check matrix such that the $2^m - 1$ non-zero binary vectors of length m constitute the columns.*

Hamming codes can correct 1 error. BCH codes are generalizations of Hamming codes and they can correct multiple errors depending on the type of the code. Given two positive integers $m \geq 3$, $t < 2^{m-1}$, a BCH code with block length $n = 2^m - 1$, number of parity-checks less than or equal to mt, and minimum Hamming distance greater than $2t + 1$ can be constructed. A comprehensive description on BCH codes can be found in [50].

2.4 Polyhedral theory

In this section, some relevant results from polyhedral theory are recalled. For a comprehensive review on polyhedral theory the reader is referred to [46] and [55]. We start with the definitions of affine independence and linear independence.

Definition 2.27. *A set of points $\nu^1, \ldots, \nu^K \in \mathbb{R}^n$ are called affinely independent if the unique solution of $\sum_{k=1}^K \mu_k \nu^k = 0$ and $\sum_{k=1}^K \mu_k = 0$ is $\mu_1 = \mu_2 \ldots = \mu_K = 0$.*

Definition 2.28. *A set of points $\nu^1, \ldots, \nu^K \in \mathbb{R}^n$ are called linearly independent if the unique solution of $\sum_{k=1}^K \mu_k \nu^k = 0$ is $\mu_1 = \mu_2 \ldots = \mu_K = 0$.*

Let $A \in \mathbb{R}^{m \times n}$ denote an $m \times n$ matrix with entries in $\mathbb{R}$, and let $I = \{1, \ldots, m\}$ and $J = \{1, \ldots, n\}$ 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 $A_{i,\cdot}$ and $A_{\cdot,j}$, 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 $h \in I \setminus \{i\}$.

Definition 2.29. *A subset $\mathcal{P}(A, b) \subseteq \mathbb{R}^n$ such that $\mathcal{P}(A, b) = \{\nu \in \mathbb{R}^n : A\nu \leq b\}$ where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ is called a polyhedron. A bounded polyhedron is called a polytope.*

In this work, we assume rational polyhedra, i.e., the entries of A and b are in $\mathbb{Q}$. The i^{th} row vector of A, $A_{i,\cdot}$ with the i^{th} entry of b, b_i, defines a closed halfspace $\{\nu | A_{i,\cdot}\nu \leq b_i\}$. In other words, a polyhedron is the intersection of a finite set of closed halfspaces.

Definition 2.30. *Let $S \subseteq \mathbb{R}^n$ and $\nu \in \mathbb{R}^n$ be given. If there exists a finite non-empty set of points $\{\nu^1, \ldots, \nu^t\}$ in S such that $\nu = \sum_{k=1}^{K} \mu_k \nu^k$, $\mu_k \geq 0$ for $k \in \{1, \ldots, K\}$ and $\sum_{k=1}^{K} \mu_k = 1$, then ν is called a convex combination of points in S. The set of all convex combinations of the points in S is called the convex hull of S and denoted by $conv(S)$.*

It is known from polyhedral theory that the convex hull of a finite set of points is a polytope.

Remark 2.31. *We assume $C \subseteq \{0, 1\}^n$ to be canonically embedded in $\mathbb{R}^n$ and refer to $conv(C)$ as the codeword polytope.*

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

Definition 2.32. *An inequality $r^T\nu \leq t$ where $r \in \mathbb{R}^n$ and $t \in \mathbb{R}$ is called valid for a set $\mathcal{P}(A, b) \subseteq \mathbb{R}^n$ if $\mathcal{P}(A, b) \subseteq \{\nu : r^T\nu \leq t\}$.*

The notion of an active inequality is used in several LPD algorithms. Therefore, a formal definition is given below.

Definition 2.33. *An inequality $r^T\nu \leq t$ where r^T, $\nu \in \mathbb{R}^n$ and $t \in \mathbb{R}$ is active at $\nu^* \in \mathbb{R}^n$ if $r^T\nu^* = t$.*

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

Definition 2.34. *Let $\mathcal{P}(A, b) \subseteq \mathbb{R}^n$ be a polyhedron, let $r^T\nu \leq t$ be a valid inequality for $\mathcal{P}(A, b)$ and define $F = \{\nu \in \mathcal{P}(A, b) : r^T\nu = t\}$. 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. If $\dim(\mathcal{P}(A,b)) = n$, then the polyhedron is full dimensional. It is shown in [55] that if $\mathcal{P}(A,b)$ is not full dimensional then there exists at least one inequality $A_{i,\cdot}\nu \leq b_i$ such that $A_{i,\cdot}\nu = b_i$ holds for all $\nu \in \mathcal{P}(A,b)$. Furthermore it holds that $\dim(F) \leq \dim(\mathcal{P}(A,b)) - 1$ for any proper face of $\mathcal{P}(A,b)$.

Definition 2.35. *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 $A_{i,\cdot}\nu \leq b_i$ may be redundant, i.e, dropping these inequalities does not change the solution set defined by $A\nu \leq b$. In [55], it is shown that a complete and non-redundant description of a polyhedron $\mathcal{P}(A,b)$ can be obtained from the inequalities which are facets.

A point $\nu \in \mathcal{P}(A,b)$ is called an extreme point of $\mathcal{P}(A,b)$ if there exists no other two points $\nu^1, \nu^2 \in \mathcal{P}(A,b)$ such that $\nu = \mu_1\nu^1 + \mu_2\nu^2$ and $0 \leq \mu_1 \leq 1$, $0 \leq \mu_2 \leq 1$, and $\mu_1 + \mu_2 = 1$. Alternatively, extreme points are zero dimensional faces of $\mathcal{P}(A,b)$. In an LP problem, a linear cost function is minimized on a polyhedron, i.e., $\min\{c^T\nu : \nu \in \mathcal{P}(A,b)\}$, $c \in \mathbb{R}^n$. If the LP is not unbounded then the minimum is attained on one of the extreme points (vertices). Below is the formal definition of an optimization problem.

Definition 2.36. *Given a bounded rational polyhedron $\mathcal{P}(A,b) \subset \mathbb{R}^n$ and a rational vector $c \in \mathbb{R}^n$ either find $\nu^* = \operatorname{argmin}\{c^T\nu : A\nu \leq b\}$ or conclude that $\mathcal{P}(A,b)$ is empty.*

Sometimes an optimization problem, may have exponentially many constraints. In this case, towards finding the minimum, it may be favorable to solve the corresponding separation problem. The separation problem over an implicitly given polyhedron is defined as follows.

Definition 2.37. *Given a bounded rational polyhedron $\mathcal{P}(A,b) \subset \mathbb{R}^n$ and a rational vector $\nu^* \in \mathbb{R}^n$, either conclude that $\nu^* \in \mathcal{P}(A,b)$ or, if not, find a rational vector $(r,t) \in \mathbb{R}^n \times \mathbb{R}$ such that $r^T\nu \leq t$ and $r^T\nu < r^T\nu^*$ for all $\nu \in \mathcal{P}(A,b)$. In the latter case (r,t) is called a valid cut.*

The following theorem is a famous result of Grötschel, Lovász, Schrijver [34] which applies to proper polyhedra (see [55]).

Theorem 2.38. *Let $\mathcal{P}$ be a proper class of polyhedra. The optimization problem for $\mathcal{P}$ is polynomial time solvable if and only if the separation problem is polynomial time solvable.*

For the rest of this thesis, besides the definitions and results presented in this chapter, we assume that the reader is somewhat familiar with linear programming and linear programming duality (see, e.g., [60]), integer programming (see, e.g., [55]), matroid theory (see, e.g., [57]), multiple objective optimization (see, e.g., [22]), as well as error-correcting codes (see, e.g., [50]).

Chapter 3

Matroid theory applications in coding theory

3.1 Introduction

Given an $m \times n$ binary matrix H, Barahona and Grötschel studied the polytope $\mathcal{P}(H) = \text{conv}\{x \in \{0,1\}^n : Hx \equiv 0 \bmod 2\}$ in [5]. They gave three interpretations of $\mathcal{P}(H)$. First, $\mathcal{P}(H)$ can be considered as the convex hull of the zero-one solutions of $Hx \equiv 0 \bmod 2$. Second, a binary matrix H can be the representation matrix of a binary matroid $\mathcal{M}[H]$. In this case, a binary vector $x \in \{0,1\}^n$ for which $Hx \equiv 0 \bmod 2$ holds, is the incidence vector of a cycle of $\mathcal{M}[H]$. Thus $\mathcal{P}(H)$ can be considered as the convex hull of the incidence vectors associated with the cycles of a binary matroid. In matroid theory, $\mathcal{P}(H)$ is known as the cycle polytope. A third interpretation of $\mathcal{P}(H)$ is from linear algebra. The set of solutions such that $Hx = 0$ in $(GF(2))^n$ is a linear subspace of $(GF(2))^n$. This set is the kernel of the linear mapping defined by H. Consequently, for a binary linear code C with parity check matrix H, $\mathcal{P}(H)$ is the convex hull of the codewords i.e., $\mathcal{P}(H) = \text{conv}(C)$ (see Remark 2.31). Thus, from the third point of view $\mathcal{P}(H)$ is the codeword polytope. Barahona and Grötschel followed the matroid theoretic point of view in [5] and studied the cycle polytope. The transfer of the polyhedral properties of the cycle polytope to the codeword polytope is straightforward.

As described in [45], there is a one-to-one correspondence between binary matroids and binary linear codes. Kashyap's article [45] is mainly concentrated on transferring matroid decomposition applications explained in [35] to binary linear codes. In this chapter, we supplement this transfer of results by concentrating on the relation between the codeword polytope and the cycle polytope [5], [36]. We first give some basic definitions and theorems from matroid theory to obtain the building blocks. Then we present some results on binary matroids from Seymour [61], Barahona and Grötschel [5] as well

as Grötschel and Truemper [35], [36]. Many of the results from the above mentioned books and articles have very involved proofs, therefore they are not repeated in this work. In the last part we focus on binary linear codes and make some translations from binary matroids to binary linear codes. We briefly summarize the contributions of Kashyap [45] in this area. The examples presented in this chapter are our own work. We also point out our own results and proofs.

This chapter is organized as follows. Some relevant and well-known results from matroid theory are presented in Section 3.2. In Section 3.3, the polyhedral properties of the cycle polytope of a binary matroid are reviewed. Section 3.4 is about decomposition of binary matroids. The relations between binary matroids and binary linear codes are discussed in Section 3.5. The chapter is concluded with some further research ideas in Section 3.6.

3.2 Matroid theory

In order to keep this work self-contained, in this section we present some well-known results from matroid theory. Definitions and theorems are from [39], [57] and [73]. Missing proofs can be found in these references.

A matroid is a mathematical structure which generalizes the concept of independence. The definition of a matroid in general is given below.

Definition 3.1. *A matroid $\mathcal{M}$ is an ordered pair $\mathcal{M} = (J,\mathcal{U})$ where J is a finite ground set and $\mathcal{U}$ is a subset of the power set (set of subsets) of J, denoted by 2^J, such that $(1)-(3)$ hold.*
(1) $\emptyset \in \mathcal{U}$.
(2) If $u \in \mathcal{U}$ and $v \subset u$, then $v \in \mathcal{U}$.
(3) If $u_1, u_2 \in \mathcal{U}$ and $|u_1| < |u_2|$ then there exists a $j \in u_2 \setminus u_1$ such that $u_1 \cup \{j\} \in \mathcal{U}$.

A set $u \in \mathcal{U}$ is called an independent set and the sets in $2^J \setminus \mathcal{U}$ are called dependent sets. The rank function, $r : 2^J \to \mathbb{N}$, for a set $s \in 2^J$ is defined by $r(s) = \max\{|b| : b \subseteq s, b \in \mathcal{U}\}$. The rank of J gives the rank of the matroid $\mathcal{M}$, $r(\mathcal{M})$. Let $b \in \mathcal{U}$ be a subset of J such that $b \cup \{j\} \notin \mathcal{U}$ for any $j \in J \setminus b$. Then b is a maximal independent set and it is called a basis of $\mathcal{M}$. A minimal dependent set, i.e., a set $v \in 2^J \setminus \mathcal{U}$ such that all proper subsets of v are in $\mathcal{U}$, is called a circuit. A one element subset $\{j\} \subset J$ is called a loop if $\{j\}$ is a dependent set, i.e., $\{j\}$ is a circuit. Let $j_1, j_2 \in J$. If $\{j_1, j_2\}$ is a circuit then j_1 and j_2 are called parallel. If a subset of J is a disjoint union of circuits then it is called a cycle. Isomorphism of matroids is defined as follows.

Definition 3.2. *Two matroids $\mathcal{M}_1 = (J_1,\mathcal{U}_1)$ and $\mathcal{M}_2 = (J_2,\mathcal{U}_2)$ are isomorphic if there exists a one-to-one correspondence between the elements of the ground sets J_1 and J_2 such that independence is preserved.*

Let $\mathcal{B}$ be the set of bases of a matroid $\mathcal{M} = (J,\mathcal{U})$. Then $\mathcal{B}^* := \{J \setminus b : b \in \mathcal{B}\}$ is the set of bases of another matroid $\mathcal{M}^*$ called the dual matroid. For the matroid $\mathcal{M}^*$ it holds that $(\mathcal{M}^*)^* = \mathcal{M}$ and $r(\mathcal{M}^*) = |J| - r(\mathcal{M})$. Usually the matroid related terms are dualized with the prefix "co". For example the circuits, cycles and loops of a dual matroid are called cocircuits, cocycles and coloops respectively. A cocircuit with three elements is called a triad.

Matroid minors play an important role in this chapter. Before we give the definition of a minor we introduce the restriction and contraction operations of a matroid. Restriction and contraction are graph theory based concepts which are extended to matroids. Both operations yield submatroids from a given matroid. Deletion and contraction of a loop yields the same submatroid. Below we give the definitions of restriction and contraction as they apply to any matroid. Later on we interpret these two operations for binary matroids.

Definition 3.3. *Let $\mathcal{M} = (J,\mathcal{U})$ be a matroid with ground set J and let $j \in J$. The matroid on $J \setminus \{j\}$, where the independent sets are given by $\mathcal{U}(\mathcal{M}|j) := \{u : u \subseteq J \setminus \{j\}, u \in \mathcal{U}\}$ is called a restriction of the matroid $\mathcal{M}$ and denoted by $\mathcal{M}|j$.*

Example 3.4. *Let $\mathcal{M} = (J,\mathcal{U})$ be a binary matroid with $J = \{1,2,3,4,5\}$ and $\mathcal{U} = \{\emptyset,\{1\},\{2\},\{3\},\{4\},\{5\}\}$. Suppose $j = \{5\}$. Then $\mathcal{M}|5$ is the sub-matroid with $J = \{1,2,3,4\}$ and $\mathcal{U} = \{\emptyset,\{1\},\{2\},\{3\},\{4\}\}$.*

Definition 3.5. *Let $\mathcal{M} = (J,\mathcal{U})$ be a matroid with ground set J and let $j \in J$. The matroid on $J \setminus \{j\}$, where the independent sets are given by $\mathcal{U}(\mathcal{M}/j) := \{u : u \subseteq J \setminus \{j\}, u \cup \{j\} \in \mathcal{U}\}$ is called a contraction of the matroid $\mathcal{M}$ and denoted by $\mathcal{M}/j$.*

Example 3.6. *Let $\mathcal{M} = (J,\mathcal{U})$ and $j \in J$ be defined as in Example 3.4. It follows from the definition that $\mathcal{M}/j$ is the sub-matroid given by the ground set $J = \{1,2,3,4\}$ whose single independent set is the empty set.*

Given a subset $K \subset J$, it is stated in Theorem 3.7 that contraction is the dual operation of restriction and vice versa. The proof can be found in [39].

Theorem 3.7. *Let $\mathcal{M} = (J,\mathcal{U})$ be a matroid with ground set J and let K be a subset of J. Then the following statements hold.*

1. $\mathcal{M}^*/j = (\mathcal{M}|j)^*$.
2. $\mathcal{M}^*|j = (\mathcal{M}/j)^*$.

Matroid minors are defined based on restriction and contraction operations.

Definition 3.8. *A minor of a matroid $\mathcal{M} = (J,\mathcal{U})$ is a sub-matroid obtained from $\mathcal{M}$ by any combination of restriction and contraction operations.*

Minors play an important role in the classification of matroid families.

Definition 3.9. *A familly $\mathcal{T}$ of matroids is called minor closed if for any $\mathcal{M} \in \mathcal{T}$ every matroid isomorphic to a minor of $\mathcal{M}$ is also in $\mathcal{T}$.*

An excluded minor of a matroid is defined as follows.

Definition 3.10. *Given a minor closed family of matroids $\mathcal{T}$, a matroid $\mathcal{M}$ is defined to be an excluded minor of $\mathcal{T}$ if $\mathcal{M} \notin \mathcal{T}$ but every proper minor of $\mathcal{M}$ is in $\mathcal{T}$.*

Excluded minors are used to determine minor closed matroid families.

Theorem 3.11. *Let $\mathcal{K}$ be a finite set of matroids. $\mathcal{T}_{\mathcal{K}}$ is defined as a family of matroids such that $\mathcal{K}$ is the set of excluded minors of $\mathcal{T}$. Then $\mathcal{T}_{\mathcal{K}}$ is minor closed.*

Next, we consider a special class of matroids. An $(m \times n)$ matrix H in the field $\mathbb{F}$ defines a matroid $\mathcal{M}[H]$. The ground set $J = \{1, \ldots, n\}$ is set to the index set of the columns of H. A subset $K \subseteq J$ is independent if and only if the column vectors $H_{.,k}$, $k \in K$ are linearly independent in the vector space defined over the field $\mathbb{F}$. A matroid constructed from a matrix as explained above is called a matrix matroid. We denote a matrix matroid by $\mathcal{M}[H]$. The matrix H is called the representation matrix.

Definition 3.12. *A matroid $\mathcal{M}$ is called $\mathbb{F}$-representable if it is isomorphic to a matrix matroid $\mathcal{M}[H]$ defined over the field $\mathbb{F}$.*

In the next theorem it is stated that the dual of an $\mathbb{F}$-representable matroid is also $\mathbb{F}$-representable.

Theorem 3.13. *[57] Let $\mathcal{M}$ be an $\mathbb{F}$-representable matroid with the representation matrix $H \in \mathbb{F}^{m \times n}$. Then the dual matroid $\mathcal{M}^*$ is also an $\mathbb{F}$-representable matroid with a representation matrix $G \in \mathbb{F}^{(n-m) \times n}$ where the rows of G span the null space of H.*

The class of matroids which is of interest in this work is the class of $\mathbb{F}_2$-representable (binary) matroids. The definition of a binary matroid is given below.

Definition 3.14. *A matroid $\mathcal{M}$ is called binary if it is isomorphic to a matrix matroid $\mathcal{M}[H]$ for some binary matrix $H \in \mathbb{F}_2^{(m \times n)}$.*

Example 3.15. *Let H be a binary matrix having all seven non-zero vectors in $\mathbb{F}_2^3$ as columns, e.g.,*

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

We remark that a binary matroid isomorphic to $\mathcal{M}[H]$ is called a Fano matroid and denoted by F_7.

Let X be a basis of $\mathcal{M}[H]$ and Y denote the index set of non-basic elements. For a given X, H can be brought into the form

$$H = \begin{array}{c|c|c} & X & Y \\ \hline & I & A \end{array}$$

where I is the $m \times m$ identity matrix and $A \in \mathbb{F}_2^{m\times(n-m)}$. This form is called the standard form.

The relation between a binary matroid $\mathcal{M}$ and its dual $\mathcal{M}^*$ follows from Theorem 3.13.

Corollary 3.16. *Let $\mathcal{M}$ be a binary matroid of rank m where $H = [I|A]$ is an $(m \times n)$ representation matrix in standard form. Then the dual matroid $\mathcal{M}^*$ can be represented by an $((n-m) \times n)$ matrix of the form $G = \left[A^T|I\right]$.*

Binary matroids have some important properties stated in Theorem 3.18. First we give the definition of symmetric difference between two finite sets.

Definition 3.17. *Let v_1, v_2 be two finite sets. The symmetric difference of v_1 and v_2 is given by $v_1 \Delta v_2 := (v_1 \setminus v_2) \cup (v_2 \setminus v_1)$.*

Theorem 3.18. *[57] The following statements about a matroid $\mathcal{M}$ are equivalent.*

1. *$\mathcal{M}$ is binary.*
2. *For every circuit $\mathcal{C}$ and every cocircuit $\mathcal{D}$, $|\mathcal{C} \cap \mathcal{D}|$ is even.*
3. *If $\mathcal{C}_1$ and $\mathcal{C}_2$ are distinct circuits then $C_1 \Delta C_2$ contains a circuit.*

Another property of binary matroids is that restriction and contraction can be performed easily. $\mathcal{M}[H]|K$ is obtained by deleting the columns indexed by the set K from the representation matrix H. Concerning the contraction operation in a binary matroid, the representation matrix of $\mathcal{M}[H]/K$ is obtained from the original representation matrix H by applying a procedure resulting from Theorem 3.7. This procedure is presented as the binary matroid contraction algorithm in the next page.

Some examples of binary matroid families used later in this chapter are regular matroids, graphic matroids, and cographic matroids. A regular matroid can be defined in several ways. First, a matroid is regular if it is representable over every field. Second, the concept of total unimodularity is used to define a regular matroid. In a totally unimodular matrix, the determinant of every square submatrix is in $\{-1, 0, 1\}$. A matroid is called regular if the associated representation matrix is totally unimodular. The third way of defining a regular matroid is deduced from Theorem 3.19. The proof is given in [57].

Binary matroid contraction algorithm

1: **for all** $j \in K$ **do**
2: Transfer $H_{.,j}$ into a unit vector by elementary row operations in $GF(2)$.
3: **for all** $i \in I$ **do**
4: **if** $H_{ij} = 1$ **then**
5: Delete row i, column j from H.
6: **end if**
7: **end for**
8: **end for**
9: Delete all zero columns from the new representation matrix.

Theorem 3.19. *A binary matroid is regular if and only if it does not contain as a minor any matroid isomorphic to F_7 or F_7^*.*

Two important subclasses of regular matroids are graphic and cographic matroids. Let $\mathbb{G} = (V, E)$ be a graph where V is the set of vertices and E is the set of edges. The incidence matrix of $\mathbb{G}$ is a binary matrix which contains a row for every vertex and a column for each edge. The entry in row $v \in V$ and column $e \in E$ is 1 if edge e is incident to vertex v and 0 if it is not. A regular matroid is called *graphic* if the associated representation matrix is the incidence matrix of a graph $\mathbb{G}$. An alternative definition of a graphic matroid is given below.

Definition 3.20. *A regular matroid is graphic if and only if it does not contain as a minor any matroid isomorphic to the matroids $(\mathcal{M}[K_5])^*$ or $(\mathcal{M}[K_{3,3}])^*$.*

A matroid $\mathcal{M}[H]$ is called cographic if $(\mathcal{M}[H])^*$ is isomorphic to a graphic matroid, or, alternatively:

Definition 3.21. *A regular matroid is cographic if and only if it does not contain as a minor any matroid isomorphic to the matroids $\mathcal{M}[K_5]$ or $\mathcal{M}[K_{3,3}]$.*

If $\mathcal{M}[H]$ is a cographic matroid, then all cycles of $\mathcal{M}[H]$ correspond to the cuts in the graph associated with the graphic matroid $(\mathcal{M}[H])^*$. $\mathcal{M}[K_5]$ and $\mathcal{M}[K_{3,3}]$ are graphic matroids where the associated vertex edge incidence matrices can be written as

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

and

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

, respectively. The associated graphs are given in Figure 3.1 and Figure 3.2. Note that the standard representation matrices of $\mathcal{M}[K_5]$ and $\mathcal{M}[K_{3,3}]$ can be obtained from the vertex edge incidence matrices by elementary row operations. The representation matrices of $(\mathcal{M}[K_5])^*$ and $(\mathcal{M}[K_{3,3}])^*$ are derived from Corollary 3.16.

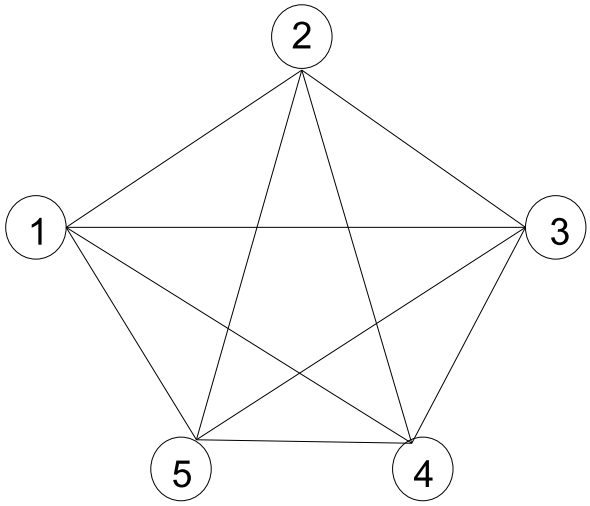


Figure 3.1: The graph K_5 associated with the matroid $\mathcal{M}[K_5]$.

Finally we present a representation matrix H of another important matroid denoted by R_{10}. This matroid is used in subsequent sections.

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

3.3 Cycle polytope

The polytope associated to the set of cycles of a binary matroid is known as the cycle polytope. We first define the incidence vectors corresponding to the

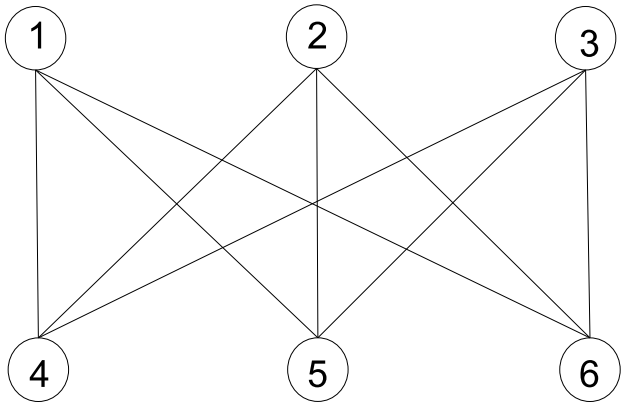


Figure 3.2: The graph $K_{3,3}$ associated with the matroid $\mathcal{M}[K_{3,3}]$.

cycles. In general, an incidence vector corresponding to a subset $\mathcal{C} \subseteq J$ is constructed as

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

for all $j \in J$. The cycle polytope is the convex hull of all incidence vectors corresponding to all cycles of a binary matroid. Polyhedral properties of the cycle polytope are studied by Barahona and Grötschel in [5]. We denote the cycle polytope given by the binary matroid $\mathcal{M}[H]$ by $\mathcal{P}(\mathcal{M}[H])$.

$$\mathcal{P}(\mathcal{M}[H]) := \text{conv}(\{x^{\mathcal{C}} \in \mathbb{R}^n : \mathcal{C} \subseteq J \text{ is a cycle}\}).$$

The dimension of $\mathcal{P}(\mathcal{M}[H])$ can be calculated using the notion of coparallel classes given in [5].

Definition 3.22. *A coparallel class of a binary matroid $\mathcal{M}[H]$ is a maximal subset $K \subseteq J$ such that K contains no coloops and every two distinct members of K are coparallel.*

Coparallel classes with one element are called trivial. If $\mathcal{M}[H]$ has no coloops and coparallel elements then it has only trivial coparallel classes.

Theorem 3.23. *Let $\mathcal{M}[H]$ be a binary matroid and $\mathcal{P}(\mathcal{M}[H])$ be the associated cycle polytope. The dimension of the cycle polytope $\mathcal{P}(\mathcal{M}[H])$ is given by the number of the coparallel classes of $\mathcal{M}[H]$.*

If there exists no coloops and coparallel elements of $\mathcal{M}[H]$ then the number of coparallel classes of $\mathcal{M}[H]$ is $|J|$.

Corollary 3.24. *If $\mathcal{M}[H]$ does not contain any coloop or coparallel elements, then $\mathcal{P}(\mathcal{M}[H])$ is full-dimensional, i.e., dim($\mathcal{P}(\mathcal{M}[H])$)=$|J|$.*

In the polyhedral analysis of $\mathcal{P}(\mathcal{M}[H])$, the symmetry property stated in Theorem 3.25 plays an important role.

Theorem 3.25. *[5] If $a^T x \leq \alpha$ defines a face of $\mathcal{P}(\mathcal{M}[H])$ of dimension d, and $\mathcal{C}$ is a cycle, then the inequality $\bar{a}^T x \leq \bar{\alpha}$ also defines a face of $\mathcal{P}(\mathcal{M}[H])$ of dimension d, where*

$$\bar{a}_j := \begin{cases} a_j, & \text{if } j \notin \mathcal{C} \\ -a_j, & \text{if } j \in \mathcal{C} \end{cases}$$

and $\bar{\alpha} := \alpha - a^T x^{\mathcal{C}}$.

Using this theorem, Barahona and Grötschel [5] showed that if all facets containing a given cycle are known then the complete description of $\mathcal{P}(\mathcal{M}[H])$ can be obtained.

Some valid inequalities for the cycle polytope $\mathcal{P}(\mathcal{M}[H])$ are the trivial inequalities

$$0 \leq x_j \leq 1 \; \forall j \in J, \tag{3.1}$$

and the so-called cocircuit inequalities (forbidden set inequalities, see Chapter 4),

$$\sum_{j \in \mathcal{F}} x_j - \sum_{j \in \mathcal{D} \setminus \mathcal{F}} x_j \leq |\mathcal{F}| - 1 \text{ for all } \mathcal{F} \subseteq \mathcal{D} \text{ and } |\mathcal{F}| \text{ odd}, \tag{3.2}$$

where $\mathcal{D}$ is a cocircuit. To show that cocircuit inequalities are valid, we propose our own proof.

Proof. Let $\mathcal{C}$ and $\mathcal{D}$ be any cycle, cocycle pair of a binary matroid $\mathcal{M}[H]$. We show that the left-hand side of Inequality (3.2) is less than or equal to $|\mathcal{F}| - 1$ for all cycles.

Case 1:
If $\mathcal{C} \cap \mathcal{D} \nsubseteq \mathcal{F}$, then there exists at least one variable x_j, $j \in \mathcal{D} \setminus \mathcal{F}$ such that $x_j = 1$. It follows that the left-hand side of Inequality (3.2) can be maximally $|\mathcal{F}| - 1$.

Case 2:
If $\mathcal{C} \cap \mathcal{D} \subseteq \mathcal{F}$, then $|\mathcal{C} \cap \mathcal{D}| \leq |\mathcal{F}| - 1$ since $|\mathcal{C} \cap \mathcal{D}|$ is even due to the second property of Theorem 3.18. □

Some conditions for which a trivial inequality or a cocircuit inequality defines a facet of $\mathcal{P}(\mathcal{M}[H])$ are given in [5]. For the proofs of Theorem 3.26 and Theorem 3.28 we refer to [5].

Theorem 3.26. *Let $\mathcal{M}[H]$ be a binary matroid such that:*

(1) $\mathcal{M}[H]$ has no coloops and coparallel elements. If $j \in J$ is not contained in a triad, then $x_j \geq 0$, $x_j \leq 1$ are facets of $\mathcal{P}(\mathcal{M}[H])$.

(2) $\mathcal{M}[H]$ has no F_7^* minor. If there exists a triad $\{j, j', j''\} \subset J$ then the following inequalities are facets of $\mathcal{P}(\mathcal{M}[H])$

$$\begin{aligned} x_j + x_{j'} + x_{j''} &\leq 2 \\ x_j - x_{j'} - x_{j''} &\leq 0 \\ -x_j + x_{j'} - x_{j''} &\leq 0 \\ -x_j - x_{j'} + x_{j''} &\leq 0. \end{aligned}$$

To identify if a cocircuit inequality is facet defining it should be checked if the associated cocircuit has a chord.

Definition 3.27. *Let $\mathcal{C}$ be a cocircuit. If there exists two cocircuits $\mathcal{C}'$, $\mathcal{C}''$ such that $\mathcal{C}' \cap \mathcal{C}'' = j$, $j \in J \setminus \mathcal{C}$ and $\mathcal{C}' \Delta \mathcal{C}'' = \mathcal{C}$, then j is called a chord of $\mathcal{C}$.*

Theorem 3.28. *Let $\mathcal{M}[H]$ be a binary matroid without an F_7^* minor and $\mathcal{D}$ be a cocircuit with at least 3 elements and no chord. Then the inequalities*

$$\sum_{j \in \mathcal{F}} x_j - \sum_{j \in \mathcal{D} \setminus \mathcal{F}} x_j \leq |\mathcal{F}| - 1, \mathcal{F} \subseteq \mathcal{D}, |\mathcal{F}| \text{ odd}$$

are facets of $\mathcal{P}(\mathcal{M}[H])$.

Example 3.29. *Let $\mathcal{M}[H]$ be a binary matroid with the representation matrix*

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

There are 7 cocircuits of $\mathcal{M}[H]$: $\{\{1,4,5\}, \{2,4,6\}, \{3,5,6\}, \{1,2,5,6\}, \{1,3,4,6\}, \{2,3,4,5\}, \{1,2,3\}\}$. It can easily be verified that 4, 5, 6 are chords. In particular, for the cocircuit $\{1,2,5,6\}$ there exists cocircuits $\mathcal{C}' = \{1,4,5\}$, $\mathcal{C}'' = \{2,4,6\}$ such that $\mathcal{C}' \cap \mathcal{C}'' = \{4\}$, $4 \in \{1,2,3,4,5,6\} \setminus \{1,2,5,6\}$ and $\{1,4,5\}\Delta\{2,4,6\} = \{1,2,5,6\}$. Since the cocircuits $\{1,2,5,6\}$, $\{1,3,4,6\}$, $\{2,3,4,5\}$ contain a chord the cocircuit inequalities that can be derived from these cocircuits do not define facets of $\mathcal{P}(\mathcal{M}[H])$.

An interesting question answered in [5] is: Are the trivial and cocircuit inequalities sufficient to describe $\mathcal{P}(\mathcal{M}[H])$ in general? Let $\mathcal{Q}(\mathcal{M}[H])$ be the

polytope which is described by the trivial inequalities and the cocircuit inequalities, i.e.,

$$\mathcal{Q}(\mathcal{M}[H]) := \{x \in \mathbb{R}^n : x \text{ satisfies } (3.1),\ (3.2) \text{ for all cocircuits }\}. \tag{3.3}$$

It is stated in Theorem 3.30 that for some special binary matroids, the cycle polytope can be described completely by the trivial inequalities and the cocircuit inequalities. These matroids have the so called *sum of circuits* property.

Theorem 3.30. *[5] $\mathcal{P}(\mathcal{M}[H]) = \mathcal{Q}(\mathcal{M}[H])$ if and only if $\mathcal{M}$ has none of the following minors: $F_7^*, R_{10}, (\mathcal{M}[K_5])^*$.*

3.3.1 Complete binary matroids

In [36], Grötschel and Truemper introduce a family of binary matroids called complete binary matroids. Complete binary matroids are denoted by L_m for $m \geq 1$. L_1 contains a single loop. For $m \geq 2$, L_m is represented by a matrix $[I|A^m]$ where A^m is an $(2^m - m - 1) \times m$ zero-one matrix and I is the $(2^m - m - 1) \times (2^m - m - 1)$ identity matrix. The $(2^m - m - 1)$ rows of A^m are all possible distinct $0-1$ row vectors with the exception of the 0 vector and the m unit vectors.

Example 3.31.

$$A^2 = [1\ 1] \text{ and } A^3 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

A^{m+1} is obtained from A^m by the following scheme where I is the $m \times m$ identity matrix and 0 and 1 are column vectors of appropriate size.

$$A^{m+1} = \begin{bmatrix} A^m & 0 \\ A^m & 1 \\ I & 1 \end{bmatrix}.$$

For the ease of notation, before stating the main theorem on complete binary matroids, the notion of representative inequalities has to be introduced. Let the inequality $a^T x \leq \alpha$ be a face of a cycle polytope. All inequalities $\bar{a}^T x \leq \bar{\alpha}$ which can be derived from $a^T x \leq \alpha$ and any cycle $\mathcal{C}$ by using Theorem 3.25 are represented by $a^T x \leq \alpha$.

Grötschel and Truemper [36] collected the results on complete binary matroids in Theorem 3.33. However, a proof of Theorem 3.33 was not included in [36]. We summarize in Proposition 3.32 the partial results which help in the understanding of Theorem 3.33. We also propose our own proofs. Note that a full understanding of complete binary matroids, helps in identifying the polyhedral properties of simplex codes.

Proposition 3.32. *Let m be an integer larger than 2.*

(1) For $m \geq 3$, a circuit $\mathcal{C}$ of L_m cannot consist only of elements from the index set of the columns of A^m.

(2) For $m \geq 2$, in every column of A^m there are $2^{m-1} - m$ zeros and $2^{m-1} - 1$ ones.

(3) Every circuit $\mathcal{C}$ of L_m has cardinality 2^{m-1}.

(4) Every non-empty cycle of L_m is actually a circuit.

(5) For $m \geq 2$, any L_m has 2^m cycles.

Proof. In the first part of the proof it is assumed that $m \geq 3$. In the remaining parts it is assumed that $m \geq 2$.

(1) $\mathcal{C}$ is a circuit of L_m if and only if the elements of $\mathcal{C}$ are the indices of the columns of a minimal Eulerian column submatrix of $\left[\ I \mid A^m\ \right]$, i.e., a minimal column submatrix where each row has an even number of ones. By the definition of A^m, no column submatrix of A^m can be Eulerian. Hence, a circuit cannot consist only of elements from the index set of the columns of A^m.

(2) By construction A^m contains all possible distinct $0-1$ row vectors with the exception of the 0 vector and the m unit vectors. For a fixed column $A^m_{\cdot,i}$, $i \in \{1,\ldots,m\}$, there are $\binom{m-1}{k}$ rows of A^m which have a zero at position i and have exactly k ones for $k = 2,3,\ldots,m-1$. Hence, column i has $\sum_{j=2}^{m-1}\binom{m-1}{j} = 2^{m-1} - (m-1) - 1 = 2^{m-1} - m$ zeros. Since A^m has $2^m - m - 1$ rows, each column contains $2^m - m - 1 - (2^{m-1} - m) = 2^{m-1} - 1$ ones.

(3) The proof is done by induction on m. For $m = 2$ the statement is obvious. Consider the case $m = 3$:
The complete binary matroid is $L_3 = \left[\ I \mid A^3\ \right]$. The index sets of the columns of I and A^3 are $\{1,2,3,4\}$ and $\{5,6,7\}$, respectively. Due to (1), any circuit $\mathcal{C}$ of L_3 can be written as $\mathcal{C} = X \cup Y$, where $X \subseteq \{1,2,3,4\}$ and $Y \subseteq \{5,6,7\}$. Three cases have to be distinguished:
Case 1: $|Y| = 1$.
Since every column of A^3 has 3 ones, X must contain 3 elements of $\{1,2,3,4\}$ such that there are two ones in every row of the column submatrix built by the columns whose indices are in $\mathcal{C}$. Hence, $|\mathcal{C}| = 4 = 2^{3-1}$.
Case 2: $|Y| = 2$.
No two columns of A^3 have all their ones at the same positions and there is a single zero in every column. Hence, a two column submatrix of A^3 has two rows that have a single one. A circuit $\mathcal{C}$ containing the indices of these two columns must also contain the indices of two unit vectors of I

such that the column submatrix indexed by $\mathcal{C}$ has two ones in each row. Again, $|\mathcal{C}| = 4 = 2^{3-1}$.
Case 3: $|Y| = 3$.
Only one of the rows of A^3 has an odd number of ones. Hence, a circuit of L^3 containing the indices of all 3 columns of A^3 must also contain the index of one unit vector from I such that the mentioned row has an even number of ones in the column submatrix of $\left[\ I \mid A^3\ \right]$ indexed by $\mathcal{C}$. Thus, $|\mathcal{C}| = 4 = 2^{3-1}$.

Assume that the statement is true for all $m = 2, 3, 4, \ldots$.
Inductive Step:
Let $\mathcal{C}_m = X_m \cup Y_m$ be a circuit of L_m with $X_m \subseteq \{1, 2, \ldots, |2^m - m - 1|\}$ and $Y_k \subseteq \{|2^m - m|, \ldots, |2^m - 1|\}$. Then $|\mathcal{C}_m| = |X_m| + |Y_m| = 2^{m-1}$.

A circuit $\mathcal{C}_{m+1}$ of L_{m+1} can be distinguished from the circuits of L_m in three ways.
Case 1:
Choose the columns of A^{m+1} that contain the $|Y_m|$ columns of A^m whose indices are in C_m. Then a circuit $\mathcal{C}_{m+1}$ of L_{m+1} containing the indices of these columns must also contain the indices of $2\,|X_m| + |Y_m|$ distinct columns from $\{1, 2, \ldots, |2^m - m - 1|\}$ by the structure of A^{m+1}. Hence, $|\mathcal{C}_{m+1}| = |Y_m| + 2\,|X_m| + |Y_m| = 2\,|\mathcal{C}_m| = 2^m$.
Case 2:
Choose last column of A^{m+1}. In order to have a Eulerian column submatrix of $\left[\ I \mid A^{m+1}\ \right]$:

- the columns of A^{m+1} that contain the $|Y_m|$ columns of A^m, whose indices are in C_m,
- $|X_m|$ columns from I for the first $2^m - m - 1$ rows,
- $2^m - m - 1 - |X_m|$ further columns from I for the second $2^m - m - 1$ rows,
- $m - |Y_m|$ further columns from I for the remaining rows

have to be chosen. Hence $|\mathcal{C}_{m+1}| = 2^{m-1}+1+2^m-m-1-|X_m|+m-|Y_m| = 2^{m-1} + 2^m - 2^{m-1} = 2^m$.
Case 3:
Choose only the last column of A^{m+1}. Then $2^m - 1$ appropriate columns from I should also be chosen. The indices of these columns form a circuit $\mathcal{C}$ of L_{m+1} and $|\mathcal{C}_{m+1}| = 2^m - 1 + 1 = 2^m$.

(4) Let C be a circuit of L_m. Then $2\,|\mathcal{C}| = 2^m > 2^m - 1$, i.e, no disjoint union of circuits is possible.

(5) For any subset $\emptyset \neq Q \subseteq \{|2^m - m|, \ldots, |2^m - 1|\}$ there exists a circuit $\mathcal{C}$ of L_m that contains Q. Since $\emptyset$ is a cycle by definition and every non-empty

cycle of L_m is actually a circuit, L_m has $1 + \binom{m}{1} + \binom{m}{2} + \ldots + \binom{m}{m} = 2^m$ cycles. The cycles of L_2 are $\emptyset, \{1,2\}, \{1,3\}, \{2,3\}$. Thus, the statement also holds for $m = 2$.

□

Now we state the theorem of Grötschel and Truemper [36]. It is followed by an example.

Theorem 3.33. *[36] For any $m \geq 1$, the complete binary matroid L_m has $2^m - 1$ elements and $2^m - 1$ non-empty cycles. Each such cycle C is a circuit, and $|C| = 2^{m-1}$. The cycle polytope $P(L_m)$ is a full-dimensional simplex with 2^m vertices. The facet defining inequalities of $P(L_m)$ are represented by the inequality*

$$\sum_{j \in J} x_j \leq 2^{m-1}.$$

Example 3.34. *Consider L_3 (This binary matroid is associated with the $[7,3,4]$ simplex code.) with the representation matrix below.*

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

It can easily be verified that L_3 consists of the cycles $\{\emptyset, \{1,2,3,5\}, \{1,2,4,6\}, \{2,3,4,7\}, \{3,4,5,6\}, \{$

The representative inequality is

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 \leq 4. \tag{3.4}$$

It follows from Theorem 3.33 that the system of inequalities

$$\begin{array}{rrrrrrr} x_1 & +x_2 & +x_3 & +x_4 & +x_5 & +x_6 & +x_7 \leq 4 \\ -x_1 & -x_2 & -x_3 & +x_4 & -x_5 & +x_6 & +x_7 \leq 0 \\ -x_1 & -x_2 & +x_3 & -x_4 & +x_5 & -x_6 & +x_7 \leq 0 \\ x_1 & -x_2 & -x_3 & -x_4 & +x_5 & +x_6 & -x_7 \leq 0 \\ x_1 & +x_2 & -x_3 & -x_4 & -x_5 & -x_6 & +x_7 \leq 0 \\ -x_1 & +x_2 & -x_3 & +x_4 & +x_5 & -x_6 & -x_7 \leq 0 \\ -x_1 & +x_2 & +x_3 & -x_4 & -x_5 & +x_6 & -x_7 \leq 0 \\ -x_1 & -x_2 & -x_3 & +x_4 & +x_5 & +x_6 & -x_7 \leq 0 \end{array}$$

describe $\mathcal{P}(L_3)$ (the codeword polytope of the $[7,3,4]$ simplex code) completely and non-redundantly.

Let J be the ground set of L_m and $\mathcal{P}(L_m)$ denote the cycle polytope of L_m. Grötschel and Truemper [36] call $\mathcal{P}(L_m)$ the master polytope. The motivation for the name "master" follows from the observation that for a binary matroid $\mathcal{M}[H]$ without coloops and coparallel elements, there exists a relation between the facets of $\mathcal{P}(L_m)$ and the facets of $\mathcal{P}(\mathcal{M}[H])$.

In the rest of this section it is assumed that a given binary matroid $\mathcal{M}[H]$ has no coloops and coparallel elements. Grötschel and Truemper [36] show that $\mathcal{M}[H]$ with the representation matrix $H = [I|A]$ can be obtained by contraction operations from a complete binary matroid L_m. If the complete binary matroid $L_m = [I|A^m]$ is chosen such that A is a row sub-matrix of A^m, then $\mathcal{M}[H] = L_m/N$ for some $N \subseteq J$. Consequently $\mathcal{P}(\mathcal{M}[H])$ is obtained by projecting out the components x_e, $e \in N$ from $\mathcal{P}(L_m)$.

Although it is theoretically possible, in general deriving $\mathcal{P}(\mathcal{M}[H])$ from $\mathcal{P}(L_m)$ is computationally inefficient. Two reasons for that are:

(1) exponentially many projections may be needed,

(2) no computationally efficient procedure for projection is known.

A special case for which $\mathcal{P}(\mathcal{M}[H])$ can be derived from $\mathcal{P}(L_m)$ is given in Theorem 3.35.

Theorem 3.35. *Let $e \in J$. For $m \geq 2$ the system of facet defining inequalities for $P(L_m/e)$ is represented by 2^{m-1} inequalities of the type*

$$\sum_{j \notin \mathcal{C}} x_j \leq 2^{m-2}$$

where each set of inequalities corresponds to a circuit $\mathcal{C}$ of L_m with $e \in \mathcal{C}$.

Example 3.36. *Let $m = 3$ and $e = 4$. The representation matrix of L_3 is already given in Example 3.34 and the representation matrix for L_m/e is*

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

where the column index set is $J = \{1,2,3,5,6,7\}$. Four cycles of L_3 which contain index 4 are $\{\{1,2,4,6\},\{2,3,4,7\},\{3,4,5,6\},\{1,4,5,7\}\}$. The system of facet defining inequalities of $P(L_3/4)$ are represented by the inequalities

$$\begin{aligned} x_3 + x_5 + x_7 &\leq 2 \\ x_1 + x_5 + x_6 &\leq 2 \\ x_1 + x_2 + x_7 &\leq 2 \\ x_2 + x_3 + x_6 &\leq 2. \end{aligned}$$

Grötschel and Truemper [36] also propose a lifting procedure to deduce the facets of an arbitrary cycle polytope $\mathcal{P}(\mathcal{M}[H])$ from a certain master polytope $P(L_m)$, $m \geq 1$. In this procedure it is checked if $\mathcal{M}[H]$ contains contraction minors (see Definition 3.38). The inequalities associated with the maximum of these minors (see Definition 3.38) are lifted to define facets of $\mathcal{P}(\mathcal{M}[H])$. Concerning the complexity[1] of the lifting procedure, the dominating operation is the determination of the maximal complete contraction minor of $\mathcal{M}[H]$. It was shown that such a minor can be found in polynomial time. By employing the lifting procedure only a subset of the facets of $\mathcal{P}(\mathcal{M}[H])$ can be found.

Let $H = [I|A]$ be the representation matrix in standard form with basis X. Then $J = X \cup Y$ where Y is the set of non-basic elements. For the ease of notation Grötschel and Truemper [35] introduce a short representation matrix R. In R only the sub-matroid A is written. The rows and columns of R are indexed by X and Y respectively.

$$R = \begin{array}{c|c} & Y \\ \hline X & A \end{array}$$

Example 3.37. *We consider again L_3 with the representation matrix given in Example 3.34. Let $X = \{1, 2, 3, 4\}$ the basis of the matroid. Then*

$$H = \begin{array}{c|cccc|ccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}$$

and

$$R = \begin{array}{c|ccc} & 5 & 6 & 7 \\ \hline 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 \end{array}.$$

[1]Here and in the following chapters the term complexity means time complexity unless stated otherwise.

In their lifting procedure Grötschel and Truemper [36] first search for maximal complete contraction minors in the binary matroid $\mathcal{M}[H]$.

Definition 3.38. *A minor $\hat{\mathcal{M}}$ of a binary matroid $\mathcal{M}[H]$ is a complete contraction minor if*

1. *$\hat{\mathcal{M}}$ is obtainable from $\mathcal{M}[H]$ by using only contraction operations,*
2. *$\hat{\mathcal{M}}$ is isomorphic to a complete binary matroid L_m, $m \geq 1$.*

$\hat{\mathcal{M}}$ isomorphic to L_m for some m is called a maximal complete contraction minor if $\mathcal{M}[H]$ does not contain L_{m+1} as a complete contraction minor.

Any element of $\mathcal{M}[H]$ forms a complete contraction minor L_1. The existence of L_2 in $\mathcal{M}[H]$ is checked by searching a triad. A binary matroid has L_2 as a complete contraction minor if and only if it contains a triad. In general, suppose L_m where $m \geq 2$ is a complete contraction minor of $\mathcal{M}[H]$. If Lemma 3.39 is satisfied, L_{m+1} is also a complete contraction minor of $\mathcal{M}[H]$.

Lemma 3.39. *Let L_m, where $m \geq 2$, be a contraction minor of a binary matroid $\mathcal{M}[H]$. Additionally, let J_0 be the ground set of L_m which can be partitioned into X_0 and Y_0, $J_0 = X_0 \cup Y_0$. Extend X_0 to an arbitrary basis X of $\mathcal{M}[H]$. Then $\mathcal{M}[H]$ has L_{m+1} as a contraction minor if and only if the short representation matrix R^X with basis X is equal to R_1 (see Figure 3.3) or R_2 (see Figure 3.4).*

The rows of the short representation matrix R_1 given in Figure 3.3 is partitioned into X_0, X_1, and X_2 so that X_0 is a basis of L_m, $X_0 \cup X_1$ is a basis of L_{m+1} and $X_0 \cup X_1 \cup X_2$ is a basis of $\mathcal{M}[H]$. The columns of R_1 are also partitioned into Y_0, e, and Y_2. The non-basic elements of L_m are collected in Y_0. $Y_0 \cup \{e\}$ is the set of non-basic elements of L_{m+1}. Finally $Y_0 \cup \{e\} \cup Y_2$ contains the non-basic elements of $\mathcal{M}[H]$. Grötschel and Truemper [36] show that any other short representation matrix of $\mathcal{M}[H]$ has either the structure of R_1 except that the index sets X_2 and Y_2 are different or it has the structure of R_2.

For a given element $j \in J$, a polynomial time procedure which finds the maximal complete contraction minor containing j is described in [36]. We rewrite this procedure as an algorithm and refer to it as maximal complete contraction minor algorithm (MCCMA). Repeating MCCMA for all $j \in J$, the maximal complete contraction minor of $\mathcal{M}[H]$ can be found.

Having found a maximal complete contraction minor, the main result used in the lifting procedure of Grötschel and Truemper [36] is stated in Theorem 3.40.

Theorem 3.40. *Let $\mathcal{M}[H]$ be a binary matroid without coloops and coparallel elements. Suppose a complete binary matroid L_m with ground set J_0 is a complete contraction minor of $\mathcal{M}[H]$. Then the following statements are equivalent.*

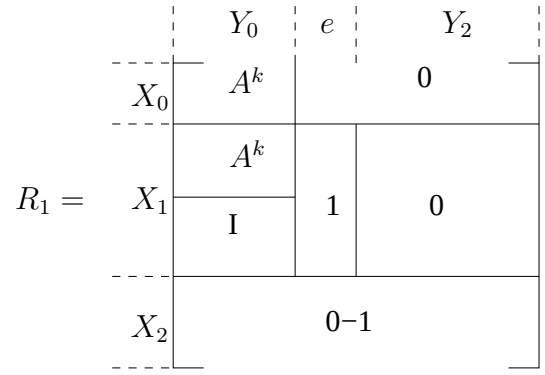


Figure 3.3: Short representation matrix 1.

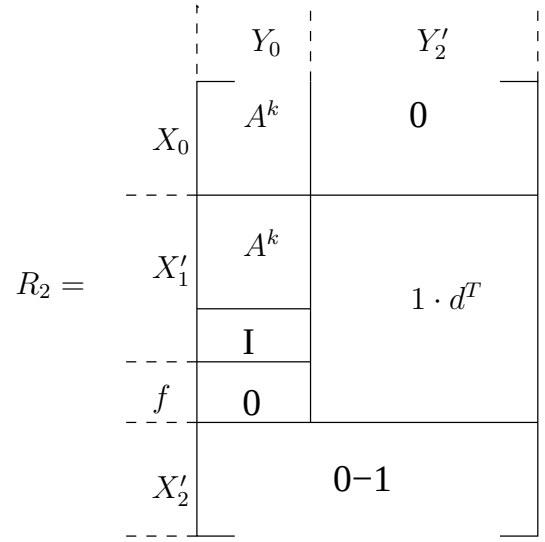


Figure 3.4: Short representation matrix 2.

Maximal complete contraction minor algorithm
Input: A binary matroid $\mathcal{M}[H]$, $j \in J$.
Output: A maximal complete contraction minor L_m.

```
m = 1.
if x_j is contained in a triad then
    repeat
        m = m + 1.
        Select an arbitrary basis X of M[H] such that:
        X contains a basis X_0 of L_m,
        X does not contain any other element of L_m.
    until R^X ≠ R_1 and R^X ≠ R_2.
end if
Output L_m.
```

1. *L_m is a maximal complete contraction minor of $\mathcal{M}[H]$.*
2. *Any facet defining inequality of $P(L_m)$ also defines a facet of $P(\mathcal{M}[H])$.*
3. *At least one facet defining inequality of $P(L_m)$ defines a facet of $P(\mathcal{M}[H])$.*
4. *The facet defining inequality*

$$\sum_{j \in J_0} x_j \leq 2^{m-1}$$

of $P(L_m)$ also defines a facet of $P(\mathcal{M}[H])$.

3.4 Decomposition of binary matroids

An essential research direction within the theory of matroids is the decomposition of matroids. Based on Seymour's work [61] it can be shown that some matroids can be composed from smaller matroids by performing some suitable operations. The resulting matroid is referred to as the k-sum, $k \in \mathbb{Z}$, $k \geq 1$, of smaller matroids. Conversely, it is possible to decompose some matroids into smaller matroids under certain assumptions.The simplest k-sum is for $k = 1$, known as the direct sum. The following theorem from [57] describes the direct sum operatian denoted by $\oplus$.

Theorem 3.41. *Let $\mathcal{M}' = (J', \mathcal{U}')$ and $\mathcal{M}'' = (J'', \mathcal{U}'')$ be two matroids and $J' \cap J'' = \emptyset$. Then there exists a matroid $\mathcal{M} = (J, \mathcal{U})$ such that $J = J' \cup J''$ and $\mathcal{U} = \{U' \cup U'' : U' \in \mathcal{U}', U'' \in \mathcal{U}''\}$ called the direct sum of $\mathcal{M}'$ and $\mathcal{M}''$ and denoted by $\mathcal{M}' \oplus \mathcal{M}''$.*

Example 3.42. *Let $\mathcal{M}[H_1]$ and $\mathcal{M}[H_2]$ be two binary matroids with representation matrix H_1 and H_2 respectively. The representation matrix H of $\mathcal{M}[H_1] \oplus \mathcal{M}[H_2]$ is:*

$$H = \begin{bmatrix} H_1 & 0 \\ 0 & H_2 \end{bmatrix}$$

where 0 denotes a matrix of zeros with appropriate dimensions.

Definitions 3.43 and 3.45 formulated by Tutte [67] play an important role in the composition and decomposition of matroids.

Definition 3.43. *Given a positive integer k and a matroid $\mathcal{M} = (J, \mathcal{U})$. A partition (T, T^C) of the ground set J is called a Tutte k-separation of $\mathcal{M}$ if*

$$\min\{|T|, |T^C|\} \geq k, \tag{3.5}$$

and

$$r(\mathcal{M}|T^C) + r(\mathcal{M}|T) - r(\mathcal{M}) \leq k - 1. \tag{3.6}$$

Note that $\mathcal{M}|T^C$ and $\mathcal{M}|T$ are the restrictions of $\mathcal{M}$ on the sets $T, T^C \subseteq J$. If (3.5) is satisfied with equality, then the separation is called a minimal k-separation. If (3.6) is satisfied with equality, then the separation is called an exact k-separation. There is a trivial relationship between the Tutte 1-separated matroids and the matroids constructed by direct sum operatin. We prove this in Theorem 3.44.

Theorem 3.44. *A matroid $\mathcal{M}$ is Tutte 1-separated if and only if it can be constructed from two non-empty matroids via direct sum.*

Proof: Let J be the ground set of the matroid $\mathcal{M}$. For any partition (T, T^C) of J it holds that $r(T) + r(T^C) \geq r(J)$. It follows that (T, T^C) is a 1-separation of $\mathcal{M}$ if and only if T and T^C are not empty and $r(T) + r(T^C) = r(J)$. The latter is equivalent to $C = C|T^C \oplus C|T$. □

Definition 3.45. *Let $l \in \mathbb{Z}^+$ and $k \in \mathbb{Z}^+$, $k \geq 2$. A matroid $\mathcal{M}$ is called Tutte k-connected if it has no Tutte l-separation for $l < k$.*

In the following we concentrate on composition and decomposition of binary matroids via k-sums, $k = 2, 3$. Definitions below are from Seymour [61].

Definition 3.46. *Given two binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$ and $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$ such that $\min\{|J_1|, |J_2|\} \geq 3$. Suppose $J_1 \cap J_2 = \{e\}$ and e is not a loop or coloop of $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$. Then the binary matroid $\mathcal{M}[H] = (J, \mathcal{U})$, $J = J_1 \Delta J_2$, with the property that all cycles $\mathcal{C}$ of $\mathcal{M}[H]$ are of the form $\mathcal{C} = \mathcal{C}_1 \Delta \mathcal{C}_2$ where $\mathcal{C}_1$ and $\mathcal{C}_2$ are the cycles of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$, respectively, is a 2-sum of $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$.*

Definition 3.47. *Given two binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$ and $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$ such that $\min\{|J_1|, |J_2|\} \geq 7$. Suppose $J_1 \cap J_2 = \{e, f, g\}$ where the set $\{e, f, g\}$ is a circuit of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ but contains no cocircuit of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$. Then the matroid $\mathcal{M}[H] = (J, \mathcal{U})$, $J = J_1 \Delta J_2$, with the property that all cycles $\mathcal{C}$ of $\mathcal{M}[H]$ are of the form $\mathcal{C} = \mathcal{C}_1 \Delta \mathcal{C}_2$ where $\mathcal{C}_1$ and $\mathcal{C}_2$ are the cycles of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ respectively, is a 3-sum of $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$.*

Combining Definitions 3.46 and 3.47 with the properties of Tutte k-separation, $k = 2, 3$, the following existence statements given in Theorem 3.48 and 3.49 can be made. These theorems have involved proofs which can be found in [61].

Theorem 3.48. *Let (T, T^C) be an exact 2-separation of a binary matroid $\mathcal{M}[H] = (J, \mathcal{U})$, $J = T \cup T^C$ and $\{e\}$ a new element. Then there exists two binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$ and $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$ such that $J_1 = T \cup \{e\}$, $J_2 = T^C \cup \{e\}$ and $\mathcal{M}[H]$ is a 2-sum of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$. Conversely if $\mathcal{M}[H]$ is a 2-sum of two binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ then $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ are isomorphic to proper minors of $\mathcal{M}[H]$ and (T, T^C) where $T = J_1 \setminus \{e\}$, $T^C = J_2 \setminus \{e\}$ is an exact Tutte 2-separation of $\mathcal{M}[H]$.*

Theorem 3.49. *Let (T, T^C) be an exact 3-separation of a binary matroid $\mathcal{M}[H] = (J, \mathcal{U})$, $J = T \cup T^C$ such that $\min\{|T|, |T^C|\} \geq 4$ and $\{e, f, g\}$ be a set of new elements. Then there exists two binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$ and $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$ such that $J_1 = T \cup \{e, f, g\}$, $J_2 = T^C \cup \{e, f, g\}$ and $\mathcal{M}[H]$ is a 3-sum of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$. Conversely if $\mathcal{M}[H]$ is a 3-sum of two binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ then $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ are isomorphic to proper minors of $\mathcal{M}$ and (T, T^C) where $T = J_1 \setminus \{e, f, g\}$, $T^C = J_2 \setminus \{e, f, g\}$ is an exact Tutte 3-separation of $\mathcal{M}[H]$ with $\min\{|T|, |T^C|\} \geq 4$.*

Although Seymour presented the necessary and sufficient conditions for 2 and 3-sums to exist, a composition scheme for obtaining $\mathcal{M}[H]$ from $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$, or a decomposition scheme of $\mathcal{M}[H]$ into $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ is not given in [61].

Grötschel and Truemper observe that the decomposition theory of Seymour has some important implications on the problem of minimizing a linear cost function over the cycles of a binary matroid which is called the cycle problem. Specifically, if a binary matroid $\mathcal{M}[H]$ is a k-sum, $k = 1, 2, 3$, of two binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ then each circuit of $\mathcal{M}[H]$ can be composed from the circuits of $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$. For some binary matroid classes, the decomposition theory results in a polynomial time algorithm for the cycle problem.

If $k = 1$ then a circuit $\mathcal{C}$ of $\mathcal{M}[H]$ is either a circuit in one of the matroids $\mathcal{M}_i[H_i]$, $i = 1, 2$, or there exists two circuits $\mathcal{C}_1, \mathcal{C}_2$ in $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$ such that $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$. In order to make statements for $k = 2, 3$, some (de)composition schemes are introduced in [35]. Next we summarize the (de)composition results from [35].

Theorem 3.50. *Given an exact Tutte k-separation (T, T^C) of $\mathcal{M}[H]$, $k \geq 1$, there exists a basis X_2 of $\mathcal{M}[H]|T$ and an independent set X_1 of $\mathcal{M}|T^C$ such that $X := X_1 \cup X_2$ is a basis of $\mathcal{M}$ and the short representation matrix R corresponding to the basis X is of the form*

$$R = \begin{array}{c|c|c} & Y_1 & Y_2 \\ \hline X_1 & A_1 & 0 \\ \hline X_2 & D & A_2 \end{array} \tag{3.7}$$

where $T = X_1 \cup Y_1$, $T^C = X_2 \cup Y_2$ and $r(D) = k - 1$.

Conversely, a matrix of the form given in (3.7) defines a Tutte k-separation ($X_1 \cup Y_1$, $X_2 \cup Y_2$) if $\min\{|X_1 \cup Y_1|, |X_2 \cup Y_2|\} \geq k$. The proof of this theorem can be found in [39].

Let $\mathcal{M}[H]$ be a binary matroid and there exists an exact Tutte 2-separation (T, T^C) of $\mathcal{M}[H]$. In the decomposition scheme of Grötschel and Truemper, the short representation matrix of the matroid to be decomposed is brought in the form given in (3.7). Then $\mathcal{M}[H]$ is decomposed into binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ with short representation matrices R_1^e and R_2^e respectively.

$$R_1^e = \begin{array}{c|c} & Y_1 \\ \hline X_1 & A_1 \\ \hline e & a \end{array} \qquad R_2^e = \begin{array}{c|c|c} & e & Y_2 \\ \hline X_2 & u & A_2 \end{array}. \tag{3.8}$$

For $k = 2$, $r(D) = 1$ according to Theorem 3.50. It follows that all non-zero rows of D or all non-zero columns of D are identical. Let a and u denote the non-zero row and column of D respectively. The short representation matrix R_1^e of $\mathcal{M}_1[H_1]$ is obtained from A_1 in (3.7) and the non-zero row of D. The non-zero row of D is indexed by e in R_1^e. The short representation matrix R_2^e of $\mathcal{M}_2[H_2]$ is constructed by A_2 of (3.7) and the non-zero column of D. The index of the non-zero column of D is also set to e.

Example 3.51. *Let $\mathcal{M}[H]$ be a binary matroid with the standard representation matrix*

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

It can easily be verified that (T, T^C), $T := \{1, 4\}$ *and* $T^C := \{2, 3\}$ *is an exact Tutte* 2*-separation of* $\mathcal{M}[H]$. *Let* X_1, Y_1, X_2, *and* Y_2 *of Theorem 3.50 are set to* $\{1\}$, $\{4\}$, $\{2, 3\}$ *and* $\emptyset$ *respectively. Then the short representation matrix* R *is*

$$R = \begin{array}{c|c} & 4 \\ \hline 1 & 0 \\ 2 & 1 \\ 3 & 1 \end{array}.$$

Using the decomposition scheme in (3.8) for 2*-sums, the short representation matrices* R_1^e *and* R_2^e *are given as*

$$R_1^e = \begin{array}{c|c} & 4 \\ \hline 1 & 0 \\ e & 1 \end{array} \qquad R_2^e = \begin{array}{c|c} & e \\ \hline 2 & 1 \\ 3 & 1 \end{array}.$$

It follows that the standard representation matrices H_1, H_2 *of* $\mathcal{M}_1[H_1]$ *and* $\mathcal{M}_2[H_2]$ *with* $J_1 = \{1, e, 4\}$, $J_2 = \{2, 3, e\}$ *are*

$$H_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad H_2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Conversely if the assumptions of Definition 3.46 hold then $\mathcal{M}[H]$ can be composed from $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ by reversing the decomposition scheme. The submatrices A_1 and A_2 are identified in R_1^e and R_2^e respectively. The submatrix D is given by $D = a \cdot u$. Grötschel and Truemper use the notation $\mathcal{M}[H] = \mathcal{M}_1[H_1] \oplus_e \mathcal{M}_2[H_2]$ to denote a 2-sum decomposition where the index e indicates that 2-sum is performed over the element e. It is stated that if $\mathcal{M}[H] = \mathcal{M}_1[H_1] \oplus_e \mathcal{M}_2[H_2]$ then a circuit $\mathcal{C}$ of $\mathcal{M}[H]$ is either a circuit $\mathcal{C}_i$ of $\mathcal{M}_i[H_i]$, $i = 1, 2$, or there exists circuits $\mathcal{C}_1$, $\mathcal{C}_2$ of $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$ such that $\mathcal{C} = (\mathcal{C}_1 \Delta \mathcal{C}_2)$ and $e \in \mathcal{C}_1, \mathcal{C}_2$.

Composition and decomposition schemes for 3-sums are more involved. Furthermore there exists alternative schemes. Two schemes given in [35] are referred to as Δ-sum and Y-sum. Under the assumptions of Theorem 3.49 there exists $X_1 \subseteq T$ and $X_2 \subseteq T^C$ such that (3.7) in Theorem 3.50 is of the form

$$R = \begin{array}{c|ccc|c} & \multicolumn{3}{c|}{Y_1} & Y_2 \\ \hline X_1 & \multicolumn{3}{c|}{A_1} & 0 \\ \hline & a & 1 & 0 & \\ X_2 & b & 0 & 1 & A_2 \\ & \bar{D} & u & v & \end{array}. \tag{3.9}$$

Note that

$$\begin{array}{|c|c|c|}\hline a & 1 & 0 \\ \hline b & 0 & 1 \\ \hline \bar{D} & u & v \\ \hline \end{array} = D.$$

The first 3-sum decomposition scheme is the Δ-sum. Let $\mathcal{M}[H]$ be a binary matroid with the short representation matrix R given in (3.9). Then $\mathcal{M}[H]$ can be decomposed into binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ with the short representation matrices R_1^Δ and R_2^Δ respectively.

$$R_1^\Delta = \begin{array}{c|c|c|c|c} & & Y_1 & & g \\ \hline X_1 & \multicolumn{3}{c|}{A_1} & 0 \\ \hline e & a & 1 & 0 & 1 \\ \hline f & b & 0 & 1 & 1 \\ \hline \end{array} \qquad R_2^\Delta = \begin{array}{c|c|c|c|c} & e & f & g & Y_2 \\ \hline X_2 & 1 & 0 & 1 & \multirow{2}{*}{A_2} \\ & 0 & 1 & 1 & \\ \hline & u & v & w & \\ \hline \end{array} \tag{3.10}$$

According to Theorem 3.50 for $k = 3$, $r(D) = 2$. It follows that the submatrix D has two linearly independent rows and two linearly independent columns. Considering (3.9), the two linearly independent rows of D are the rows containing a, b and the two linearly independent columns of D are the columns containing u and v. To obtain the short representation matrix R_1^Δ of $\mathcal{M}_1[H_1]$, first the two rows of D containing a and b are added to A_1. The added rows are indexed with e and f. Second a new column is added to the extended A_1 matrix. The index of the new column is set to g. The column with index g has zero entries except the rows e and f. In these two rows, column g has 1 as entries. In order to obtain the short representation matrix R_2^Δ of $\mathcal{M}_2[H_2]$ the two columns of the submatrix D containing u, v are added to A_2. The two new columns are indexed by e and f. Finally, a third new column, indexed by g, is found by adding the columns e and f.

Example 3.52. *Let $\mathcal{M}[H]$ be a binary matroid with the standard representation matrix*

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

It can be verified that (T, T^C) is an exact Tutte 3-separation of $\mathcal{M}[H]$, where $T = \{1, 7, 8, 9\}$ and $T^C = \{2, 3, 4, 5, 6, 10, 11, 12\}$. Let X_1, Y_1, X_2, and Y_2 of Theorem

3.50 be set to $\{1\}$, $\{7,8,9\}$, $\{2,3,4,5,6\}$ *and* $\{10,11,12\}$, *respectively. Then the short representation matrix of* $\mathcal{M}[H]$ *is*

$$R = \begin{array}{c|c|c|c|ccc|} & 7 & 8 & 9 & 10 & 11 & 12 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 2 & 0 & 1 & 0 & 1 & 0 & 0 \\ \hline 3 & 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 4 & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline 5 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 6 & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array}.$$

Using the decomposition scheme (3.10) the short representation matrices R_1^Δ *and* R_2^Δ *of* $\mathcal{M}_1[H_1]$ *and* $\mathcal{M}_2[H_2]$ *are given as*

$$R_1^\Delta = \begin{array}{c|c|c|c|c|} & 7 & 8 & 9 & g \\ \hline 1 & 0 & 1 & 1 & 0 \\ \hline e & 0 & 1 & 0 & 1 \\ \hline f & 1 & 0 & 1 & 1 \\ \hline \end{array} \qquad R_2^\Delta = \begin{array}{c|c|c|c|ccc|} & e & f & g & 10 & 11 & 12 \\ \hline 2 & 1 & 0 & 1 & 1 & 0 & 0 \\ \hline 3 & 0 & 1 & 1 & 1 & 1 & 1 \\ \hline 4 & 0 & 1 & 1 & 0 & 1 & 1 \\ \hline 5 & 0 & 1 & 1 & 0 & 0 & 1 \\ \hline 6 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}.$$

Conversely, if the assumptions of Definition 3.47 hold then $\mathcal{M}[H]$ can be composed from $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ by reversing the Δ-sum decomposition scheme. The submatrices A_1, a, b and A_2, u, v are identified in R_1^Δ and R_2^Δ respectively. The submatrix $\bar{D}$ is given by

$$\bar{D} = \begin{array}{|c|c|} \hline u & v \\ \hline \end{array} \cdot \begin{array}{|c|} \hline a \\ \hline b \\ \hline \end{array}$$

An alternative 3-sum decomposition scheme given in [35] is the Y-sum. A Definition similar to Definition 3.47 for Y-sum can be written.

Definition 3.53. *Given two binary matroids* $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$ *and* $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$ *such that* $\min\{|J_1|, |J_2|\} \geq 7$. *Suppose* $J_1 \cap J_2 = \{r,s,t\}$ *where the set* $\{r,s,t\}$ *is a cocircuit of* $\mathcal{M}_1[H_1]$ *and* $\mathcal{M}_2[H_2]$ *but contains no circuit of* $\mathcal{M}_1[H_1]$ *and* $\mathcal{M}_2[H_2]$. *Then the matroid* $\mathcal{M}[H] = (J, \mathcal{U})$, $J = J_1 \Delta J_2$ *with the property that all cycles* $\mathcal{C}$ *of* $\mathcal{M}[H_1]$ *are of the form* $\mathcal{C} = \mathcal{C}_1 \Delta \mathcal{C}_2$ *where* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are the cycles of* $\mathcal{M}_1[H_1]$ *and* $\mathcal{M}_2[H_2]$ *respectively, is a* 3*-sum of* $\mathcal{M}_1[H_1]$, $\mathcal{M}_2[H_2]$.

Let $\mathcal{M}[H]$ be a binary matroid for which the assumptions in Theorem 3.49 hold. Then $\mathcal{M}[H]$ can be decomposed into $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ with the short

representation matrices R_1^Y and R_2^Y respectively.

$$R_1^Y = \begin{array}{c|c|c|c} & Y_1 & & \\ X_1 & A_1 & & \\ \hline r & a & 1 & 0 \\ s & b & 0 & 1 \\ t & c & 1 & 1 \end{array} \qquad R_2^Y = \begin{array}{c|cc|c} & r & s & Y_2 \\ t & 1 & 1 & 0 \\ \hline & 1 & 0 & \\ X_2 & 0 & 1 & A_2 \\ & u & v & \end{array} \tag{3.11}$$

In the Y-sum decomposition of a suitable $\mathcal{M}[H]$, first the short representation matrix of the form (3.9) is written. To obtain R_1^Y of $\mathcal{M}_1[H_1]$, the two rows of the submatrix D which contain a and b are added to A_1. The added rows are indexed with r and s. Then a third row which is the sum of the rows indexed with r and s is added to the extended A_1 matrix. The third row is indexed with t. R_2^Y of $\mathcal{M}_2[H_2]$ is constructed by first adding two columns to A_2 and then adding a row to the extended matrix. The new added columns are the two linearly independent columns of the submatrix D of (3.9) which contain u and v. The indices of these columns are set to r and s. The new added row has all 0 entries except in columns r, s. The index of the row is set to t.

Example 3.54. *Let $\mathcal{M}[H]$ be the binary matroid with the representation matrix H given in Example 3.52. Moreover the exact Tutte 3-separation (T, T^C) as well as the sets X_1, Y_1, X_2, and Y_2 are chosen as before. Then the short representation matrices R_1^Y, R_2^Y of the binary matroids $\mathcal{M}_1[H_1]$ and $\mathcal{M}_2[H_2]$ are*

$$R_1^Y = \begin{array}{c|ccc} & 7 & 8 & 9 \\ \hline 1 & 1 & 1 & 0 \\ r & 0 & 1 & 0 \\ s & 1 & 0 & 1 \\ t & 1 & 1 & 1 \end{array} \qquad R_2^Y = \begin{array}{c|cc|c} & r & s & 10\ 11\ 12 \\ \hline t & 1 & 1 & 0\,0\,0 \\ \hline 2 & 1 & 0 & 1\,0\,0 \\ 3 & 0 & 1 & 1\,1\,1 \\ \hline 4 & 0 & 1 & 0\,1\,1 \\ 5 & 0 & 1 & 0\,0\,1 \\ 6 & 0 & 1 & 0\,0\,0 \end{array}.$$

Let $\mathcal{M}[H]$ be a binary matroid for which the assumptions of Theorem 3.49 are valid. Then there exists $\mathcal{M}_1^\Delta[H_1], \mathcal{M}_2^\Delta[H_2]$ ($\mathcal{M}_1^Y[H_1], \mathcal{M}_2^Y[H_2]$) such that $\mathcal{M}[H] = \mathcal{M}_1^\Delta[H_1] \oplus_\Delta \mathcal{M}_2^\Delta[H_2]$ ($\mathcal{M}[H] = \mathcal{M}_1^Y[H_1] \oplus_Y \mathcal{M}_2^Y[H_2]$). One of the main results in [35] is that every circuit $\mathcal{C}$ of $\mathcal{M}[H]$ can be obtained from circuits $\mathcal{C}_1^\Delta$, $\mathcal{C}_2^\Delta$ of $\mathcal{M}_1^\Delta[H_1], \mathcal{M}_2^\Delta[H_2]$ or equivalently from circuits $\mathcal{C}_1^Y$, $\mathcal{C}_2^Y$ of $\mathcal{M}_1^Y[H_1], \mathcal{M}_2^Y[H_1]$. The details of the composition of $\mathcal{C}$ from $\mathcal{C}_1^\Delta$, $\mathcal{C}_2^\Delta$ or $\mathcal{C}_1^Y$, $\mathcal{C}_2^Y$ are given in Proposition 3.55 and 3.56.

Proposition 3.55. *Let $\mathcal{M}[H] = (J, \mathcal{U})$ be a binary matroid obtained by the Δ-sum of binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$, $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$, i.e.,$\mathcal{M}[H] = \mathcal{M}_1[H_1] \oplus_\Delta$*

$\mathcal{M}_2[H_2]$ and $J_1 \cap J_2 = \{e, f, g\}$. If $\mathcal{C}$ is a circuit of $\mathcal{M}[H]$ then $\mathcal{C}$ is either a circuit $\mathcal{C}_i$ of $\mathcal{M}_i[H_i]$, $i = 1, 2$ or $\mathcal{C} = \mathcal{C}_1 \Delta \mathcal{C}_2$ such that there exists a unique $z \in \{e, f, g\}$ and $C_i := (C \cap (J_i \setminus \{e, f, g\})) \cup \{z\}$ is a circuit of $\mathcal{M}_i[H_i]$ for $i = 1, 2$.

Proposition 3.56. *Let $\mathcal{M}[H] = (J, \mathcal{U})$ be a binary matroid obtained by the Y-sum of binary matroids $\mathcal{M}_1[H_1] = (J_1, \mathcal{U}_1)$, $\mathcal{M}_2[H_2] = (J_2, \mathcal{U}_2)$, i.e.,$\mathcal{M}[H] = \mathcal{M}_1[H_1] \oplus_Y \mathcal{M}_2[H_2]$ and $J_1 \cap J_2 = \{r, s, t\}$. If $\mathcal{C}$ is a circuit of $\mathcal{M}[H]$ then $\mathcal{C}$ is either a circuit $\mathcal{C}_i$ of $\mathcal{M}_i[H_i]$, $i = 1, 2$ or $\mathcal{C} = \mathcal{C}_1 \Delta \mathcal{C}_2$ such that there exists a unique two element subset $\Omega \subset \{r, s, t\}$ and $C_i := (C \cap (J_i \setminus \{r, s, t\})) \cup \Omega$ is a circuit of $\mathcal{M}_i[H_i]$ for $i = 1, 2$.*

The cycle problem can be solved in polynomial time for the graphic matroids and the cographic matroids without $(\mathcal{M}[K_5])^*$ minor. The former result is shown by Edmonds and Johnson [21] using a matching algorithm and the latter result is proved by Barahona [4]. (De)composition techniques used in [35] extend the matroid classes for which the cycle problem is polynomially solvable. Explaining these techniques in detail is out of the scope of our work. Instead we briefly mention the main results.

First of all, it should be shown that there exists a polynomial time algorithm which performs 2 and 3-sum decompositions. Grötschel and Truemper show that for any connected binary matroid $\mathcal{M}[H]$ without F_7 or F_7^* minor 2 and 3-sum decompositions can be performed in polynomial time.

One solution approach to the NP hard cycle problem is to maximize the linear cost function over a relaxation of $\mathcal{P}(\mathcal{M}[H])$. If the solution is non-integral, the relaxed polytope is iteratively tightened via valid inequalities until the optimum is integer or no valid cut can be found. A relaxation $\mathcal{Q}(\mathcal{M}[H])$ which contains exponentially many constraints is known from Section 3.3. Due to the result of Gröetschel, Lovasz, Schrijver [34] which we introduced as Theorem 2.38 in Chapter 2, the cycle problem on $\mathcal{Q}(\mathcal{M}[H])$ is polynomial time solvable if and only if the separation problem for $\mathcal{Q}(\mathcal{M}[H])$ is polynomial time solvable. In [35] it is shown that the separation problem for $\mathcal{Q}(\mathcal{M}[H])$ can be solved in polynomial time if $\mathcal{M}[H]$ is

- graphic,
- cographic,
- a matroid without F_7 or F_7^* minor.

The last matroid class is obtained by applying a decomposition approach. A polynomial time separation algorithm for the matroids listed above implies that the corresponding optimization problem can be solved for the matroid classes given in the following corollary.

Corollary 3.57. *Let $\lambda \in \mathbb{R}^n$ and $\mathcal{M}[H]$ be a matroid with n elements. Then the linear program $\max\{\lambda^T x : x \in \mathcal{Q}(\mathcal{M}[H])\}$ can be solved in polynomial time if $\mathcal{M}[H]$*

- *has no F_7 or F_7^* minor,*
- *has the sum of circuits property,*
- *is cographic*
- *is graphic.*

From Theorem 3.30 it is known that $\mathcal{P}(\mathcal{M}[H]) = \mathcal{Q}(\mathcal{M}[H])$ if and only if the underlying matroid has the sum of circuits property. Consequently for the matroids with the sum of circuits property, the cycle problem can be solved in polynomial time. (De)composition techniques enable to make more general statements about the separation problem for $\mathcal{Q}(\mathcal{M}[H])$. Grötschel and Truemper state that there exists a polynomial time separation algorithm for $\mathcal{Q}(\mathcal{M}[H])$ for all matroids $\mathcal{M}[H]$ which can be decomposed recursively via 2-sums, Δ-sums into graphic, cographic matroids or to a matroid from a prespecified finite list provided that 2 and Δ-sum decompositions can be performed in polynomial time.

In the theoretical statements which eventually lead to Corollary 3.57, it is assumed that the ellipsoid algorithm is used to solve LP problems. The reason for that is the runtime of the ellipsoid algorithm is provably polynomial. However in practice the ellipsoid algorithm is slower than the simplex method or the interior-point methods (or its variants) and thus it has less practical use.

For the cycle problem of some binary matroids, a polynomial time combinatorial algorithm is proposed in [35]. This algorithm can be found in [35], [45], [39][2]. We do not repeat the algorithm here but give the main result on this combinatorial algorithm and its implications.

Theorem 3.58. *[35] Let $\mathcal{N}$ be a matroid family which contains the graphic matroids and a finite number of matroids which are not graphic. Moreover let $\mathbb{M}$ be a matroid family such that for each $\mathcal{M} \in \mathbb{M} \setminus \mathcal{N}$ one can determine in polynomial time a 2-sum $\mathcal{M} = \mathcal{M}_1 \oplus_e \mathcal{M}_2$ or a Y-sum $\mathcal{M} = \mathcal{M}_1 \oplus_Y \mathcal{M}_2$ where $\mathcal{M}_1 \in \mathcal{N}$ and $\mathcal{M}_2 \in \mathbb{M}$. Then there is a polynomial time combinatorial algorithm that solves the cycle problem for all matroids in $\mathbb{M}$.*

Corollary 3.59. *Let $\lambda \in \mathbb{R}^n$ and $\mathcal{M}[H]$ is a matroid with n elements. Then the linear program $\max\{\lambda^T x : x \in \mathcal{P}(\mathcal{M}[H])\}$ can be solved in polynomial time if $\mathcal{M}[H]$*

- *has no F_7 and $(\mathcal{M}[K_5])^*$ minor,*
- *has no F_7^* and $(\mathcal{M}[K_5])^*$ minor,*
- *has the sum of circuits property,*
- *is cographic and has no $(\mathcal{M}[K_5])^*$ minor,*
- *is graphic.*

[2] An example can also be found in this master thesis.

3.5 Matroid theory applications in coding theory

There exists a unique binary matroid corresponding to each binary linear code and the converse is also true (see also [45]). A binary linear code C is the null space of a binary matrix $H \in \{0,1\}^{m\times n}$. This null space is equal to the circuit space of a binary matroid $\mathcal{M}[H]$ where H is a representation matrix. Note that elementary row operations on H alters neither the binary linear code C nor the binary matroid $\mathcal{M}[H]$. Two binary linear codes are equivalent, i.e.,one code can be obtained from the other by a permutation of the bit positions, if and only if the corresponding binary matroids are isomorphic. The next two definitions facilitate the translations between the coding theoretic terminology and the matroid theoretic terminology.

Definition 3.60. *Let C be a binary linear code with parity check matrix H and $x \in C \subseteq \{0,1\}^n$. The index set $supp(x) = \{j \in J : x_j = 1\}$ is called the support of the codeword x.*

Definition 3.61. *A codeword x, $0 \neq x \in C$, is called a minimal codeword if there is no codeword $0 \neq y \in C$ such that $supp(y) \subseteq supp(x)$.*

Obviously, x is a minimal codeword if and only if $supp(x)$ is a circuit of $\mathcal{M}[H]$ and the support of a non-zero codeword of C corresponds to a cycle of $\mathcal{M}[H]$. The equivalence between binary matroids and binary linear codes implies the equivalence between the dual matroid and the dual of a binary linear code. Let $C^\perp$ denote the dual code of C. A possible parity check matrix of $C^\perp$ is G which is a generator matrix of C. Using the results from coding theory and matroid theory, it can be shown that $C^\perp$ is equivalent to $\mathcal{M}[G] = (\mathcal{M}[H])^*$. Translations between binary matroids and binary linear codes are listed in Table 3.1. All theoretical results on binary matroids given so far apply to binary linear codes. Instead of repeating the results under the coding theory context we suggest the reader to use the translation table.

In coding theory, the notions of restriction and contraction are known as shortening and puncturing, respectively. In a binary linear code shortening is performed via deletion of one or more columns from the given parity check matrix. Puncturing on the other hand is performed via deletion of one or more columns from the generator matrix of the code. Shorter codes which can be obtained from a parent code via shortening and puncturing operations are called subcodes. Subcodes correspond to matroid minors.

Some special binary matroids are introduced in Sections 3.2, 3.3, and 3.4. These matroids can directly be transferred to binary linear codes. Interestingly, the famous Fano matroid F_7 of the matroid theory corresponds to the $[7,4]$ Hamming code that we denote by H_7. The dual of the Fano matroid, F_7^*, corresponds to $[7,3]$ simplex code. Some other special matroids we use in what follows are $(\mathcal{M}[K_5])^*$, $(\mathcal{M}[K_{3,3}])^*$, and R_{10}. We denote the associated binary

Table 3.1: A translation table from matroid theory to coding theory.

Matroid theory	**Coding theory**
binary Matroid, $\mathcal{M}[H]$	binary Linear Code, C
cardinality of the ground set, $\|J\|$	block length, n
representation matrix, H	parity check matrix, H
standard form of H	systematic structure of H
isomorphism on matroids	equivalence on codes
representation matrix of $(\mathcal{M}[H])^*$, $\mathcal{M}[G]$	generator matrix, G
dual matroid, $\mathcal{M}[G]$	dual code, $C^\perp$
a cycle $\mathcal{C}$ of $\mathcal{M}[H]$	$supp(x)$, $x \in C$
a circuit $\hat{\mathcal{C}}$ of $\mathcal{M}[H]$	$supp(\hat{x})$, $\hat{x} \in C$ minimal codeword
$r(\mathcal{M}[H])$	$dim(C)$
$r(\mathcal{M}[G])$	$dim(C^\perp) = n - dim(C)$
cardinality of a minimal circuit of $\mathcal{M}[H]$	minimal distance of C
a cocycle $\mathcal{D}$ of $\mathcal{M}[H]$	codeword of $C^\perp$
triad	codeword of $C^\perp$ with weight 3
relaxation $\mathcal{Q}(\mathcal{M}[H])$	fundamental polytope (see 5.2) on $C^\perp$
cycle polytope $\mathcal{P}(\mathcal{M}[H])$	codeword polytope conv(C)
minor	subcode
fano matroid, F_7	$[7, 4]$ Hamming code, H_7
complete binary matroids	simplex codes
regular matroids	regular codes
graphic matroids	graphic codes
cographic matroids	cographic codes
matroids with sum of circuits property	geometrically perfect codes
cocircuit inequalities	FS inequalities (see Section 4.2) derived from a minimal dual codeword

linear codes by $C(K_5^{\perp})$, $C(K_{3,3}^{\perp})$, and $C(R_{10})$. Kashyap [45] derived classifications for codes similar to classification of matroids.

Definition 3.62. *A binary linear code is called regular if and only if it has no minor equivalent to H_7 or H_7^*.*

Definition 3.63. *A regular code is called graphic if and only if it has no minor equivalent to $C(K_5^{\perp})$ or $C(K_{3,3}^{\perp})$.*

Definition 3.64. *A regular code is called cographic if and only if it has no minor equivalent to $C(K_5)$ or $C(K_{3,3})$.*

The dimension and the facets are the essentials of the polyhedral analysis of the codeword polytope. Corollary 3.24 on the dimension of the cycle polytope applies to the codeword polytope, i.e., conv(C).

Corollary 3.65. *If $d(C^{\perp}) \geq 3$ then conv(C) is full-dimensional, i.e., dim(conv(C))= n.*

It is known from polyhedral theory that if conv(C) is a full-dimensional polytope then every inequality of a non-redundant system of inequalities describing the codeword polytope defines a facet. The results of Barahona and Grötschel [5] on the cycle polytope (see Section 3.3) apply to binary linear codes. Especially, Theorem 3.30 implies that in general the forbidden set (FS) inequalities (see Section 4.2) derived from all codewords do not necessarily describe conv(C) completely.

It follows from the definition of complete binary matroids that the representation matrix of a complete binary matroid L_m, $m \geq 3$, corresponds to the generator matrix of a $[2^m - 1, 2^m - 1 - m]$ Hamming code (see Definition 2.26). Alternatively a representation matrix of a complete binary matroid L_m, $m \geq 3$, corresponds to a parity check matrix of a $[2^m - 1, m]$ dual Hamming code (simplex code). Hence, the codeword polytopes of simplex codes are completely described by Theorem 3.33. By this theorem a non-zero codeword of a $[2^m - 1, m]$ dual Hamming code is a minimum codeword with weight 2^{m-1}. Since $m \geq 3$, there is no non-zero codeword in any simplex code with weight less than or equal to 3. In other words any matroid corresponding to a Hamming code has no coloop, no coparallel elements and no triad. For $m = 3$ a Hamming code has no F_7^*, $C(K_5^{\perp})$, $C(R_{10})$ subcode thus any $[7, 4]$ Hamming code can be completely described by the FS inequalities derived from all dual codewords and the box inequalities. For $m \geq 4$ however it can be shown that a Hamming code has F_7^* as subcode. We state that as a theorem and propose our own proof.

Theorem 3.66. *Any Hamming code of length $n = 2^m - 1$, $m \geq 4$ has a subcode equivalent to $\mathcal{H}_7^{\perp}$.*

Proof. To prove the statement, we show that the matroid associated with any Hamming code for $m \geq 4$ has a minor which is isomorphic to F_7^*. Let J_m be the ground set of some L_m. Since a representation matrix of any matroid isomorphic to L_4^* has all distinct binary vectors of length 4 except for the zero vector as its columns, a minor isomorphic to F_7^* can be obtained from L_4^* by restrictions only. Equivalently, there exists a subset $S \subset J_4$ such that $F_7^* \cong L_4^*|S$.

Any L_m is a contraction minor of some L_{m+1} for $m \geq 1$. Thus, for all $m \geq 4$ there exists a subset $T \subset J_{m+1}$ such that $L_m \cong L_{m+1}/T$. By Theorem 3.7 $L_m \cong L_{m+1}/T$ is equivalent to $L_m^* \cong L_{m+1}^*|T$. It follows that for $m \geq 4$ there exists a subset $S \subset J_4$ and a subset $V \subset J_m$ such that $F_7^* \cong L_4^*|S \cong L_m^*|V$. Hence, for $m \geq 4$, any matroid associated with a corresponding Hamming code has $H_7^\perp$ as a subcode. □

From the observations given above, we deduce the following and present the proofs.

Theorem 3.67. *Let C be a $[2^m - 1, 2^m - 1 - m]$ Hamming code such that $m \geq 4$. Then*

(1) The codeword polytope, conv(C), is full-dimensional,

(2) The inequalities $0 \leq x_j \leq 1$ define facets of conv(C),

(3) The FS inequalities derived from minimal dual codewords (cocircuit inequalities) do not describe conv(C) completely,

(4) L_1 is a maximal complete contraction minor of a matroid associated with C.

Proof. The proofs below follow from the previous results given in this chapter.

(1) For Hamming codes with $m \geq 3$, $d(C^\perp) \geq 3$ since a binary matroid associated with a Hamming code has no coloop, no coparallel elements. The result follows from Corollary 3.65.

(2) A Hamming code with $m \geq 3$ has no dual codeword $d \in C^\perp$, $w(d) = 3$, (no triad). The result follows from Theorem 3.26.

(3) It is shown in Theorem 3.66 that a Hamming code with $m \geq 4$, has a subcode equivalent to $\mathcal{H}_7^\perp$. The result follows from Theorem 3.28.

(4) Let $\mathcal{M}[H]$ be the binary matroid corresponding to C. From the lifting procedure explained in Section 3.3.1 it is known that any element of $\mathcal{M}[H]$ is contained in L_1. Furthermore L_1 is a contraction minor of $\mathcal{M}[H]$ if $\mathcal{M}[H]$ has no coloop and no coparallel elements. If some L_1 is contained in a triad then L_2 is also a contraction minor of $\mathcal{M}[H]$ and L_1 is not maximal. Since $\mathcal{M}[H]$ has no triad, L_1 is a maximal complete contraction minor of $\mathcal{M}[H]$.

□

It is to be expected that many codes have dual codes with minimum distance greater than three. Thus Theorem 3.67 can be extended to other code families.

Feldman et al. show in [27] that the maximum likelihood decoding problem can be formulated as an IP problem in which a linear cost function is minimized over the codeword polytope. Thus, solving the ML decoding problem on a binary linear code is equivalent to solving the cycle problem of a binary matroid. This observation enables to use matroid theoretic results known from the cycle problem in the context of coding theory. Kashyap [45] is transfers the decomposition results of Seymour [61] and Grötschel and Truemper [35] (see Section 3.4) to binary linear codes. The theoretical background required to understand these transfers are explained in detail in [39, Chapter 6]. Different from the decomposition schemes given in [35] where the representation matrices of binary matroids are used for (de)composition, Kashyap introduces (de)composition schemes based on the generator matrices of binary linear codes.

Code composition is accomplished as follows. Let $G' = \left[G'_{.,1}, \ldots, G'_{.,n'}\right]$, $G'' = \left[G''_{.,1}, \ldots, G''_{.,n''}\right]$ be the generator matrices of the binary linear codes C' and C'' respectively. The block length of C' is n' and the block length of C'' is n''. A composition over an integer m, $0 \leq 2\mathrm{m} < min\{n', n''\}$, is obtained by constructing G from G' and G'' by

$$G := \begin{bmatrix} g'_1 & \ldots & g'_{n'-\mathrm{m}+1} & g'_{n'-\mathrm{m}} & \ldots & g'_{n'} & 0 & \ldots & 0 \\ 0 & \ldots & 0 & g''_1 & \ldots & g''_{\mathrm{m}} & g''_{\mathrm{m}+1} & \ldots & g''_{n''} \end{bmatrix} \tag{3.12}$$

and contracting the m elements in G where G' and G'' overlap. The block length of the resulting code, denoted by $S_{\mathrm{m}}(C', C'')$, is $n'+n''-2\mathrm{m}$. In particular m values 0, 1, and 3 are considered. For m$=0$, $S_0(C', C'')$ is equivalent to the direct sum of binary linear codes (binary matroids). Direct sum operation is denoted by $\oplus$ in [45]. For m$=1$, $S_1(C', C'')$ is equivalent to Kashyap's 2-sum which is denoted by $\oplus_2$ and defined next.

Definition 3.68. *Let C', C'' be binary linear codes with block length $n' \geq 3$ and $n'' \geq 3$, respectively. Furthermore $(0 \ldots 01)$ is not a codeword of C' or $(C')^{\perp}$ and $(10 \ldots 0)$ is not a codeword of C'' or $(C'')^{\perp}$. Then, $S_1(C', C'') = C' \oplus_2 C''$.*

The e-sum $\oplus_e$ introduced in [35] and the 2-sum $\oplus_2$ introduced in [45] are closely related. Let C' and C'' be binary linear codes, i.e., binary matroids. The 2-sum $(C' \oplus_2 C'')$ can be defined if and only if $(C' \oplus_e C'')$ can be defined and $(C' \oplus_2 C'')$ is equivalent to $(C' \oplus_e C'')$. Both sums yield equivalent codes but via $\oplus_2$-sum, the resulting code is defined by a generator matrix and via $\oplus_e$-sum the resulting code is defined by a parity check matrix. Conversely a binary linear code C can be decomposed via $\oplus_2$-sum if and only if Theorem 3.48 holds.

Based on similar arguments used for the e-sum, Kashyap introduces a decomposition scheme for $\oplus_2$-sum. We refer to Kashyap [45] or [39] for the details of the decomposition procedure. One remark we make here is that it may be necessary to permute the elements of C in Kashyap's decomposition procedures. Let C be a binary linear code defined by a $k \times n$ generator matrix G and (T, T^C) is an exact 2-separation, C can be decomposed in polynomial time into two binary linear codes C' and C'' such that $C = \Pi(C' \oplus_2 C'')$ where Π is a permutation of the elements.

For $m = 3$, $S_3(C', C'')$ is equivalent to Kashyap's 3-sum denoted by $\oplus_3$.

Definition 3.69. *Let C', C'' be binary linear codes with block length $n' \geq 7$ and $n'' \geq 7$ respectively.*

(1) $(0 \ldots 0111)$ is a minimal codeword of C' and $(C')^\perp$, i.e., there exists no codeword whose support is entirely contained in the last three coordinates.

(2) $(1110 \ldots 0)$ is a minimal codeword of C'' and $(C'')^\perp$, i.e., there exists no codeword whose support is entirely contained in the first three coordinates.

Then, $S_3(C', C'') := C' \oplus_3 C''$.

Analogous to the relation between $\oplus_e$ and $\oplus_2$, $(C' \oplus_3 C'')$ can be defined if and only if $(C' \oplus_\Delta C'')$ can be defined. Moreover, $(C' \oplus_3 C'')$ is equivalent to $(C' \oplus_\Delta C'')$. Conversely C can be decomposed via $\oplus_3$-sum if and only if Theorem 3.49 holds. Kashyap explains how to decompose a code via $\oplus_3$-sum and $\bar{\oplus}_3$-sum which corresponds to Y-sum of [35]. We refer again [45] or [39] for the details of the decomposition procedures. Let C be a binary linear code defined by a $k \times n$ generator matrix G and let (T, T^C) be an exact 3-separation with $\min\left|T, T^C\right| \geq 4$. C can be decomposed in polynomial time into two binary linear codes C' and C'' such that $C = \Pi(C' \oplus_3 C'')$ or $C = \Pi(C' \bar{\oplus}_3 C'')$.

Decomposition of codes lead to equivalent results stated in Theorem 3.58 and Corollary 3.59. Below we repeat these results in coding theory terminology.

Theorem 3.70. *[35] Let $\mathcal{N}$ be a code class which contains the graphic codes and a finite number of codes which are not graphic. Moreover, let $\mathbb{M}$ be a code class such that for each $\mathcal{M} \in \mathbb{M} \setminus \mathcal{N}$ one can determine in polynomial time a 2-sum $C = C' \oplus_e C''$ $(C = \Pi(C' \oplus_2 C''))$ or a special 3-sum $C = C' \oplus_Y C''$ $(C = \Pi(C' \bar{\oplus}_3 C''))$ where $C' \in \mathcal{N}$ and $C'' \in \mathbb{M}$. Then there is a polynomial time combinatorial algorithm that solves the ML decoding problem for all codes in $\mathbb{M}$.*

Kashyap calls code families $\mathbb{M}$ polynomially almost graphic codes. An important example of polynomially almost graphic codes are geometrically perfect codes. Geometrically perfect codes are the codes for which the codeword polytope can be described completely by the trivial inequalties and the FS inequalities derived from all dual codewords, i.e., $\text{conv}(C) = \mathcal{Q}(C^\perp)$. In other

words geometrically perfect codes are the codes for which the associated binary matroids have the sum of circuits property. Thus geometrically perfect codes can also be defined using excluded minors.

Definition 3.71. *A binary linear code is called geometrically perfect if it has no subcode equivalent to $H_7^\perp$, $C(K_5^\perp)$ or $C(R_{10})$.*

Other examples of polynomially almost graphic codes are summarized in Corollary 3.72. These are the code families for which the maximum likelihood decoding problem can be solved in polynomial time using decomposition techniques. The solution algorithm can be found in [45] ($\oplus_2$ and $\bar{\oplus}_3$ is used) or in [35]($\oplus_e$ and $\oplus_Y$ is used).

Corollary 3.72. *ML decoding can be achieved for all codes $C \in \mathbb{M}$ if $\mathbb{M}$ is a code family such that any code in $\mathbb{M}$*

- *has no H_7 and $C(K_5^\perp)$ minor,*
- *has no $H_7^\perp$ and $C(K_5^\perp)$ minor,*
- *is cographic and has no $C(K_5^\perp)$ minor,*
- *is graphic.*

From a coding theoretic point of view, a family of error-correcting codes is not asymptotically good if the dimension or minimum distance of the codes in this family grow sublinearly with the code length. Kashyap proved that for the family of polynomially almost graphic codes either the dimension or the minimum distance grows sublinearly with the code length.

Previously we showed some structural results on Hamming codes (see Theorem 3.66, 3.67). Next we present a decomposition result on Hamming codes.

Theorem 3.73. *No Hamming Code can be decomposed via direct sum, 2-sums or 3-sums.*

Proof : Let C be a $[2^m - 1, 2^m - 1 - m]$ Hamming code such that $m \geq 3$. By the definition of a Hamming code, a parity check matrix H of C has m rows and all the non-zero m-tuples as its columns. Let $\mathcal{M}$ be the binary matroid represented by H and let J be its ground set. To prove that $\mathcal{M}$ cannot be decomposed via direct sums, it suffices to show that $\mathcal{M}$ is connected, i.e., for every pair of distinct elements of J, there exists a circuit containing both elements, see Theorem 3.44. Let e, f be two distinct elements of $\mathcal{M}$. Then the column vectors of H indexed by $\{e, f\}$, call them x, y, are linearly independent by the structure of H. Again by the structure of H the vector $z = x + y$ is a column of H. Let g by the index of this column. Then the set $\{e, f, g\}$ is a circuit of $\mathcal{M}$. Thus, $\mathcal{M}$ is connected.

By the above argument every subset of J with cardinality two has rank two and a maximal subset of J with rank two has cardinality three. By an induction argument and the structure of H it follows that any subset of J with rank l has at most cardinality $2^l - 1$.
Let (T, T^C) be a partition of the ground set J. To prove that $\mathcal{M}$ cannot be decomposed via 2-sums one has to show that the assumption

$$min\left\{|T|, \left|T^C\right|\right\} \geq 2 \quad \text{and} \quad r(T) + r(T^C) = m + 1 \tag{3.13}$$

implies $|T| + \left|T^C\right| < n$. Namely, from (3.13) it follows that $r(T), r(T^C) \leq m - 1$. Thus,

$$\begin{aligned} |T| + \left|T^C\right| &\leq 2^{r(T)} - 1 + 2^{r(T^C)} - 1 \\ &\leq 2^{m-1} - 1 + 2^{m-1} - 1 \\ &= 2^m - 2 < 2^m - 1 = n \end{aligned}$$

The proof $\mathcal{M}$ can not be decomposed via 3-sums is analogous to that for the 2-sums. □

3.6 Conclusion and further research

In this chapter we reviewed some polyhedral properties of the cycle polytope of a binary matroid, e.g., dimension, valid inequalities, characterization of facets. Due to the one-to-one correspondence between binary matroids and binary linear codes, these results are also valid for the codeword polytope. Decomposition of binary matroids has also interesting implications on MLD of binary linear codes, e.g., MLD in polynomial time is possible for certain code classes. Main results to understand decomposition of binary matroids, are also discussed in this chapter.

An interesting and ambitious further research direction is to find more structural results on decomposition of binary matroids, e.g., decomposition rules, existence of certain minors. Motivated by the decomposition of binary matroids, heuristic approaches which complete somehow the following steps :

- decompose a possible parity-check matrix into smaller pieces according to some rule,
- find components of a codeword,
- compose the components into a codeword

can be studied.

Chapter 4

Mathematical modeling for ML decoding and minimum distance calculation

4.1 Introduction

Integer programming (IP) provides powerful means for modeling several real-world problems. The maximum likelihood decoding (MLD) problem for binary linear codes has been first modeled as an integer programming problem in [9], [23], and [27]. In this chapter, we study several integer programming formulations modeling the MLD and minimum distance problems. Most of these formulations are derived from linear programming (LP) formulations which have been proposed in the LP decoding literature. Some of them are applicable to general binary linear codes while others are designed for some specific subclasses. Integer programming models are usually solved by commercial solver packages. Figure 4.1, illustrates the mathematical modeling approach we use in this chapter. It can be summarized as, model the MLD and minimum distance problems with a cost function and constraints which are linear in $\mathbb{R}^n$. Then solve the formulation with a general purpose IP solver.

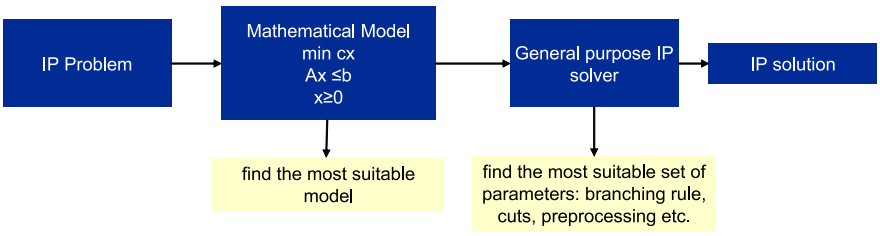


Figure 4.1: Mathematical modeling

Together with general purpose solvers, IP models allow practitioners to

study codes closely. For codes with short and medium block lengths, information about ML decoding curves or about code features like the minimum distance can be obtained without the need of implementing a code dependent algorithm. Moreover, LP decoding is improved by branch and bound approaches (see [19], [20], [76]) and cut generation algorithms (see [27], [63], [64]). In this context, the results presented in this chapter may also provide benchmarks for the approaches going from LP decoding to ML decoding.

Problem sizes that can be solved with a general purpose solver depend on the efficiency of the mathematical model and the efforts put in choosing the right parameters for the solver, e.g., branching rule, cuts. Since several different models have been formulated in the literature, the question of the capabilities of these different models arises. In this chapter, we use the IP solver of CPLEX [1] to compare the existing models numerically.

This chapter is organized as follows. In Section 4.2, we recall the different models by explaining their objective functions and sets of constraints. An analysis of the performance of these models with respect to number of variables, number of constraints, average number of simplex iterations, nodes in the branch and bound tree, cuts, and decoding time is given in Section 4.3. ML curves and minimum distances of selected LDPC, BCH, and turbo codes are also presented in this Section. In Section 4.4 we elaborate on some heursitic approaches which may be used to find initial solutions for the IP solver. Finally, we give some concluding remarks and suggest further research directions in Section 4.5.

4.2 Integer programming models

We group the integer programming formulations into 5 categories.

- Modulo operator model,
- Convex hull model,
- Forbidden sets model ,
- Parity polytope model,
- Turbo-like codes model.

In the formulations, the vector of log-likelihood ratios is denoted by $\lambda \in \mathbb{R}^n$ and a possible parity-check matrix is denoted by $H \in \{0,1\}^{m \times n}$ (see Chapter 2 for notation).

4.2.1 Modulo operator model

Parity-check equations in $GF(2)$ can be modeled in $\mathbb{R}^n$ by introducing an auxiliary variable for each row of the parity-check matrix. The auxiliary z variables are positive integers and it can easily be verified that $x \in \{0,1\}^n$ is a codeword if and only if $Hx = 2z$, $z \in \mathbb{Z}^m$. This approach is independently introduced in Breitbach et al. [9] and Tanatmis et al. [64]. Formally, IPD1 can be written as

$$\begin{aligned} &\text{minimize} && \lambda^T x && \text{(IPD1)} \\ &\text{subject to} && Hx - 2z = 0 && \\ &&& x_j \in \{0,1\} && \text{for all } j \in \{1,\dots,n\} \\ &&& z_i \geq 0, \text{ integer} && \text{for all } i \in \{1,\dots,m\}. \end{aligned}$$

In an alternative formulation, the minimum weight error vector is found. First, the received hard decision vector is calculated, i.e., for all $j \in \{1,\dots,n\}$, $\bar{y}_j = 1$ if $\lambda_j < 0$ and $\bar{y}_j = 0$ otherwise. The value $\hat{z} = \lambda^T \bar{y}$ is a lower bound for the MLD problem, i.e., $\hat{z} \leq \min\{\lambda^T x : x \in C\}$.

Second, $s = H\bar{y}$ is calculated. If $s_i = 0$ for all $i \in I$ then $\bar{y}$ is an ML codeword. Otherwise $\bar{y}$ is not a valid codeword, and it holds that $\bar{y} \equiv (x + e) \bmod 2$ where $x \in C$ and $e \in \{0,1\}^n$ is an error vector. Consequently, we have $s \equiv H(x + e) \bmod 2$ and since x is a codeword it holds that $s \equiv He \bmod 2$.

The error vector which causes the minimum deviation from $\hat{z}$ is the minimum weight error vector. The problem of finding the minimum weight error vector can be formulated as [9]

$$\begin{aligned} &\text{minimize} && \sum_{j:e_j \neq 0} |\lambda_j| \\ &\text{subject to} && He \equiv s \bmod 2. \end{aligned}$$

The constraints which are in $GF(2)$ in the above formulation can also be modeled in $\mathbb{R}^n$ by introducing auxiliary integer variables for each row of the parity-check matrix. The vector of integers is again denoted by z. Let $\left|\lambda^T\right|$ be the vector that contains the absolute values of λ, the optimal solution of $IPD2$ given below is a minimum weight error vector. A maximum likelihood (ML) codeword is given by the modulo 2 sum of the hard decision vector $\bar{y}$ and an optimal solution of IPD2.

$$\begin{aligned} &\text{minimize} && \left|\lambda^T\right| e && \text{(IPD2)} \\ &\text{subject to} && He - 2z = s && \\ &&& e_j \in \{0,1\} && \text{for all } j \in \{1,\dots,n\} \\ &&& z_i \geq 0, \text{ integer} && \text{for all } i \in \{1,\dots,m\}. \end{aligned}$$

4.2.2 Convex hull model

Another way to model parity check equations in $GF(2)$ is to use an auxiliary variable for each local codeword. Each row (check node) $i \in I$ of a parity-check matrix defines a local code C_i and local codewords are the bit sequences which satisfy the i^{th} parity-check constraint. For a valid codeword, there is one local codeword satisfying the parity-check equation for each parity-check. In an IP formulation introduced in [27], each local codeword is modeled by a help variable $w_{i,S}$.

$$
\begin{aligned}
&\text{minimize} && \lambda^T x && \text{(IPD3)} \\
&\text{subject to} && \sum_{S \in E_i} w_{i,S} = 1 && \text{for all } i \in \{1,\dots,m\} && (4.1) \\
& && x_j = \sum_{\substack{S \in E_i \\ j \in S}} w_{i,S} && \text{for all } j \in N_i, i \in \{1,\dots,m\} && (4.2) \\
& && x_j \in \{0,1\} && \text{for all } j \in \{1,\dots,n\} && (4.3) \\
& && w_{i,S} \in \{0,1\} && \text{for all } S \in E_i, i \in \{1,\dots,m\}. && (4.4)
\end{aligned}
$$

The constraints in the above formulation describe C_i for each $i \in I$. The variables x_j, $j \in \{1,\dots,n\}$ model the code bits. Each row of a possible parity check matrix is a check node of an associated Tanner graph whereas each column corresponds to a variable node (see Chapter 2). The index set of variable nodes which are incident to check node i is defined as $N_i = \{j \in J : H_{ij} = 1\}$.

Let $S \subseteq N_i$ be the index set of variable nodes which are set to 1. If $x_j = 1$ for all $j \in S$, and $x_j = 0$ for all $j \in N_i \setminus S$, then these value assignments are feasible for the local code C_i if $|S|$ is even. For $j \notin N_i$, the value of x_j can be chosen arbitrarily. The set of all subsets $S \subseteq N_i$ with even cardinality is denoted by $E_i = \{S \subseteq N_i : |S| \text{ even}\}$. Auxiliary variables $w_{i,S}$ indicate that the check node i is satisfied by the value assignment implied by S.

The idea of assigning a variable to each local codeword is also used in an LP formulation proposed in [71]. Vontobel and Kötter [71] propose an LP which leads to an efficient, message passing like decoding algorithm. Here we consider the IP version of this LP. To transform an LP to IP the boxing constraints,

e.g., $0 \leq x_j \leq 1$, are replaced by integrality constraints $x_j \in \{0, 1\}$.

$$
\begin{aligned}
&\text{minimize} && \lambda^T x && \text{(IPD4)} && \\
&\text{subject to} && x_j = u_{j,0} && \text{for all } j \in \{1, \ldots, n\} && (4.5)\\
& && u_{j,i} = v_{i,j} && \text{for all } (i,j) \in I \times J : H_{i,j} = 1 && (4.6)\\
& && u_{j,i} = \sum_{\substack{S \in A_j \\ j \in S}} \alpha_{j,S} && \text{for all } i \in N_j \cup \{0\}, j \in \{1, \ldots, n\} && (4.7)\\
& && \sum_{S \in A_j} \alpha_{j,S} = 1 && \text{for all } j \in \{1, \ldots, n\} && (4.8)\\
& && v_{i,j} = \sum_{\substack{S \in E_i \\ j \in S}} w_{i,S} && \text{for all } j \in N_i, i \in \{1, \ldots, m\} && (4.9)\\
& && \sum_{S \in E_i} w_{i,S} = 1 && \text{for all } i \in \{1, \ldots, m\} && (4.10)\\
& && \alpha_{j,S} \in \{0, 1\} && \text{for all } S \in A_j, j \in \{1, \ldots, n\} && (4.11)\\
& && w_{i,S} \in \{0, 1\} && \text{for all } S \in E_i, i\{1, \ldots, m\} && (4.12)\\
& && x_j \in \{0, 1\} && \text{for all } j \in \{1, \ldots, n\} && (4.13)\\
& && u_{j,i}, v_{i,j} \in \{0, 1\} && \text{for all } (i,j) \in I \times J : H_{i,j} = 1. && (4.14)
\end{aligned}
$$

IPD4 models a Forney-style factor graph (FFG) [28] representation of C. In an FFG, function nodes represent constraints and edges represent variables. This FFG has two groups of nodes, one group represents equality constraints and the other represents parity-check constraints. An illustration is given in Figure 4.2.2. IPD4 is very similar to IPD3 of [27]. The main difference is that variable nodes $j \in J$ define repetition codes A_j. Codewords of A_j are all-zero or all-one codewords of length $|N_j| + 1$. As in IPD3 of [27], E_i is the set of supports of all local codewords in C_i. The variables x, u, and v model the edges of this FFG.

4.2.3 Forbidden sets model

A second integer programming formulation proposed in [27] is obtained by employing the so called forbidden set (FS) inequalities (4.15) [19]. The FS inequalities are motivated by the observation that one can forbid those value assignments to variables where $|S|$ is odd. Note that these inequalities are equivalent to cocircuit inequlities described in Chapter 3. For all local codewords in C_i it holds that

$$\sum_{j \in S} x_j - \sum_{j \in N_i \setminus S} x_j \leq |S| - 1 \; \forall S \in \Sigma_i \tag{4.15}$$

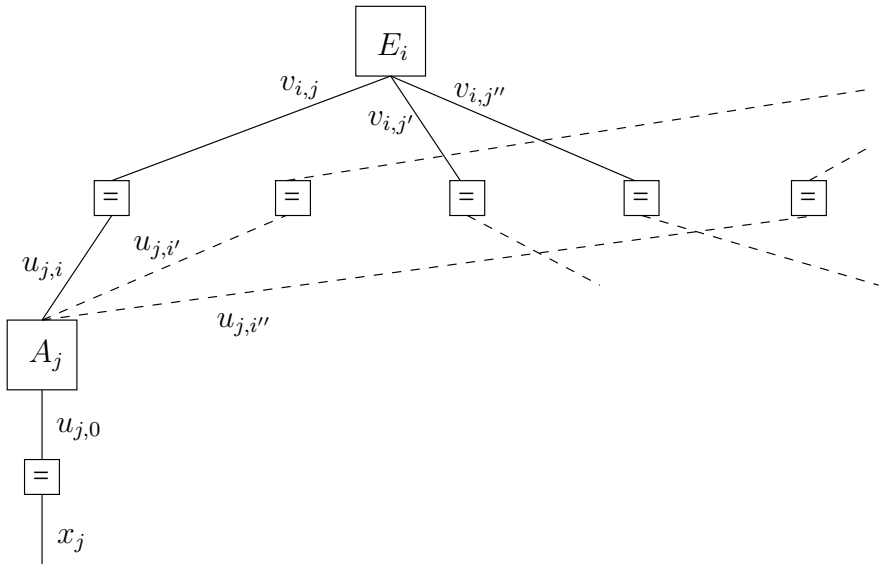


Figure 4.2: Forney-style factor graph

where $\Sigma_i = \{S \subseteq N_i : |S| \text{ odd}\}$. The integer programming formulation, IPD5, is given below.

$$\begin{array}{lll} \text{minimize} & \lambda^T x & \text{(IPD5)} \\ \text{subject to} & \displaystyle\sum_{j\in S} x_j - \sum_{j\in N_i\setminus S} x_j \leq |S| - 1 & \text{for all } S \in \Sigma_i, i \in \{1,\ldots,m\} \\ & x_j \in \{0,1\} & \text{for all } j \in \{1,\ldots,n\}. \end{array}$$

In IPD5, a constraint has to be written for each odd cardinality subset of an index set N_i. Thus, the number of constraints increases exponentially in the check node degree.

The FS inequalities used in IPD5 can also be written after decomposing the Tanner graph representation of C as shown by Yang et al. [77] and Chertkov and Stepanov [15]. Although Yang et al. [77] used the decomposition technique for their LP formulation, it can be extended to the IP version. After decomposition, an IP formulation which has size linear in the length and check node degrees can be obtained. In the approach of [77] 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 4.3 illustrates this decomposition technique: a check node with degree 4 is decomposed into 3 parity checks each with degree at most 3. The parity-check nodes are illustrated by squares. In the example, original variables are denoted by $\nu_1,\ldots,\nu_4$ while the auxiliary variable node is named ν_5. In general, this decomposition technique is iteratively applied until every check node has degree less than 4. For the details of the decomposition we refer to [77] and [15].

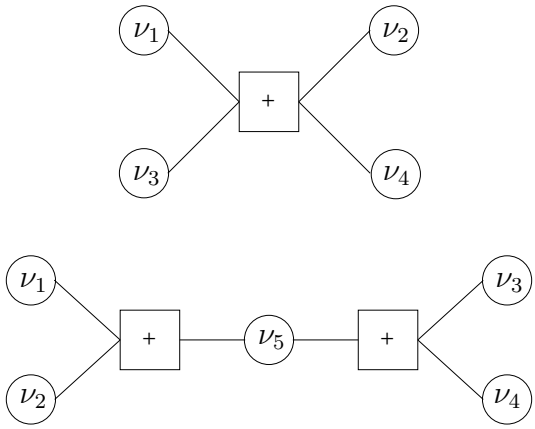


Figure 4.3: Check node decomposition.

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 $\nu_1^k + \nu_2^k + \nu_3^k \equiv 0 \bmod 2$. Note that with our notation, some of these variables might represent the same variable node, e.g., in Figure 4.3, ν_5 is adjacent to two parity checks. Yang et al. show that the parity-check constraint $\nu_1^k + \nu_2^k + \nu_3^k \equiv 0 \bmod 2$ can be replaced by the linear constraints $\nu_1^k + \nu_2^k + \nu_3^k \leq 2, \nu_1^k - \nu_2^k - \nu_3^k \leq 0, \nu_2^k - \nu_1^k - \nu_3^k \leq 0, \nu_3^k - \nu_1^k - \nu_2^k \leq 0$. If $d_c(k) = 2$ then $\nu_1^k = \nu_2^k$ models the parity-check. The constraint set of the resulting IP formulation, IPD6, is the union of all constraints modeling $|K|$ parity checks and the integrality constraints.

$$
\begin{array}{lll}
\text{minimize} & \bar{\lambda}^T \nu & \text{(IPD6)} \\
\text{subject to} & \displaystyle\sum_{j \in S} \nu_j^k - \sum_{j \in N_k \setminus S} \nu_j^k \leq |S| - 1 & \text{for all } S \in \Sigma_k, k \in \{1, \ldots, |K|\} \\
& \nu_j^k \in \{0, 1\} & \text{for all } j \in N_k, k \in K.
\end{array}
$$

In the objective function only the ν variables corresponding to the original x variables have non-zero coefficients. Thus, the objective function of IPD6 is the same as the objective function of IPD5. The constraints in IPD6 are the FS inequalities used in IPD5 with the property that the degree of the check node is less than 4.

4.2.4 Parity polytope model

As the maximum check node degree increases, the sizes of IPD3, IPD4, and IPD5 become prohibitively large. The size of the third formulation proposed in [27] increases polynomially in block length. It is based on the parity polytope of Yannakakis [78]. An index set $K_i = \{0, 2, \ldots, \left\lfloor \frac{\delta(i)}{2} \right\rfloor\}$ is defined for

$i = 1, \ldots, m$. There are two types of auxiliary variables in IPD7. The variable $p_{i,k}$ is set to one if k many variable nodes, $k \in K_i$, are set to one in the neighborhood of parity check i. Furthermore, the variable $q_{j,i,k}$ is set to one if k many variable nodes are set to one in the neighborhood of check node i and the variable node j is one of the variable nodes which are set to one.

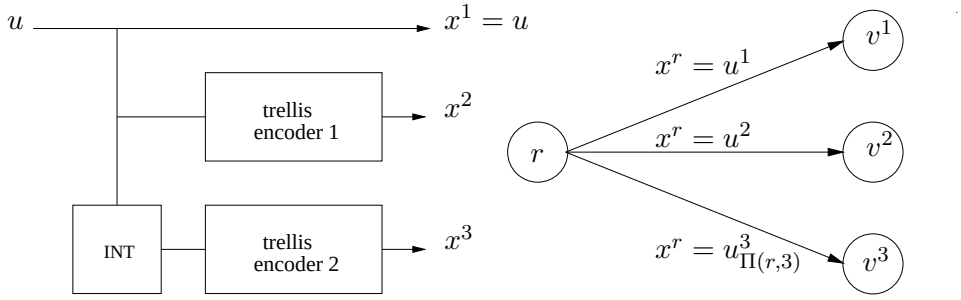
$$
\begin{array}{lll}
\text{minimize} & \lambda^T x & \text{(IPD7)} \\
\text{subject to} & x_j = \sum_{k \in K_i} q_{j,i,k} & \text{for all } j \in \{1, \ldots, n\}, i \in N_j \\
& \sum_{k \in K_i} p_{i,k} = 1 & \text{for all } i \in \{1, \ldots m\} \\
& \sum_{j \in N_i} q_{j,i,k} = k p_{i,k} & \text{for all } i \in \{1, \ldots m\}, k \in K_i \\
& q_{j,i,k} \leq p_{i,k} & \text{for all } j \in \{1, \ldots, n\}, i \in N_j, k \in K_i \\
& x_j \in \{0, 1\} & \text{for all } j \in \{1, \ldots, n\} \\
& p_{i,k} \in \{0, 1\} & \text{for all } i \in \{1, \ldots, m\}, k \in K_i \\
& q_{j,i,k} \in \{0, 1\} & \text{for all } j \in \{1, \ldots, n\}, i \in N_j, k \in K_i.
\end{array}
$$

4.2.5 Turbo-like codes model

An LP formulation for those codes which can be defined by a finite state machine is given by Feldman [23]. The last IP formulation we study in this chapter, which we refer to as IPD8, is the IP version of this LP. Particularly, IPD8 is applied to turbo codes. Any turbo code can be represented by a directed tree $T = (V, E)$ such that the nodes $v \in V$ model trellis graphs (see for example [50]). A parallel concatenated turbo code encoder and the representing directed tree is illustrated in Figures 4.4(a) and 4.4(b), respectively. It is assumed that from a root node, information word x^r is conveyed to trellises v^1, v^2. Trellis graph v^3 receives a permutation of x^r constructed along the edge $(r, 3) \in E$ and denoted by $\Pi(r, 3)$.

In a trellis graph edge labels are binary bit strings which are the outputs of the encoder for information bits. Paths from the starting node to the end node correspond to codewords. The cost of a codeword is derived from the received word and the edge labels on the path associated with this codeword. Note that in v^1, the output bits x^1 are simply set to the input bits. Trellises v^2 and v^3 output parity bits x^2 and x^3, respectively.

For each $v \in V$, the trellis graph is modeled by flow conservation, capacity constraints [2], and side constraints connecting the flow variables f^v to auxiliary variables u^v and x^v, respectively. The information bits entering trellis v are modeled by variables u^v whereas the code bits obtained in trellis v are



(a) A parallel concatenated turbo code encoder.

(b) A directed tree representing the encoder.

Figure 4.4: Encoder scheme of a parallel concatenated turbo code.

modeled by variables x^v. (see Section 7.3 for an example)

For any $v \in V$, we assume that G^v is a terminated trellis with start state $s^{\text{start},v}$ and end state $s^{\text{end},v}$. $G^v = (S^v, E^v)$ where S^v is the index set of nodes (states) and E^v is the set of edges e in G^v. The set of edges entering and leaving state $s \in S^v$ are denoted by in(s) and out(s) respectively. In the trellis G^v, feasible f^v variables satisfy flow conservation constraints

$$\sum_{e \in \text{ out}(s^{\text{start},v})} f^v_e = 1, \tag{4.16}$$

$$\sum_{e \in \text{ in}(s^{\text{end},v})} f^v_e = 1, \tag{4.17}$$

$$\sum_{e \in \text{ out}(s)} f^v_e = \sum_{e \in \text{in}(s)} f^v_e \text{ for all } s \in S^v \setminus \{s^{\text{start},v}, s^{\text{end},v}\}. \tag{4.18}$$

Some additional notation is given in [23] to connect u^v and x^v variables to flow variables f^v. In trellis layer $t \in \{1, \ldots, |u^v|\}$, the set of edges which correspond to information bit 1 is denoted by I_t. For a given code bit $j \in \{1, \ldots, |x^v|\}$, O_j denotes the set of edges where a 1 is output for the j^{th} code bit. For the formal definitions of I_t and O_j we refer to [23]. Information bits and code bits are written in terms of flows in the trellis as shown in (4.19) and (4.20)

$$u^v_t = \sum_{e \in I_t} f^v_e \text{ for all } t \in \{1, \ldots, |u^v|\}, \tag{4.19}$$

$$x^v_j = \sum_{e \in O_j} f^v_e \text{ for all } j \in \{1, \ldots, |x^v|\}. \tag{4.20}$$

The interleavers are modeled on the edges $(v, \hat{v}) \in E$ of T which connect trellises (nodes v in V). Given a permutation $\Pi\,[v, \hat{v}]$ between adjacent trellises

v and $\hat{v}$, i.e., $v, \hat{v} \in V$ such that $(v, \hat{v}) \in E$. The constraints

$$x_t^v = u^{\hat{v}}_{\Pi[v,\hat{v}](t)} \text{ for all } t \in \{1, \ldots, |x^v|\}$$

ensures that the information bits passed to trellis $\hat{v}$ are the same as the code bits output by trellis v. Thus, the interleaver consistency is guaranteed. Let C^v denote the set of (f^v, u^v, x^v) tuples which satisfy the constraints given in (4.16)-(4.20) and $L \subseteq V$ be the leaves of T. IPD8 can then be stated as

$$\begin{aligned}
&\text{minimize} && \sum_{v \in L} (\lambda^v)^T x^v && \text{(IPD8)} \\
&\text{subject to} && (f^v, u^v, x^v) \in C^v && \text{for all } v \in V && (4.21) \\
& && x_t^v = u^{\hat{v}}_{\Pi[v,\hat{v}](t)} && \text{for all } t \in \{1, \ldots, |x^v|\}, (v, \hat{v}) \in E && (4.22) \\
& && (f^v, u^v, x^v) \in \{0,1\}^{|E^v|+|u^v|+|x^v|} && \text{for all } v \in V && (4.23)
\end{aligned}$$

where λ^v is the vector of log likelihood ratios corresponding to code bits obtained from trellis $v \in L$.

The general scheme of IPD8 can easily be tailored to codes which can be defined by a finite state machine. We apply it to 3GPP Long Term Evolution (LTE) codes[1] with the encoder illustrated in Figure 4.4(a). A detailed description of the formulation can be found in Chapter 7.

4.2.6 Complexities of formulations

Sizes of different IP formulations in terms of memory complexity are given in Table 4.1. Note that integrality constraints are not included. In IPD8, the formulation size depends on the number of states in component trellises. We denote the number of states in the trellis containing the maximum number of states by Φ. In the formulation IPD4 the number of constraints is in $O(\kappa)$ where $\kappa = \max\{nd_v^{max}, md_c^{max}\}$.

Proposition 4.1. *The number of variables in IPD3, IPD4, and the number of constraints in IPD5 grow exponentially in d_c^{max}.*

4.2.7 IP formulation for minimum distance computation

Slightly modified versions of the formulations above can be used to compute the minimum distances of codes. Setting λ to the all-ones row vector $\underline{1}$ and adding the constraint $\sum_{j \in J} x_j \geq 1$ to the set of constraints, it is ensured that a non-zero codeword with minimum weight is found. To demonstrate this we choose the formulation IPD1. We denote the resulting IP model by IPMD.

[1] 3rd generation partnership project, http://www.3gpp.org

Table 4.1: A comparison of the sizes (number of variables, number of constraints) of different IP formulations in O notation.

Formulation	**variables**	**constraints**
IPD1 (Breitbach et al. [9])	$O(n)$	$O(m)$
IPD2 (Breitbach et al. [9])	$O(n)$	$O(m)$
IPD3 (Feldman et al. [27])	$O(m2^{d_c^{max}})$	$O(md_c^{max})$
IPD4 (Vontobel et al. [71])	$O(m2^{d_c^{max}})$	$O(\kappa)$
IPD5 (Feldman et al. [27])	$O(n)$	$O(m2^{d_c^{max}})$
IPD6 (Yang et al. [77])	$O(2md_c^{max} - 3m)$	$O(4md_c^{max} - 8m)$
IPD7 (Feldman et al. [27])	$O(n^3)$	$O(n^3)$
IPD8 (Feldman [23])	$O(n\Phi)$	$O(n\Phi)$

$$
\begin{aligned}
&\text{minimize } \underline{1}x && \text{(IPMD)}\\
&\text{subject to } Hx - 2z = 0\\
&\qquad \sum_{j\in J} x_j \geq 1\\
&\qquad x_j \in \{0,1\} && \text{for all } j \in \{1,\ldots,n\}\\
&\qquad z_i \geq 0, \text{integer} && \text{for all } i \in \{1,\ldots,m\}.
\end{aligned}
$$

Proposition 4.2. *IPMD finds the minimum distance of a binary linear code.*

Proof. The minimum distance of a binary linear code is $d(C) = \min\{\mathrm{w}(x) : x \in C, x \neq 0\}$. Note that $\mathrm{w}(x) = \sum_{j\in J} x_j = \underline{1}x$. The constraints $x \in C$ and $x \neq 0$ can be modeled with $Hx = 2z, (x, z) \in \mathbb{Z}^{n+m}$ and $\sum_{j\in J} x_j \geq 1$ respectively. □

4.3 Numerical results

4.3.1 Comparison of IP formulations

In this section we compare the aforementioned formulations numerically using CPLEX [1] version 12. CPLEX IP solver employs an LP based branch and cut procedure to solve IP problems. In the following we briefly describe a generic branch and cut algorithm. For details [55] is referred. Initially, i.e., in the root node of the branch and bound tree, the integrality constraints of the given IP problem are relaxed and the resulting LP is solved, e.g., by a variant of the simplex method [55]. If an integer solution is found, then the solver outputs the LP optimum as the solution since it is the IP optimum as well. If the optimal solution is non-integral, then the solver tightens the polytope of feasible solutions by generic cut generation techniques, e.g., mixed integer

rounding cuts or Gomory fractional cuts (see [1]). Cuts found are added to the LP formulation and the resulting LP problem is solved again. If the LP optimum remains non-integral and no further cuts can be found, then the solver branches on a variable, say x_i, having a non-integral value, say v, in the current solution. Branching is achieved by adding constraints on this variable, thus producing two new subproblems referred to as child nodes of the current node in the branch and bound tree. For example, one subproblem is obtained by adding the constraint $x_i \leq \lfloor v \rfloor$ to the original formulation while the second is obtained by adding $x_i \geq \lceil v \rceil$. By this technique, the search space is implicitly enumerated. Several fathoming rules based on lower and upper bounds of subproblems are established for pruning branches in the search tree, thus speeding up the computation. Repeating this procedure iteratively to all child nodes of the search tree yields the IP optimum eventually.

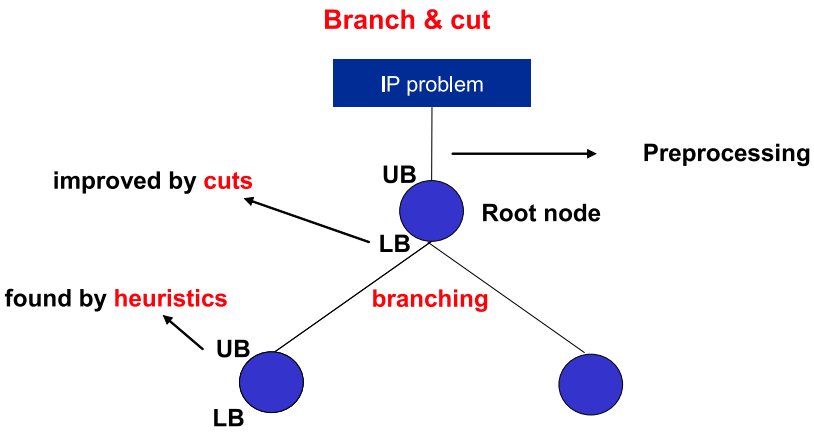


Figure 4.5: The branch and cut approach.

We compare the performances of IPD formulations as ML decoders on:

- $(3, 6)$-regular LDPC codes with $n \in \{100, 200\}$;
- $(63, 24)$, $(127, 28)$ BCH codes;
- $(132, 40)$, $(180, 56)$ LTE turbo codes;

in our numerical experiments. The BIAWGNC is assumed to be the transmission channel and the zero codeword is transmitted. Computations are executed on a compute server equipped with a Dual AMD Opteron 242 1.60GHz and 2 GB RAM, running under Linux Kernel 2.6.16 SMP x86_64.

IPD1 and IPD2 presented in Section 4.2.1 perform computationally very similar in terms of average CPU times and IP performance measures like average number of cuts, etc. Hence we only report statistics collected for IPD1.

Likewise the performance of IPD3 and IPD4 presented in Section 4.2.2 is very similar in the test instances. It should be noted that the formulation IPD4 contain some auxiliary variables and constraints which facilitate the LP approximation approach used in [71]. If the ML decoding problem is modeled via IPD4, the presolver and aggregator [1] functions of the CPLEX IP solver reduces the size of the initial problem to the size of the initial problem of IPD3. Therefore we only report statistics collected for IPD3.

For LDPC codes, formulations IPD1, IPD3, IPD5, IPD6, and IPD7 are compared. IPD8 is applied specially to LTE turbo codes. For BCH codes the formulations IPD1, IPD6, and IPD7 are compared since for codes with dense parity-check matrices, building models IPD3, IPD4, and IPD5 is computationally inefficient. For decoding of LTE turbo codes IPD1, IPD6, IPD7, and IPD8 are considered. This is because the parity check matrices we used for these codes have a dense structure. IPD1, IPD6, and IPD7 are models derived from a parity check matrix.

In the tables where statistics are presented, the sign $-$ is used to indicate that the corresponding formulation is not considered for the given code. For each SNR value, 100 received words are simulated. IPD formulations are tested under these received words, i.e., for each test SNR, the same set of received words are used in each formulation. An experimental study showed that increasing the number of instances to 1000 or 10000 does not lead to more accurate statistics.

Running time is measured after building the model and therefore only the computation time consumed by the solver is considered. Depending on the density of the given parity check matrix, block length, and SNR value running times of the IP solver for some formulations become prohibitively large. In those cases we consider only formulations where 100 instances can be solved in 14400 seconds (4 hours) of CPU time. The $*$ sign in the subsequent report tables means that 100 instances could not be solved in the predefined time limit for the corresponding formulation.

The performance of the IP solver under high, medium, and low SNR values is investigated. In the high SNR regime we are able to collect statistics for different IP formulations since decoding times are relatively small. It should be noted that at high SNR, ML decoding of codes with block lengths larger than the codes used in this chapter can be achieved. As the SNR values are decreased, the decoding times increase and often the time limit is reached before 100 instances are decoded. Initially, the SNR$= \frac{E_b}{N_0}$ is set to a value for which the frame error rate (FER) is about 0.1%. It is then decreased twice in steps of 0.8 dB.

The performance measures we use for the comparison of IP formulations are the number of variables, number of constraints, average number of simplex iterations, average number of cuts found by the IP solver, average number of nodes in the branch and bound tree, and the average CPU time measured in

seconds.

4.3.2 The number of variables and constraints

To demonstrate the sizes of the IP formulations described in Section 4.2 numerically, we report the number of variables and constraints used in IPD1-IPD8 in Table 4.2. As it can be concluded from Proposition 4.1 the number of variables in IPD3, IPD4 and the number of constraints in IPD5 are comparatively large. In IPD7, the number of variables and constraints grow polynomially but in $O(n^3)$ which also results in a fairly large number especially when block length increases.

For LDPC codes, IPD3 has many variables but the number of constraints is not large. Conversely, based on the idea of forbidding odd cardinality subsets of N_i with inequalities (see Section 4.2), IPD5 has many constraints and only few variables. On the other hand, formulation IPD7 has large number of variables and constraints.

Generally, it is observed that the running time of the IP solver increases with the number of variables and constraints of IPD formulations for ML decoding of LDPC and BCH codes. This is due to the increasing size of the search space and the increasing computational effort required to solve LP problems. A large number of variables and constraints leads to a complicated polytope and increases the size of the simplex tableau.

For LTE turbo codes however, IPD8 performed the best in terms of running time although it has more variables and constraints than IPD1 and IPD2. This is due to the fact that IPD8 has a tighter LP relaxation than the formulations based on the parity check representation of the code (see [23, Chapter 6]). Moreover, the CPLEX IP solver can detect and handle the network flow type of constraints in IPD8 efficiently. This also has a positive effect on the decoding time.

The sizes of the formulations IPD3, IPD4, and IPD5 become prohibitively large for dense parity-check matrices. In contrast, CPLEX was able to build models for IPD1, IPD2, IPD6, and IPD7 in all codes we tested. Moreover, the number of constraints and variables grow linearly in block length in IPD1, IPD2, and IPD6.

4.3.3 The average number of nodes

The average number of nodes which occurred in the search tree are listed in Table 4.3. This number corresponds to the number of subproblems which have been generated in the branch and cut procedure and affects the memory requirement. A low average number of nodes indicates that branching occurs only a few times. This is observed for codes with short block lengths or for

Table 4.2: The number of variables and constraints for selected codes.

(100,50) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	150	1700	100	250	1500	-
Constraints	50	350	1600	800	1750	-
(200,100) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	300	3400	200	500	3000	-
Constraints	100	700	3200	1600	3500	-
(63,39) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	87	-	-	552	6761	-
Constraints	24	-	-	1941	7246	-
(127,28) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	155	-	-	1445	35823	-
Constraints	28	-	-	5282	37092	-
(132,40) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	224	-	-	1394	14690	1372
Constraints	92	-	-	4966	16038	808
(180,56) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
Variables	304	-	-	2524	33197	1932
Constraints	124	-	-	9056	35584	1128

high SNR. Increasing the block length leads to larger IP problems which have to be divided into smaller, manageable problems in the branch and bound tree, i.e., the number of nodes increases. For low SNR values, the objective function coefficients become less reliable and, thus, the IP problem becomes harder to solve which leads to an increased number of nodes.

For the LDPC code with block length 100, the average number of nodes are close to each other in IPD formulations. If the block length is doubled, the increase in the average number of nodes in IPD1 and IPD2 exceeds the increase in formulations IPD3 to IPD7. For the BCH codes, the average number of nodes in IPD6 and IPD7 is larger than the same number in IPD1 and IPD2 by a factor of more than 10. For the LTE turbo codes, the average number of nodes needed for IPD8 is far less than the average number of nodes needed for IPD1, IPD2, IPD6, and IPD7. The reason for that is, the network flow constraints in IPD8 provide a tighter relaxation compared to IPD6 and IPD7 as well as IPD1 and IPD2. Hence, less branching constraints are added to IPD8 during branch and cut.

4.3.4 The average number of cuts

Table 4.4 shows the average number of cuts found by CPLEX. The IP solver tightens the LP relaxation by cut generation algorithms if necessary. As a consequence, non-integral optimal vertices of the relaxed polytope are cut off and the chance that an integral optimum is found in the subsequent iterations increases. For all codes and SNR values, formulations IPD1 and IPD2 have the largest average number of cuts. The IP solver adds significantly (by one, two, or three orders of magnitude - depending on the SNR value, block length, and the density of the given parity check matrix) less cuts in formulations IPD3 to IPD8 than in IPD1 or IPD2. This is reasonable since formulations IPD3 to IPD8 have tighter LP relaxations.

4.3.5 The average number of simplex iteartions

The simplex algorithm moves from vertex to vertex on the underlying polytope until the vertex corresponding to the optimal solution is reached. If there exists a vertex for which the objective function can be improved in the current step, the simplex algorithm moves to this vertex and otherwise the algorithm stops. The procedure of moving from one vertex to another is one simplex iteration. The average number of simplex iterations are reported in Table 4.5. A large number of variables or constraints seem to lead to polytopes with a large number of vertices and facets in our experiments. As a result, more simplex iterations are done.

In the formulations IPD3, IPD4, IPD6, IPD7, and IPD8 the number of vari-

Table 4.3: The average number of nodes.

(100,50) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	1.48	0.41	0.65	0.64	0.59	-
SNR: 2dB	26.92	19.82	27.17	32.71	27.63	-
SNR: 1.2dB	167.56	96.99	79.66	113.62	153.24	-
(200,100) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	2.55	0.3	0.49	0.5	0.61	-
SNR: 2dB	1191.42	677.21	731.24	782.76	620.3	-
SNR: 1.2dB	8129.26	2575.19	2153.41	3881.64	*	-
(63,39) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	29.6	-	-	213.25	125.47	-
SNR: 3.0dB	55.05	-	-	562.09	*	-
SNR: 2.2dB	92.57	-	-	31336.13	*	-
(127,28) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	132.36	-	-	*	*	-
SNR: 3.0 dB	2672.07	-	-	*	*	-
SNR: 2.2 dB	19667.8	-	-	*	*	-
(132,40) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	13	-	-	*	*	0.22
SNR: 1.8dB	622.83	-	-	*	*	14.16
SNR: 1.0dB	25489	-	-	*	*	89.46
(180,56) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	10.41	-	-	*	*	0
SNR: 1.8dB	387.75	-	-	*	*	5.56
SNR: 1.0dB	*	-	-	*	*	307.66

Table 4.4: The average number of cuts.

(100,50) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	39.76	0	0.13	0.29	0.04	-
SNR: 2dB	60.81	0.09	0.69	1.27	0.55	-
SNR: 1.2dB	88.21	0.42	1.61	3.04	1.14	-
(200,100) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	84.15	0	0.03	0.14	0.02	-
SNR: 2dB	126.71	0.16	0.41	0.7	0.13	-
SNR: 1.2dB	266.39	1.53	2.65	3.45	*	-
(63,39) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	7.82	-	-	5.15	0.66	-
SNR: 3.0dB	13.42	-	-	6.58	*	-
SNR: 2.2dB	25.34	-	-	8.73	*	-
(127,28) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	6.06	-	-	*	*	-
SNR: 3.0 dB	16.77	-	-	*	*	-
SNR: 2.2 dB	26.83	-	-	*	*	-
(132,40) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	77.28	-	-	*	*	0.09
SNR: 1.8dB	111.4	-	-	*	*	0.48
SNR: 1.0dB	171.5	-	-	*	*	1.26
(180,56) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	95.8	-	-	*	*	0
SNR: 1.8dB	169.26	-	-	*	*	0.45
SNR: 1.0dB	*	-	-	*	*	1.7

ables is larger than in formulations IPD1 and IPD2. Similarly, the number of constraints is larger in IPD5 than it is in IPD1 and IPD2. On the other hand, the relaxed polytopes of IPD1 and IPD2 have either a lower dimension or less number of constraints. Less simplex iterations are needed on the initial LP. However, as the block length increases, the number of cuts and branching constraints added in IPD1 and IPD2 increases as well. During the process of generating cuts and resolving LPs, a larger number of LPs are solved and, consequently, the number of simplex iterations also increases. For the LDPC codes at low SNR, IPD5 has the smallest average number of iterations. An explanation for that is, IPD5 has the smallest number of variables. For the BCH codes, far less (two or three orders of magnitude) iterations are needed in IPD1 and IPD2 than IPD6 and IPD7. It can be seen in Tables 4.3 and 4.4 that much less cuts and branching constraints are added to IPD8 than to IPD1 or IPD2. Consequently, less simplex iterations are performed in IPD8.

4.3.6 The average running times

Finally, we analyze the average running times of the IP solver for each formulation. The CPU times in seconds are listed in Table 4.6. For the LDPC codes, average decoding times are close to each other at high SNR or short block lengths. Especially at low SNR values IPD1 performs best in terms of average decoding time. For the BCH codes, the running times of IPD1 and IPD2 are significantly lower (two or three orders of magnitude) than those of IPD6 and IPD7. As the block length increases, the running times of IPD6 and IPD7 becomes prohibitively large. For the LTE turbo codes, IPD8 has the best avergae decoding times. For $n = 228$, SNR$= 1.0$dB, it is the only formulation where 100 instances can be decoded within the time limit.

4.3.7 Summary

To sum it up, the IP formulations can be grouped into the following categories.

1. The formulations IPD1 and IPD2 are constructed with less variables and less constraints. Their LP relaxations are not tight. Hence, more cuts and branching constraints are generated when traversing the search tree.

2. The formulations IPD3 to IPD7 have tight LP relaxations with relatively large numbers of variables and constraints. This leads to more complicated polytopes and larger simplex tableaus. Simplex iterations need considerable effort. On the other hand, less number of cuts have to be added to tighten the LP relaxation.

3. For codes which can be defined by a finite state machine, IPD8 performs best in terms of decoding time. IPD8 has a tighter LP relaxation than

Table 4.5: The average number of simplex iterations.

(100,50) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	33.33	129.99	43.89	141.28	110.65	-
SNR: 2dB	788.89	1039.68	736.5	1595.42	1678.31	-
SNR: 1.2dB	4828.01	4726.1	3287.82	7569.2	7782.69	-
(200,100) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	120.2	241.83	73.39	260.29	223.66	-
SNR: 2dB	42354.9	58145.7	40203.7	64727.3	63733.5	-
SNR: 1.2dB	312908	221514	110660	249259	*	-
(63,39) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	86.13	-	-	19508.9	57764	-
SNR: 3.0dB	206.47	-	-	50788.3	*	-
SNR: 2.2dB	417.78	-	-	274295	*	-
(127,28) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	629.43	-	-	*	*	-
SNR: 3.0 dB	14661	-	-	*	*	-
SNR: 2.2 dB	117074	-	-	*	*	-
(132,40) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	1483.73	-	-	*	*	862.68
SNR: 1.8dB	29679.2	-	-	*	*	2725.8
SNR: 1.0dB	533959	-	-	*	*	11349.4
(180,56) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	2723.88	-	-	*	*	1071.39
SNR: 1.8dB	24399.3	-	-	*	*	2758.77
SNR: 1.0dB	*	-	-	*	*	48911.3

Table 4.6: The average decoding times in CPU seconds.

(100,50) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	0.02	0.03	0.04	0.02	0.06	-
SNR: 2dB	0.13	0.23	0.36	0.36	0.58	-
SNR: 1.2dB	0.73	1.16	1.34	1.80	2.74	-
(200,100) LDPC code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.8dB	0.04	0.06	0.08	0.05	0.13	-
SNR: 2dB	10.6	21.79	29.92	25.17	34.08	-
SNR: 1.2dB	76.25	93.70	85.99	125.55	*	-
(63,39) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	0.03	-	-	4.99	40.58	-
SNR: 3.0dB	0.05	-	-	12.58	*	-
SNR: 2.2dB	0.08	-	-	65.98	*	-
(127,28) BCH code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 3.8dB	0.13	-	-	*	*	-
SNR: 3.0 dB	1.54	-	-	*	*	-
SNR: 2.2 dB	14.25	-	-	*	*	-
(132,40) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	0.47	-	-	*	*	0.12
SNR: 1.8dB	8.33	-	-	*	*	0.9
SNR: 1.0dB	103.7	-	-	*	*	3.82
(180,56) LTE turbo code						
Formulation	IPD1	IPD3	IPD5	IPD6	IPD7	IPD8
SNR: 2.6dB	1.18	-	-	*	*	0.17
SNR: 1.8dB	10.95	-	-	*	*	1.00
SNR: 1.0dB	*	-	-	*	*	23.84

the formulations based on the parity check representation of the code. Moreover, the CPLEX IP solver can detect and handle the network flow type of constraints in IPD8 efficiently. This also has a positive effect on the decoding time.

Based on our tests we conjecture that for LDPC and BCH codes, it is favorable to start with a formulation with few variables and constraints and iteratively generate cuts in regions of the relaxed polytope lying in the minimization direction. For codes which can be defined by a finite state machine, IPD8 can be preferred.

4.3.8 Maximum likelihood decoding via integer programming

The ML curves of the selected LDPC, BCH, and LTE turbo codes are presented below. These curves are obtained by solving IPD1 for LDPC, BCH codes and IPD8 for LTE turbo codes, with the CPLEX IP solver. The formulations IPD1 and IPD8 are selected since they performed best in terms of decoding time in the tests reported in Section 4.3. Signal to noise ratio (SNR) is measured as E_b/N_0. The frame error rates (FER) are calculated by counting 100 erroneous blocks. The ML curves are shown in Figures 4.6, 4.7, and 4.8.

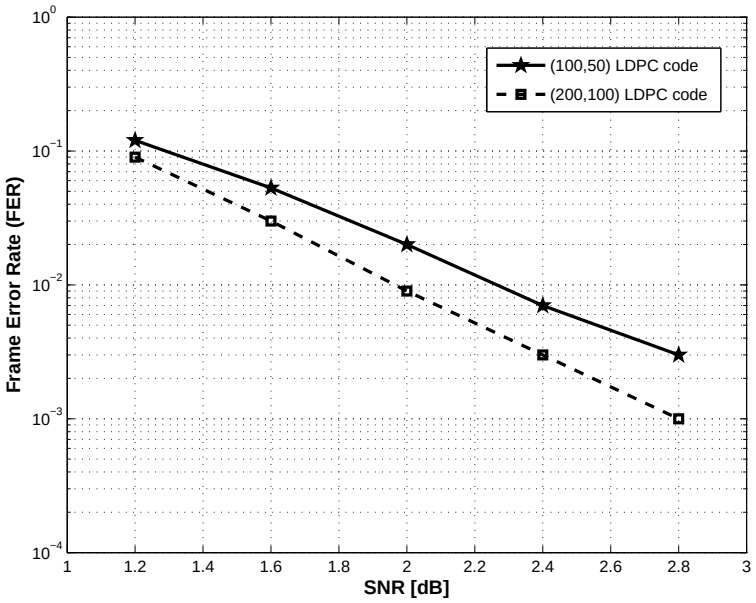


Figure 4.6: ML curves for $(3, 6)$-regular LDPC codes with $n = 100, 200$.

Under ML decoding, the error correcting performance of a code can be increased by increasing the block length or decreasing the code rate, i.e., more

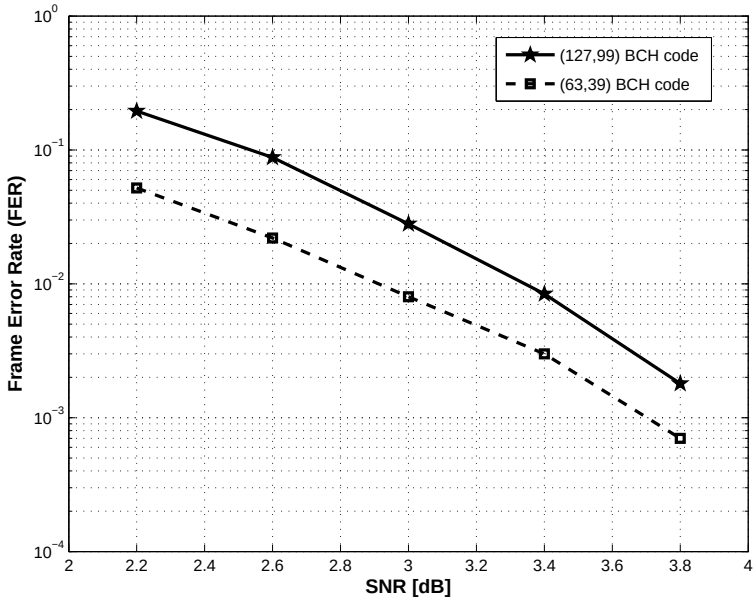


Figure 4.7: ML curves for $(63, 39)$ BCH, $(127, 99)$ BCH codes.

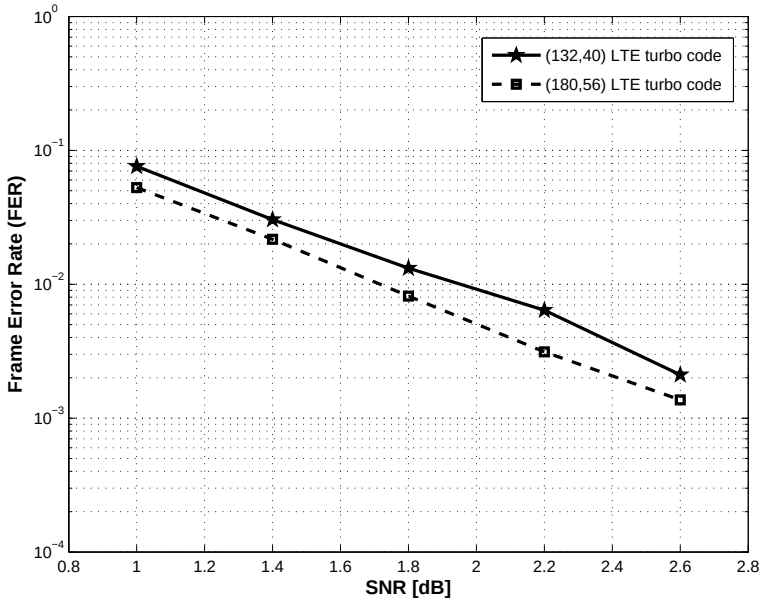


Figure 4.8: $(132, 40)$ LTE turbo, $(180, 56)$ LTE turbo codes.

parity bits are added. The LDPC codes and turbo codes discussed in this section have rate $\frac{1}{2}$ and $\frac{1}{3}$(approximately), respectively. Increasing the block length improves the error correcting performance of codes. This is demonstrated in Figures 4.6 and 4.8. The selected BCH codes have different block lengths as well as rates given by $\frac{k}{n}$. The error correcting performances are shown in Figure 4.7.

4.3.9 Computing the minimum distance via integer programming

The exact minimum distances (or upper bounds thereof) of some LDPC codes and turbo codes are computed by solving an appropriate formulation with the CPLEX IP solver. If the CPLEX IP solver returns the optimal solution then it is provably the minimum distance of the given code. If the IP solver can't find an optimal solution because of time and memory restrictions, it returns the best feasible solution found which is an upper bound on the minimum distance. The minimum distances (upper bounds) of some LDPC codes are listed in Table 4.7. For LDPC codes, the formulation IPMD is used.

Table 4.7: Minimum distances (upper bounds) of selected LDPC Codes.

Code	**Minimum distance**	**Upper bound**
(204,102) LDPC [51]	8	
(576,288) WiMax LDPC [41]	13	
(1152,576) WiMax LDPC [41]		30
(648,324) WiFi LDPC [40]		15
(1200,600) WiMedia LDPC[2]		33

Various approaches have been proposed in the literature to compute the exact minimum distances (upper bounds thereof) in the literature (see [30] and references therein). In [56], Nimbalker et al. reported estimates on minimum distances of LTE turbo codes. These results are upper bounds obtained by the combination of single and double error impulse method [7], [17]. We model the minimum distance problem as an IP and solve it with CPLEX IP solver. Instead of using IPMD which can be used for any binary linear code, we modify IPD8 (applied to LTE turbo codes see Figure 7.1) by setting the vector of objective function coefficients to all-ones vectors and adding the constraint $\sum_{j\in J} x_j \geq 1$ to the constraint set. We were able to compute the exact minimum distances of LTE turbo codes up to block length approximately 1500. For larger block lengths we present upper bounds. In Table 4.8, the exact minimum distances or upper bounds (found by CPLEX IP solver) for some selected LTE turbo codes are compared with the upper bounds reported in [56]. It is observed that

in general our mathematical modeling method improves the bounds given in [56].

Table 4.8: Minimum distances (upper bounds) for selected LTE turbo codes.

Block length	Minimum distance	Upper bound	Upper bound from [56]
132	11		17
876	23		36
2124		23	48
3468		31	49
9612		29	50
12684		33	50
16524		41	58

The importance of choosing an appropriate formulation for a given code family is demonstrated in Table 4.9. The computation times required to solve the modified IPD8 is compared with the computation times required to solve IPMD for several LTE turbo codes. It is observed that as the block length increases the difference between the computation times (in CPU seconds) of modified IPD8 and IPMD increase. Observe that for $n = 276$ the computation time of modified IPD8 is less than the computation time of IPMD by a factor of 10. Hence, modified IPD8 can tackle the minimum distance problem for LTE turbo codes more efficiently than the formulation IPMD.

Table 4.9: A comparison of solution times (in CPU seconds) of modified IPD8 and IPMD for several LTE turbo codes.

Block length	Minimum distance	Modified IPD8 (CPU sec)	IPMD (CPU sec)
180	13	38.1	70.75
228	15	132.56	842.17
276	15	143.23	1283.23

4.4 Finding initial solutions for the IP solver

In this section we describe some polynomial time heuristics to find estimates of a maximum likelihood codeword or a codeword with minimum weight. Passing such estimates to the CPLEX IP solver as initial solutions may reduce the total computation time. Heuristic 1 and heuristic 2 proposed in this section are used to estimate an ML codeword whereas heuristic 3 finds an estimate of

a minimum weight codeword. In all heuristics, it is assumed that the given parity-check matrix H has full row rank.

The basic idea of heuristic 1 is to reset the values of least reliable bits so that the parity check equations are satisfied. To this aim we propose a procedure by which the original parity check matrix H is transformed into an equivalent alternative parity check matrix $\bar{H}$ by elementary row operations in $GF(2)$. Our aim is to obtain m unit vectors in columns of H corresponding to the least reliable bits. Given the log-likelihood ratio vector $\lambda \in \mathbb{R}^n$, let Π be a permutation of $J = \{1, \ldots, n\}$ such that $\left|\lambda_{\Pi(1)}\right| \geq \left|\lambda_{\Pi(2)}\right| \geq \ldots \geq \left|\lambda_{\Pi(n)}\right|$ and $\Omega \subset J$. For $\Pi(n) \in J$ a unit vector can be obtained in column $H_{.,\Pi(n)}$. At the first step of the algorithm, Ω has one element $\Pi(n)$. At the following steps the index set Ω is extended with columns $j \in J \setminus \Omega$ such that:

- $H_{.,j}$ and columns with indices in Ω are linearly independent,
- $|\lambda_j|$ is minimum.

Such a column j exists if $|\Omega| < m$ since H has full row rank. This procedure is presented as an algorithm below. It is referred to as the find alternative parity-check matrix algorithm.

Find alternative parity-check matrix algorithm
Input: $H \in \{0,1\}^{m \times n}$ and $\{\Pi(1), \ldots, \Pi(n)\}$.
Output: An alternative parity check matrix $\bar{H}$.

1: $\Omega = \{\Pi(n)\}$.
2: Obtain first unit vector in column $H_{.,\Pi(n)}$.
3: Set $k = 1$.
4: **while** $|\Omega| < m$ **do**
5: **if** $H_{.,\Pi(n-k)}$ and $H_{.,\omega}$, are linearly independent for all $\omega \in \Omega$ **then**
6: $\Omega = \Omega \cup \Pi(n-k)$
7: Obtain next unit vector in column $H_{.,\Pi(n-k)}$
8: **end if**
9: $k = k + 1$
10: **end while**

For the codes C where the minimum distance, $d(C)$, is known, Ω can be set to $\{\Pi(n), \ldots, \Pi(n - d(C) + 1)\}$ at the beginning. If $\{\Pi(n-m+1), \ldots, \Pi(n)\} = \{n-m+1, \ldots, n\}$ then no change has to be done on the original parity check matrix H.

Assuming $\lambda_j \neq 0$ for $j \in J$, we construct the output of heuristic 1 as follows. For $j \in J \setminus \Omega$ if $\lambda_j < 0$ we set $x_j = 1$ otherwise $x_j = 0$. Note that in each row i there exists exactly one $\omega \in \Omega$ such that $H_{i,\omega} = 1$. We denote this index

Heuristic 1
Input: LLR vector $\lambda \in \mathbb{R}^n$, $H \in \{0,1\}^{m \times n}$, and $\{\Pi(1), \ldots, \Pi(n)\}$.
Output: A feasible solution, $x \in C$.

1: Call find alternative parity check matrix algorithm with H and $\Pi(n)$.
2: **for all** $j \in J \setminus \Omega$ **do**
3: **if** $\lambda_j < 0$ **then**
4: Set $x_j = 1$.
5: **else**
6: Set $x_j = 0$.
7: **end if**
8: **end for**
9: **for all** $i \in I$ **do**
10: Find index $\omega(i)$.
11: Set $K_i = \sum_{j \in N_i \setminus \omega(i)} \bar{H}_{i,j} x_j$.
12: **if** K_i is even **then**
13: Set $x_{\omega(i)} = 0$.
14: **else**
15: Set $x_{\omega(i)} = 1$.
16: **end if**
17: **end for**

with $\omega(i)$, i.e., $\omega(i) = N_i \cap \Omega$. Then for each row $i \in I$ we calculate $K_i = \sum_{j \in N_i \setminus \omega(i)} H_{i,j}$. If K_i is even then $x_{\omega(i)}$ is set to zero. Otherwise $x_{\omega(i)} = 1$.

Proposition 4.3. *Heuristic 1 outputs a codeword.*

Proof. If K_i, $i \in I$ computed in Step 11 of heurisic 1 is an even number, then the i-th parity check is satisfied. Thus $x_{\omega(i)}$ is set to 0. If K_i is an odd number, then $x_{\omega(i)}$ is set to 1. Setting $x_{\omega(i)}$ to 1 has no effect on other parity checks since $H_{.,\omega(i)} = 0$ for all $i \in I \setminus \omega(i)$. In this way it is ensured that $Hx \equiv 0 \bmod 2$ holds. □

In heuristic 2, we search for a minimum weight error vector (see Section 4.2.1). The hard decision vector $\bar{y}$ and the vector s are found as in the description of IPD2 in Section 4.2.1. The original parity check matrix is modified so that unit vectors are obtained in columns with indicies in $\Omega \subset J$. The index set $\Omega = \{\omega(1), \ldots, \omega(m)\}$ is found with the find alternative parity-check matrix algorithm. Note that while modifying a parity check matrix H, the vector s has to be modified also. We denote the modified s vector by $\bar{s}$. As a result we have $\bar{s} \equiv \bar{H}\bar{y} \bmod 2$. Finally, we go through the columns of $H_{.,j}$ for all $j \in J$.

Proposition 4.4. *For each column of an alternative parity check matrix $\bar{H}$, an error vector e such that $\bar{y} + e \equiv x \bmod 2$, $x \in C$ can be constructed.*

Proof. The error vector found for column $j \in J$ is constructed by setting $e_j = 1$ and $e_{\omega(i)} = 1$, such that $\bar{H}_{i,j} \neq \bar{s}_i$, $i \in I$. In this way it is ensured that $\bar{s} \equiv \bar{H}e \mod 2$. For such an e vector it holds that $x \equiv \bar{y} + e \mod 2$. □

Heuristic 2
Input: LLR vector $\lambda \in \mathbb{R}^n$, $H \in \{0,1\}^{m \times n}$, and $\{\Pi(1), \ldots, \Pi(n)\}$.
Output: A feasible solution $x \equiv \bar{y} + \bar{e} \mod 2$.

for all $j \in J$ **do**
  Set $\bar{y}_j = \frac{1}{2}\left[1 - \text{sign}(y)_j\right]$.
end for
Set $s \equiv H\bar{y} \mod 2$.
if $s_i = 0$ for all $i \in I$ **then**
  Set $x = \bar{y}$, output x.
else
  Call find alternative parity check matrix algorithm with H and $\Pi(n)$.
  Set IncreaseInObjective=∞ $\bar{e}_h = 0$ for all $h \in J$.
  Set TempSum$= 0$, $e_h = 0$ for all $h \in J$.
  for all $j \in J$ **do**
    Set TempSum$= |\lambda_j|$, $e_j = 1$
    for all $i \in I$ **do**
      if $\bar{s}_i = \bar{H}_{i,j}$ **then**
        Get next i.
      else
        Find index $\omega(i)$.
        TempSum=TempSum+$\left|\lambda_{\omega(i)}\right|$
        $e_{\omega(i)} = 1$
      end if
    end for
    if TempSum $<$ IncreaseInObjective **then**
      IncreaseInObjective = TempSum
      $\bar{e}_h = e_h$ for all $h \in J$
    end if
    Set TempSum$= 0$, $e_k = 0$ for all $k \in J$
  end for
end if

For short block lengths, heuristic 2 outputs good estimates. Heuristic 1 on the other hand is inferior to heuristic 2. This is demonstrated on a $(3, 6)$-regular LDPC code with $n = 40$ in Figure 4.9. Heuristic 2 approximates the MLD curve. In our numerical experiments it is observed that as the block length increases the accuracy of the estimates of heuristic 2 decrease. Thus, passing

the estimates obtained from heuristic 2 to the CPLEX IP solver as an initial solution does not lead to a significant improvement in terms of decoding time of MLD.

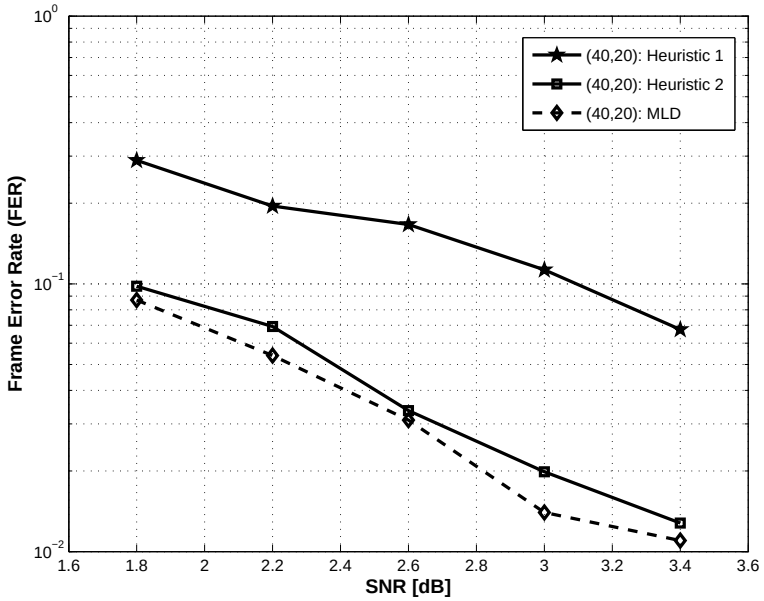


Figure 4.9: Error correcting performances of MLD, heuristic 1, heuristic 2.

For the computation of the minimum distance of a code the corresponding integer programming problem has to be solved only once. It may be worthwhile to put more computational effort in finding minimum weight codeword estimates for the minimum distance problem. An intuitive procedure to find initial solutions for the minimum distance problem is proposed in heuristic 3. In heuristic 3, a given parity check matrix H is first brought into systematic form H^s. The last m columns of H^s form an identity matrix.

Proposition 4.5. *Let $w(H^s_{.,j})$ denote the weight (number of ones) of column $j \in \{1,\ldots,n-m\}$ and $\Omega = \{n-m+i : H^s_{i,j} = 1, i \in \{1,\ldots,m\}\}$. Then a codeword $x \in C$ can be constructed from $\{\Omega \cup \{j\}\}$ by setting $x_h = 1$ for all $h \in \{\Omega \cup \{j\}\}$ and $x_h = 0$ for all $h \in J \setminus \{\Omega \cup \{j\}\}$. Moreover, $w(H_{.,j}) + 1$ is an upper bound on the minimum distance.*

Proof. The proof follows from the fact that columns in $\Omega \cup \{j\}$ are linearly dependent. For an $x \in \{0,1\}^n$, if $x_h = 1$, for all $h \in \{\Omega \cup \{j\}\}$ and $x_h = 0$ for all $h \in J \setminus \{\Omega \cup \{j\}\}$ then it holds that $Hx \equiv 0 \bmod 2$. □

Heuristic 3 can be extended at the cost of increasing computation time. For example, all two element subsets of $\{1,\ldots,n-m\}$ can be considered. In this

Heuristic 3
Input: $H \in \{0,1\}^{m \times n}$.
Output: Initial solution x^{init}.

Set UB to ∞, x^{init} to all-zeros.
Transform H to H^s.
for $j = 1$ to $n - m$ **do**
  Compute $w(H_{.,j})$, find Ω.
  for all $h \in J$ **do**
    if $h \in \{\Omega \cup \{j\}\}$ **then**
      Set $x_h = 1$.
    else
      Set $x_h = 0$.
    end if
  end for
  if $w(H^s_{.,j}) + 1 <$ UB **then**
    Set UB $= w(H^s_{.,j}) + 1$.
    Set $x^{init} = x$.
  end if
end for
Terminate with UB and x^{init}.

case, a set Γ can be constructed as: $\Gamma = \{j,k\} \cup \{n - m + i : H^s_{i,j} + H^s_{i,k} \equiv 1 \bmod 2, i \in \{1,\dots,m\}\}$ where $\{j,k\} \subset \{1,\dots,n-m\}$.

In our numerical experiments it is observed that for some codes, finding an initial solution reduces the computation time insignificantly. This statement can not be generalized since especially for codes with short block length passing an initial solution to CPLEX has a negative effect on the total computation time.

4.5 Conclusion and further research

In this chapter we modeled the ML decoding problem as a mathematical programming problem. We studied 8 different formulations for modeling the parity check equations in $GF(2)$ by linear equalities or linear inequalities in $\mathbb{R}^n$. Some of these formulations are efficient for every binary parity-check matrix whereas others are only efficient for binary matrices with some special structure. A comparison of the sizes of different formulations in O notation is given in Table 4.1. Formulations also differ in memory requirements and computation time when solved with the CPLEX IP solver.

For LDPC and BCH codes starting with few constraints and variables in the initial formulation and iteratively adding cuts seems to be favorable. On

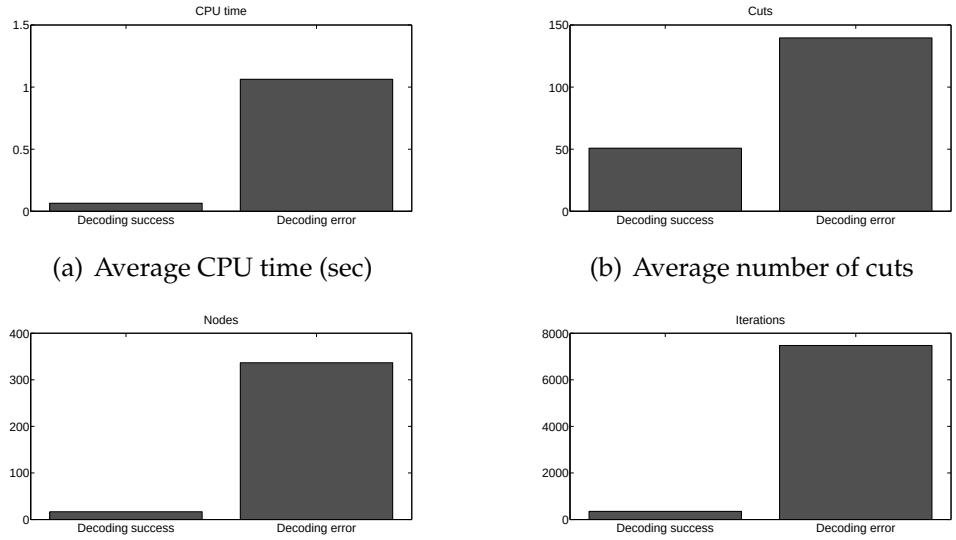
the other hand, for turbo codes the network flow based formulation performs computationally better than formulations with less variables and constraints. It can be concluded that it is worthwhile to make an analysis of different formulations for different code classes. Choosing the appropriate formulation for an IP problem is of crucial importance since it affects the computation time and the sizes of the solvable instances. This is demonstrated for BCH codes in Table 4.6 and for LTE turbo codes in Table 4.9.

The computational results in Section 4.3 imply that for binary linear codes with short and medium block length, ML curves, minimum distances can be computed by the mathematical modeling approach without the need of developing a specially designed, code-dependent algorithm. Moreover, the IP formulations can easily be modified to model puncturing or to compute the codeword weight distribution (see [58]).

Finally we report two more numerical experiments on MLD which may initiate further research. The selected code is the (80,40) LDPC code and MLD is performed by solving IPD1 with the CPLEX IP solver. 1000 instances are solved. The SNR value is fixed to 1.4 where the FER for the selected code is approximately 10%.

First, in Figures 4.10(a), 4.10(b), 4.10(c), and 4.10(d) the average computation times in CPU seconds, the average number of cuts, nodes and iterations for the instances detected as decoding success and decoding error are compared. It can be concluded that more computational effort is put in the instances detected as decoding error. Thus, it is reasonable to set limits to these parameters to obtain approximations of ML curves in less time.

Second, the distribution of the average number of cuts from different cut families employed by the CPLEX IP solver is presented. The IP solver generates cuts from more than 10 different cut families (see [1]). For MLD the most frequently used cuts are Gomory fractional cuts, implied bound cuts, mixed integer rounding cuts, and zero-half cuts. Considering the special structure of the MLD problem theoretical research can be done on these cut families.



(a) Average CPU time (sec)

(b) Average number of cuts

(c) Average number of nodes

(d) Average number of iterations

Figure 4.10: Average CPU time, average number of cuts, nodes, iterations for the instances detected as decoding success and decoding error, (80,40) LDPC code, SNR=1.4.

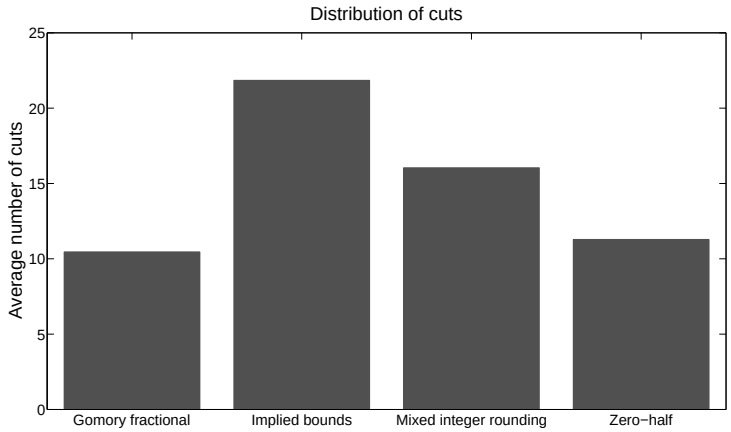


Figure 4.11: Distribution of cuts, (80,40) LDPC code, SNR=1.4.

Chapter 5

Linear programming based methods for decoding of binary linear codes

5.1 Introduction

Typically, turbo codes and LDPC codes are decoded in an iterative manner where probabilistic information is exchanged between corresponding component decoders. This is known as iterative message passing decoding (IMPD). Recently, linear programming decoding (LPD) which was introduced by Feldman et al. [23], [27] has become an interesting alternative for decoding of binary linear block codes. LPD is a polynomial time decoding approach with some desirable properties such as the maximum likelihood (ML) certificate property: if LPD outputs a codeword, then it is the ML codeword. Moreover, this approach allows finite length analysis due to the geometry of linear programming and it is not limited to sparse matrices. In this chapter we review several LP-based decoding approaches for binary linear codes.

The rest of this chapter is organized as follows. In Section 5.2 a general description of LPD is given. The linear programming (LP) relaxations of the integer programming (IP) formulations presented in Section 4.2 are mostly discussed in Section 5.3. Based on the various LP formulations, different streams of research on LPD have evolved: in Section 5.4, methods focusing on efficient realization of LPD are summarized, while approaches improving the error-correcting performance of LPD at the cost of increasing complexity are reviewed in Section 5.5. Some concluding comments are made in Section 5.6.

5.2 Basics of linear programming decoding

Although several structural properties of $\mathrm{conv}(C)$ are known (see Chapter 3), a concise description of $\mathrm{conv}(C)$ by means of linear inequalities cannot be stated in general. In LPD, the linear cost function of the IP formulation for maximum

likelihood decoding (MLD) (see Chapter 4) is minimized on a relaxed polytope $\mathcal{P}$ where $\text{conv}(C) \subseteq \mathcal{P} \subseteq \mathbb{R}^n$. Such a relaxed polytope $\mathcal{P}$ should have the following desirable properties: it should be easy to describe and codewords should correspond to vertices of $\mathcal{P}$.

A local code C_i, $i \in \{1, \ldots, m\}$ is defined as $C_i = \{x \in \{0,1\}^n : H_{i,}x \equiv 0 \bmod 2\}$. It holds that $C = C_1 \cap \ldots \cap C_m$. The description complexity of the convex hull of any local code $\text{conv}(C_i)$ is is smaller than the description complexity of the codeword polytope. Thus, a particularly interesting relaxation of $\text{conv}(C)$ is $\mathcal{P} = \text{conv}(C_1) \cap \ldots \cap \text{conv}(C_m)$ known as the fundamental polytope [70]. Set of vertices of the fundamental polytope, the so-called pseudocodewords, is a super set of C. Consequently, an integral LP optimum is an ML codeword. These observations are formally stated in the following lemma.

Lemma 5.1. *[70] Let $\mathcal{P} = \text{conv}(C_1) \cap \ldots \cap \text{conv}(C_m)$. If $C = C_1 \cap \ldots \cap C_m$ then $\text{conv}(C) \subseteq \mathcal{P}$.*

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

$$\min\{\lambda^T x : x \in \mathcal{P}\}. \tag{5.1}$$

Based on (5.1), the LPD algorithm [27] 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.

1: Solve the LP given in (5.1).
2: **if** LP solution x^* is integral **then**
3: Output x^*.
4: **else**
5: Output error.
6: **end if**

If BLPD outputs a codeword, then it is the ML codeword (this is known as the ML certificate property). BLPD succeeds if the transmitted codeword is the unique optimum of the LP given in (5.1). 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.

For some cases, the fundamental polytope $\mathcal{P}$ is equivalent to $\text{conv}(C)$, e.g., if the underlying Tanner graph is a tree or forest [70]. In these cases MLD can be achieved by BLPD.

To motivate the definition of fractional distance, we first rewrite the minimum distance of a code as

$$d(C) = \min \left\{ \sum_{j=1}^{n} |x_j - \bar{x}_j| : x, \bar{x} \in C,\ x \neq \bar{x} \right\} \tag{5.2}$$

where $|\cdot|$ denotes the l_1 norm.

For a given fundamental polytope $\mathcal{P}$, the notion of fractional distance was introduced in [27].

Definition 5.2. *Let $V(\mathcal{P})$ be the set of vertices of $\mathcal{P}$, i.e., the set of pseudocodewords. The fractional distance d_{frac} of a polytope $\mathcal{P}$ is the minimum l_1 distance between a codeword and any other vertex $v \in V(\mathcal{P})$.*

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

The fractional distance is a lower bound for the minimum distance of a code, i.e., $d(C) \geq d_{\text{frac}}(\mathcal{P})$. For binary linear codes, it is shown in [23] that the fractional distance is equal to the minimum l_1 weight of a non-zero vertex of a C-symmetric polytope (see [23, Chapter 4]) $\mathcal{P}$. Given a C-symmetric polytope $\mathcal{P}$, the fractional distance of a binary linear code can be found by the fractional distance algorithm (FDA). Let x^0 denote the all-zero codeword, let M denote the set of inequalities describing $\mathcal{P}$, and let $\mathcal{F} \subset \mathrm{M}$ denote the set of inequalities which are not active at x^0, i.e., not satisfied with equality by x^0. In FDA, the weight function $\sum_{j \in J} x_j$ is minimized on $\mathcal{P} \cap f$ for all $f \in \mathcal{F}$. In this way, the minimum weight non-zero vertex of $\mathcal{P}$ is found.

Fractional distance algorithm (FDA)
Input: $\mathcal{P} \subseteq [0,1]^n$.
Output: Minimum weight non-zero vertex of $\mathcal{P}$.

1: **for all** $f \in \mathcal{F}$ **do**
2: Set $\mathcal{P}' = \mathcal{P} \cap f$.
3: Solve $\min\{\sum_{j \in J} x_j : x \in \mathcal{P}'\}$.
4: **end for**
5: Choose the minimum value obtained over all $\mathcal{P}'$.

5.3 Methods based on solving an LP relaxation

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 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 another is a simplex iteration. Details on the simplex algorithm can be found in classical books about linear programming, e.g., [60].

The efficiency of the simplex method depends on the constraint set describing the underlying polytope. Several constraint sets which describe the fundamental polytope $\mathcal{P}$ explicitly have been proposed in the LPD literature. Some of these constraint sets can be written for any binary linear code whereas others are specialized for certain subclasses of codes. Using alternative descriptions of $\mathcal{P} \subseteq [0,1]^n$, alternative LP decoders are obtained. In the following, we are going to discuss different LP formulations obtained by relaxing the integrality constraints in the IP formulations presented in Section 4.2.

In order to describe $\mathcal{P}$ explicitly, three alternative constraint sets are suggested in [27]. The constraint sets are obtained by relaxing the integrality constraints in the IP formulations IPD3, IPD5, IPD7. We use the abbreviations LPD3, LPD5, LPD7 to denote the associated LP formulations. Note that 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.

In LPD3, for each $i \in I$, the constraints describe $\text{conv}(C_i)$. In LPD5, the forbidden set (FS) inequalities are used. If the rows of H are considered as dual codewords, the set of FS inequalities is a reinvention of cocircuit inequalities explained in Section 3.3. The number of variables, constraints in LPD3 and the number of constraints in LPD5 increase exponentially in the check node degree. Thus, for codes with high-density parity-check matrices, LPD3 and LPD5 are computationally inefficient. LPD7 is suggested as a suitable formulation for codes with high-density parity-check matrices.

Feldman et al. [27] show that LPD3, LPD5, and LPD7 are equivalent in the sense that the x variables of the optimal solutions in all three formulations take the same values. Moreover, it is shown that $\text{conv}(C_i)$ can be described completely and non-redundantly by the FS inequalities and the boxing inequalities $0 \leq x_j \leq 1$, $j \in J$. In a more general setting Grötschel [33] proved this result for the cardinality homogeneous set systems.

In particular, Feldman et al. [27] apply BLPD using formulations LPD3 or LPD5 to LDPC codes. Under the binary symmetric channel (BSC), the error-correcting performance of BLPD is compared in [27] with the min-sum decoding (MSD) of the random rate-$\frac{1}{2}$ LDPC code with $n = 200$, $d_v = 3$, $d_c = 6$; with the MSD, sum-product decoding (SPD) on the random rate-$\frac{1}{4}$ LDPC code with $n = 200$, $d_v = 3$, $d_c = 4$; with the MSD, SPD, MLD on the random rate-$\frac{1}{4}$ LDPC code with $n = 60$, $d_v = 3$, $d_c = 4$. On these codes, BLPD performs better than

MSD but worse than SPD. Using LPD5, the FDA is applied to random rate-$\frac{1}{4}$ LDPC codes with $n = 100, 200, 300, 400$ $d_v = 3$, $d_c = 4$ from an ensemble of Gallager [29]; Reed-Muller $(n-1, n)$ codes [28] with n changing between 4 to 1024 to calculate fractional distance.

The number of variables and constraints in LPD7 are in $O(n^3)$. By applying a decomposition approach, Yang, Wang, and Feldman [77] show that an alternative LP formulation which has size linear in the block length and maximum check node degree can be obtained (see Section 4.2 for the associated IP formulation, IPD6). In LPD6, the LP relaxation of IPD6, the boxing inequalities can be dropped since the convex hull of a single check node of degree 3 is described by the FS inequalities.

Yang et al. [77] and Chertkov and Stapanov [15] prove that the formulations introduced in [27] and LPD6 are equivalent. Again, equivalence is used in the sense that in an optimal solution, the x variables of LPD3, LPD5, LPD7 and the variables LPD6, which correspond to original x variables, take the same values. Moreover, it is shown that LPD6 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 LPD5, FDA is applicable only for LDPC codes. If $\mathcal{P}$ is described by the constraint set of LPD6, then in the first step of FDA, it is sufficient to choose the set $\mathcal{F}$ from the facets formed by cutting planes of type $\nu_1^k + \nu_2^k + \nu_3^k = 2$ where ν_1^k, ν_2^k, and ν_3^k are variables of the LPD6 formulation. Additionally, an adaptive branch & bound method as in [76] is suggested to find better bounds for minimum distance of a code. This is demonstrated in [77] on a random rate-$\frac{1}{4}$ LDPC code with $n = 60$, $d_v = 3$, $d_c = 4$.

The LP relaxation of IPD8, LPD8, models a minimum cost flow problem with additional side constraints and it can be solved by a general purpose LP solver. Specifically, LPD8 is applied to repeat accumulate (RA(l)) codes (see Feldman and Karger [25]). The encoder of an RA(l) repeats the information bits l times, and then sends them to an interleaver followed by an accumulator. The authors derive bounds on the error rate of LPD8 for RA codes which are later improved and extended by Halabi and Even [37] as well as Goldenberg and Burshtein [32].

5.4 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. LPD6 introduced in [77] can be considered as an efficient LPD approach since the number of variables and constraints are significantly reduced. We review several others in this section.

5.4.1 Solving the separation problem

The approach of Taghavi and Siegel [63] tackles the large number of constraints in LPD5. In their separation approach called adaptive linear programming decoding (ALPD), not all FS inequalities are included in the LP formulation as in LPD5. Instead, they are iteratively added when needed. In the initialization step, the LP $\min\{\lambda^T x : 0 \leq x \leq 1\}$ is solved. Let $(x^*)^k$ be the optimal solution in iteration k. Taghavi and Siegel [63] show that it can be checked in $O(md_c^{max} + n\log n)$ time if $(x^*)^k$ violates any FS inequality derived from $H_{i,.}x \equiv 0 \bmod 2$ for all $i \in I$. This check can be considered as a special case of the greedy separation algorithm (GSA) introduced in [33]. 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 [63] is that ALPD converges to the same optimal solution as LPD5 with significantly less constraints. It is shown empirically that in the last iteration of ALPD, less constraints than in the formulations LPD5, LPD6, and LPD7 are used. Taghavi and Siegel [63] 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.

Using binary-input additive white Gaussian noise channel (BIAWGNC), Taghavi and Siegel [63] use 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 $R = \frac{1}{2}$, 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 $R = \frac{1}{2}$ is demonstrated on a $(3, 6)$-regular LDPC code. For these codes, it is shown 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 remains below 16. The authors also investigate the effect of the rate on the number of FS inequalities created. Simulations are performed on a code 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, LPD5, and SPD (50 iterations), the average decoding time versus various block lengths are tested on $(3, 6)$-regular, $(4, 8)$-regular LDPC codes. It is shown that the computation time needed for ALPD is between the computation time of BLPD and the computation time

of SPD. 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 parity inequalities used, under increasing signal to noise ratio (SNR) is tested on the $(3, 6)$-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 [62] ALPD is improved further in terms of complexity. Taghavi, Shokrollahi, and Siegel [62] use some structural properties of the fundamental polytope. Let $(x^*)^k$ be an optimal solution in iteration k. In [63] 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. [62] modify ALPD and propose the decoding approach we refer to as modified adaptive linear programming decoding (MALPD). In the $(k+1)^{th}$ iteration of MALPD, it is checked in $O(md_c^{max})$ time if $(x^*)^k$ violates any FS inequality derived from $H_{i,:}x \equiv 0 \bmod 2$ for some $i \in I$. This check is only performed for the 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 many FS inequalities. Under the BIAWGNC, the average number of iterations performed by ALPD and MALPD are compared on various regular LDPC codes. It is concluded that MALPD converges in slightly more iterations than ALPD.

5.4.2 Message passing-like algorithms

An approach towards low complexity LPD of LDPC codes is proposed by Vontobel and Kötter in [71]. They first formulate an LP, based on the Forney-style factor graph (FFG) representation of C. The LP relaxation of IPD4, LPD4, is motivated by the usage of the structure of the LP to derive a special, message passing like LP solver. Instead of solving LPD4 with a general purpose LP solver, Vontobel and Kötter [71] derive the dual of the above formulation. The authors manipulate the constraints of the dual problem to obtain a closely related, "softened" dual linear programming decoding (SDLPD). SDLPD can then be treated efficiently: 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 SPD. The coordinate-ascent-type algorithm for SDLPD is guaranteed to converge if all the edges of FFG are updated cyclically.

Under the BIAWGNC, the authors compare the error-correcting perfor-

mance of the coordinate-ascent-type algorithm (max iterations: 64, 256) against the performance of MSD (max iterations: 64, 256) on the $(3, 6)$-regular LDPC code with $n = 1000$ and rate $R = \frac{1}{2}$. MSD performs slightly better than the coordinate-ascent-type algorithm. To sum it up, Vontobel and Kötter [71] show 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 [71] is studied further in [10] by Burshtein. His algorithm has a new scheduling scheme and its convergence rate and computational complexity is analyzed under this scheduling. Using the new scheme, the decoding algorithm of Vontobel and Kötter [71] yields an iterative approximate LPD algorithm for LDPC codes with complexity in $O(n)$. The main difference between Burshtein's algorithm [10] and the algorithm of Vontobel and Kötter [71] is the selection and update of edges of the FFG. In [71] all edges are updated cyclically during one iteration, whereas in [10], only few selected edges are updated during one particular iteration. The edges are chosen according to the variable values obtained during previous iterations.

5.4.3 Non-linear programming approach

As an approximation of BLPD for LDPC codes, Yang, Feldman, and Wang [76] introduce the box constraint quadratic programming decoding (BCQPD) which is a linear time decoding algorithm. BCQPD is a non-linear programming approach derived from the Lagrangian relaxation (see [55] for an introduction to Lagrangian relaxation) of LPD3. To achieve BCQPD, a subset of the set of the constraints are incorporated into the objective function. To simplify the notation Yang et al. rewrite the constraint blocks (4.1), (4.2) as $Ay = b$ where the matrix A and the vector b are defined appropriately. Likewise, the variables in LPD3 are grouped into a new variable vector y of length K. Finally, the objective function coefficients are rewritten in a vector c also of length K. The resulting formulation is $\min\{c^T y : Ay = b, y \in \{0,1\}^K\}$. Using a multiplier $\alpha > 0$, the Lagrangian of this problem is

$$\begin{aligned} &\text{minimize } \; c^T y + \alpha(Ay - b)^T(Ay - b) \\ &\text{subject to } \; 0 \le y_k \le 1 \qquad\qquad \text{for all } k \in \{1, \ldots, K\}. \end{aligned}$$

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 is obtained.

$$\begin{aligned} &\text{minimize } \; y^T Q y + 2r^T y \qquad\qquad \text{(BCQPD)} \\ &\text{subject to } \; 0 \le y_k \le 1 \qquad\qquad \text{for all } k \in \{1, \ldots, K\}. \end{aligned}$$

Since Q is a positive semi-definite matrix, i.e., the objective function is convex, and since the set of constraints constitute a box, each y_k can be minimized separately. This leads to efficient serial and parallel decoding algorithms. Two methods are proposed in [76] to solve the BCQPD problem, the projected successive overrelaxation method (PSORM) and the parallel gradient projection method (PGPM). These methods are generalizations of Gaussian-Seidel and Jacobi methods [8] 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 SPD. For numerical tests, the authors use a product code with block length $4^5 = 1024$ and rate $(\frac{3}{4})^5 = 0.237$. The BIAWGNC is used. It is observed that the PSORM method converges faster than PGPM. The error-correcting performance of SPD is poor for product codes due to their regular structure. For the chosen product code, Yang et al. demonstrate that PSORM outperforms SPD in computational complexity as well as in error-correcting performance.

5.4.4 Efficient LPD of SPC product codes

A code class of special interest in [77] is single parity-check (SPC) product codes built from SPC codes. The authors prove that for the class of SPC product codes the fractional distance is equal to the minimum Hamming distance. Due to this observation, the minimum distance of SPC product codes can be computed in polynomial time using FDA. Furthermore, Yang et al. [77] propose a low complexity algorithm which approximately computes the optimum of LPD6 for SPC product codes. This approach is based on the observation that a possible parity-check matrix of an SPC product code can be decomposed into component SPC codes. A Lagrangian relaxation of LPD6 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 [55]). 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. 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 per-

formance of SD if the number of iterations are set to $5, 10, 20, 100$, a (5,2) SPC product code with $n = 25$ and rate $R = \left(\frac{4}{5}^2\right) = 0.64$ is used. The error-correcting performance is improved by increasing the number of iterations. Under the BIAWGNC, this code and a (4,4) SPC product code with $n = 256$ and rate $R = \left(\frac{3}{4}^2\right) \approx 0.32$ are used to compare the error-correcting performance of SD, JSD with the performance of BLPD and MLD. It should be noted that BLPD and MLD have the same error-correcting performance for SPC product 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, a $(10, 3)$ SPC product code with $n = 1000$ and rate $R = (\frac{9}{10})^3 \approx 0.729$ is used to compare the error-correcting performance of SD and JSD with the SPD. Again the BIAWGNC is used. It is observed that SD performs slightly better than the SPD with a similar computational complexity. JSD improves the error-correcting performance of the SD at the cost of increased complexity.

5.4.5 Interior point algorithms

Efficient LPD approaches based on interior point algorithms are studied by Vontobel [69], Wadayama [72], and Taghavi et al. [62]. The use of interior point algorithms to solve LP problems as an alternative to the simplex method was initiated by Karmarkar [43]. 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 against the simplex method which has exponential worst case complexity.

The proposed interior point algorithms aim at using the special code structure. 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 [69], primal-dual interior point algorithms [62], [69] and primal path following interior point algorithm [72] 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 [69], [62]. Under the BIAWGNC, Wadayama [72] and Taghavi et al. [62] demonstrate the convergence behaviors of their algorithms on the nearly $(3, 6)$-regular LDPC code with $n = 1008$, $R = \frac{1}{2}$ and a randomly-generated $(3, 6)$-regular LDPC code with $n = 2000$, respectively.

5.5 Improving the error-correcting performance of BLPD

The error-correcting performance of BLPD can be improved by techniques from integer programming. Most of these 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.

5.5.1 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 conv(C). Valid cuts are valid inequalities which are violated by some non-integral vertex of the LP relaxation. Feldman et al. [27] obtain valid inequalities either by using the lift and project technique [48] or deriving FS inequalities from redundant parity checks. We refer to the latter inequalities as redundant parity-check (RPC) inequalities. RPC inequalities may include valid cuts which increase the possibility that BLPD outputs a codeword. An interesting question relates to the types of inequalities required to describe the codeword polytope conv(C) exactly. In general, conv(C) cannot be described completely by using only FS inequalities and boxing inequalities. The $(7, 3, 4)$ Simplex code (dual of the $(7, 4, 3)$ Hamming code) is given as a counter example in [27]. In general, it can be concluded from [36] that the FS inequalities do not define the facets of a simplex code.

RPCs can also be interpreted as dual codewords. Obviously, there exist exponentially many RPC inequalities. The RPC inequalities cutting off the non-integral optimal solutions are called RPC cuts [63]. There are several approaches in the LPD literature (cf. [27], [53], [63], [64]) to find those RPCs for which the associated RPC inequalities include an RPC cut. In [27] RPCs which result from adding any two rows of H are appended to the original parity-check matrix. In [63] the RPCs are found from fractional cycles in the Tanner graph. In [53] 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 [64], a possible parity-check matrix is reformulated such that unit vectors are obtained in the columns 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.

Feldman et al. [27] test the lift and project technique (see [48]) on the random rate-$\frac{1}{4}$ LDPC code with $n = 36$, $d_v = 3$ and $d_v = 4$ under the BIAWGNC. The random rate-$\frac{1}{4}$ 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 modified to include all RPCs obtained by adding any

two rows. Both tightening techniques improve the error-correcting performance of BLPD.

The idea of tightening the fundamental polytope is usually implemented as a cutting plane algorithm (the separation problem is solved). In cutting plane algorithms, an LP problem which contains only a subset of the constraints from the corresponding optimization problem, is solved. If the optimal LP solution is a codeword then the cutting plane algorithm terminates and outputs the ML codeword. If the optimal LP solution is not a codeword then 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 problem is resolved. In [53], [64], [63] the family of valid cuts is FS inequalities derived from RPCs.

In [53] the main motivation for the greedy cutting plane algorithm is to improve the fractional distance. This is demonstrated for $(7, 4, 3)$ Hamming code, $(24, 12, 8)$ Golay code and $(204, 102)$ LDPC code. As a by product under the BSC it is shown on the $(24, 12, 8)$ Golay code and $(204, 102)$ LDPC code that the RPC based approach of [53] improves the error-correcting performance of BLPD.

In the improved LPD approach of [63], first the ALPD (see Section 5.4) 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 from the Tanner graph for which x_j^* is integral. 2) Find a cycle by randomly walking through the pruned Tanner graph. 3) Add the rows of the H matrix in $GF(2)$ which correspond to the check nodes in the cycle. 4) Check if the found RPC introduces a cut. The improved decoder of [63] performs better than BLPD and SPD. This is shown under the BIAWGNC, on the $(3, 4)$-regular LDPC codes with $n = 32, 100, 240$.

The cutting plane approach of [64] is based on the IP formulation IPD1 (see Section 4.2). The formulation is referred to as IPD and it was also mentioned in [9]. In [64] the initial LP problem is the LP relaxation of IPD1 and it is solved by a cutting plane algorithm. Note that the LP relaxation of IPD1 is not equivalent to the LP relaxations given in Section 5.3. In almost all improved (in the error-correcting performance sense) LPD approaches reviewed in this chapter 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 [64] it is not elaborated on the solution of BLPD. It is immediately searched for cuts which can be derived from arbitrary dual codewords. To this end a possible parity-check matrix is modified and the conditions are checked under which certain RPCs define cuts. The average number of iterations performed and the average number of cuts generated in the separation algorithm decoding (SAD) of [64] are presented for the $(3, 6)$ random regular codes with $n = 40, 80, 160, 200, 400$ and for the $(31, 10), (63, 39), (127, 99), (255, 223)$ (Bose-Chaudhuri-Hocquenghem) BCH codes. These two performance measures are directly proportional with the block length. The error-correcting per-

formance of SAD is measured on the random $(3, 4)$-regular LDPC codes with block length 100, 200, and Tanner's $(155, 64)$ group structured LDPC code [65]. It is demonstrated that the improved LPD approach of [64] performs better than BLPD applied in the adaptive setting [63] and the SPD. One significant numerical result is that SAD proposed in [64] performs much better than BLPD for the $(63, 39)$ and $(127, 99)$ BCH codes. In all numerical simulations the BIAWGNC is used.

Yufit, Lifshitz and Be'ery [79] improve SAD of Tanatmis et al. [64] and ALPD of Taghavi et al. [63] 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, $Aut(C)$. 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 [79], 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 $Aut(C)$ 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 [79]. 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 a possible parity-check matrix prior to decoding such that at the columns 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 [62] is to drop the inactive inequalities at each iteration of SAD. In this way it is avoided 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 in terms of error-correcting performance and computational complexity.

5.5.2 Facet guessing approach

Based on LPD5, Dimakis, Gohari, and Wainwright [19] improve the error-correcting performance of BLPD with an approach similar to FDA (see Section 5.2). 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), $|\mathcal{K}|$ many 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 $\lambda^T v < \lambda^T x^{ML}$ for all $f \in \mathbb{A}(x^{ML})$. For suitable expander codes this result is combined with the following structural property of expander codes also proven by the authors. The number of active inequalities of a codeword is much more than the number of active inequalities in a non-integral pseudocodeword. Consequently, theoretical bounds on the decoding success conditions of the polynomial time algorithms EFGD, RFGD for expander codes are derived. The numerical experiments are performed under the BIAWGNC, on Tanner's $(155, 64)$ group structured LDPC code and a random LDPC code with $n = 200$, $d_v = 3$, and $d_c = 4$. For these codes the RFGD performs better than SPD.

It should be noted that an important structural property of the fundamental polytope is used in [19], [63], and [64]. Let $(x^*) \in \mathbb{R}^n$ be a point in the fundamental polytope. It can be shown that no check node of the associated Tanner graph can be adjacent to only one non-integral valued variable node. This result can be extended to the case where a possible parity-check matrix H contains all dual codewords and the fundamental polytope is formed by all FS inequalities derived from all dual codewords.

5.5.3 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 (for binary variables) the values of certain variables. We refer to [55] for a detailed description. Several authors improved LPD using branch & bound approach.

Breitbach, Bossert, Lucas, and Kempter [9] solve IPD1 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. We note that in [9], the heuristic approach is rather considered as a stand alone decoding algorithm. In the heuristic, IPD2 is used.

In the neighborhood search heuristic of [9], first a feasible starting solution e^0 is calculated by setting $n-m$ 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: $(31, 21, 5)$ BCH code, $(64, 42, 8)$ Reed-Muller code, $(127, 85, 13)$ BCH code, and $(255, 173, 23)$ 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 neighborhood search with single position exchanges gets worse. An extension of this heuristic allowing two position exchanges is applied to $(64, 42, 8)$ Reed-Muller code, $(127, 85, 13)$ BCH code, and $(255, 173, 23)$ BCH code. At the cost of an increased complexity, the extended neighborhood search heuristic improves the error-correcting performance. A branch & bound algorithm is simulated on the $(31, 21, 5)$ BCH code and different search tree exploration schemes are investigated. Breitbach et al. [9] suggest a combination of depth-first and breadth-first search.

In [20], Draper, Yedidia and Wang improve the ALPD approach of [63] with a branch & bound technique. Branching is done on the least certain variable, i.e., x_j such that $\left|x_j^* - 0.5\right|$ is smallest for $j \in J$. Under the BSC, it is observed on the Tanner's $(155, 64, 20)$ code that the ML codeword is found after few iterations in many cases.

In [76] two branch & bound approaches for LDPC codes are introduced. In ordered constant depth decoding (OCDD) and ordered variable depth decoding (OVDD), first LPD3 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 is motivated by this construction. It is experimentally shown in [76] 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 [55] instead. For OVDD, the number of LPs solved in the worst case is $2^{g+1} - 1$.

In OCDD the m-element subsets 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

$$x_k = b_k \text{ for all } k \in \mathcal{M} \tag{5.3}$$

to LPD3, 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 an integral solution, OCDD outputs $\hat{x}$. Otherwise another subset $\mathcal{M} \subseteq \mathcal{T}$ is chosen. OCDD exhausts all m-element subsets of $\mathcal{T}$. Thus, in the worst case $\binom{g}{m} 2^m + 1$ LPs are solved.

The branch & bound based improved LPD of Yang et al. [76] 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 $n = 60$, $R = \frac{1}{4}$, $d_c = 4$ and $d_v = 3$ that OCDD has a a

better error-correcting performance than BLPD and SPD. 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[1] used in [76] 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 [76]. On a longer code, a random LDPC code with $n = 1024$, $R = \frac{1}{4}$, $d_c = 4$ and $d_v = 3$, it is demonstrated that the OVDD performs better than BLPD and SPD.

Another improved LPD technique which can be interpreted as a branch & bound approach is randomized bit guessing decoding (RBGD) of Dimakis et al. [19]. RBGD is inspired from the special case that all facets chosen by RFGD (see Section 5.5.2) 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. There are 2^k different possibilities of assigning bit values to these k variables. LPD5 is run 2^k times with associated constraints for each of these assignments. The best integer valued solution in terms of the objective function λ is the output of RBGD. Note that by setting k to $c{\cdot}\log n$, a complexity being polynomial 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 [19]. To find this probability expression the authors first prove that for some expander codes, the number of non-integral components in any pseudocodeword scales linearly in block length.

Chertkov and Chernyak [12] applied the loop calculus approach [13], [14] 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 (see [12]) or to find branching rules for branch and bound (cf. [11]). Given parity-check matrix and channel output, in linear programming erasure decoding (LPED) of Chertkov and Chernyak [12] first BLPD is solved. If a codeword is found then the algorithm terminates. If a non-integral pseudocodeword is the optimum of LP then a so-called critical loop is found by employing loop calculus. The indices of the variable nodes along the critical loop form an index set $M \subseteq J$. LPED lowers the objective function coefficients λ_j of the variables x_j, $j \in M$ 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 [15] for Tanner's $(155, 64, 20)$ code. It is demonstrated that LPED corrects the decoding errors of BLPD for

[1] m, g are set such that OVDD and OCDD have similar worst case complexity.

this code.

In [11] Chertkov combines the loop calculus approach used in LPED [12] with RFGD of Dimakis et al. [19]. 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 sub LPD problems are deduced. The sub LPD problems 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 [76] 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 [12], Chertkov tested LGGD on the list of pseudocodewords generated in [15] for Tanner's $(155, 64, 20)$ code. Chertkov showed under the BIAWGNC that LGGD improves BLPD.

SAD of [64] is improved in terms of error-correcting performance by a branch & bound approach in [79]. In Algorithm 3 of [79], 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 absolute LLR value is chosen as the branching variable. Algorithm 3 terminates as soon as the search tree reaches the maximal allowed depth D_p. Under the BIAWGNC, on the $(63, 36, 11)$ BCH code and the $(63, 39, 9)$ BCH code Yufit et al. [79] demonstrate that the decoding performance of Algorithm 3 (enhanced with parity-check matrix adaptation) approaches MLD.

5.6 Conclusion and further research

Since the introduction of linear programming decoding of Feldman, Wainwright, and Karger [27], several concepts of mathematical programming have been applied in coding theory. This chapter aims at reviewing different LP formulations, polyhedral properties of the fundamental polytope, efficient solution strategies for LPD, and techniques for improving error-correcting performance. Other lines of research not covered in this chapter relate among others to LPD of non-binary codes, performance analysis, or relationship to IMPD.

Chapter 6

A separation algorithm for improved LP-decoding of binary linear block codes

6.1 Introduction

In this chapter, we concentrate on improving the error-correcting performance of linear programming decoding (LPD) using a separation algorithm. We base our discussion on a compact, non-binary integer programming (IP) formulation for the decoding problem. We attempt to find the maximum likelihood (ML) codeword by an iterative separation approach. First, we relax the IP formulation and solve the resulting linear program (LP). In case of a non-integral optimal solution, we derive inequalities which cut off this non-integral solution, add these inequalities to the LP formulation, and resolve the LP problem. This process continues until an optimal integer solution is found or further cuts cannot be generated.

It should be noted that this general IP approach known as the separation problem was first applied to LPD by Taghavi and Siegel [63]. They intended to lower the complexity of LPD by a separation approach. Furthermore, they improved the error-correcting performance of LPD by using redundant parity-checks (RPCs). Improving LPD, is also considered in the papers [12], [11], [19], [20], [63]. It is common to these approaches that first the LP decoding problem is solved. If the LP decoder fails, each of these approaches is equipped with some technique of manipulating the LP with the goal of finding the ML codeword at the cost of increased complexity.

In contrast our approach does not work in this two-phase fashion. In particular we do not elaborate on the solution of the LPD problem. Our motivation is to search immediately for cuts which can be derived from arbitrary dual codewords. To this end we change a possible parity-check matrix systematically

and check conditions under which certain inequalities define cuts. Specifically, our approach offers the following advantages which facilitate LP based decoding. Under some assumptions, we can prove that we detect violated inequalities in $O(d_c^{max})$ where d_c^{max} is the maximum check node degree. We formally show that particular type of inequalities introduced in [27] - referred to as forbidden set (FS) inequalities - are a subset of the set of Gomory cuts (see [55]) which can be deduced from the IP formulation we use. We provide empirical evidence that our separation algorithm performs better than LPD and sum-product decoding (SPD). This is mainly due to generating strong cuts efficiently using alternative representations of the codes at hand. We applied our algorithm, separation algorithm decoding (SAD), to decode $(3, 4)$-regular LDPC codes with block length 100, 200, Tanner's $(155, 64)$ group structured LDPC code, the $(63, 39)$ BCH code, and the $(127, 99)$ BCH code.

The rest of this chapter is organized as follows. The relevant literature is shortly reviewed in Section 6.2. In Section 6.3, we explain the IP formulation, its LP relaxation, and SAD. In Section 6.4, we present our numerical results and compare them with SPD, LPD, and the lower bound resulting from MLD. Some extensions of SAD and a new family of valid inequalities are studied in Section 6.5. The chapter is concluded with some remarks and further research ideas in Section 6.6.

6.2 Relevant literature

Each row $i \in I$ of a possible parity-check matrix defines a local code C_i, i.e., local codewords $x \in C_i$ are the bit sequences which satisfy the i^{th} parity-check constraint. In [27], a description of conv(C_i) is given.

Proposition 6.1. *The FS inequalities*

$$\sum_{j \in S} x_j - \sum_{j \in N_i \setminus S} x_j \leq |S| - 1 \; \forall S \in \Sigma_i \tag{6.1}$$

derived from row i, $i \in \{1, \ldots, m\}$, of a parity-check matrix H and the inequalities $0 \leq x \leq 1$, completely describe the convex hull conv(C_i) of the local codeword polytope C_i.

Proof. This is shown in [27, Theorem 4]. □

The fundamental polytope $\mathcal{P}$ is the intersection of the convex hulls of local codeword polytopes conv(C_i). Feldman et al. [27] introduced the LP decoder which minimizes $\lambda^T x$ over $\mathcal{P}$ where λ is the vector of log likelihood ratios.

In the LPD approach, the problem below is solved.

$$\begin{aligned} &\text{minimize } \lambda^T x && \text{(LPD)} \\ &\text{subject to } \sum_{j\in S} x_j - \sum_{j\in N_i\setminus S} x_j \leq |S|-1 && \text{for all } S\in\Sigma_i, i\in\{1,\ldots,m\} \\ &\qquad 0\leq x_j\leq 1 && \text{for all } j\in\{1,\ldots,n\}. \end{aligned}$$

If LPD has an integral optimal solution, then the ML codeword is found. If LPD has a non-integral optimal solution then the output is an error. The number of FS inequalities induced by check node i is $2^{d_c(i)-1}$ where $d_c(i) = \sum_{j=1}^{n} H_{i,j}$ is the check node degree, i.e., the number of edges incident to node i. LPD can thus be applied successfully to low density codes. As the check node degrees increase, the computational load of building and solving the LP model is however in general prohibitively large. This makes the explicit description of the fundamental polytope via FS inequalities inapplicable for high density codes. Another approach applicable to high density codes is to solve the corresponding separation problem of LPD.

In separation algorithms (see [55]) one iteratively computes families Λ of valid cuts until no further cuts can be found. In the separation algorithm of [63], which is called adaptive linear programing decoding (ALPD) by the authors, FS inequalities are not added all at once in the beginning as in [27] but iteratively. In other words, the separation problem for the fundamental polytope is solved by searching violated FS inequalities. In the initialization step of the LP, $\min\{\lambda^T x : 0\leq x\leq 1\}$ is computed. Let x^* be an optimal solution. It can be checked in $O(md_c^{max} + n\log n)$ time, if x^* violates any FS inequality where d_c^{max} denotes the maximum check node degree. Recently, it was shown in [62] that the complexity of this check can be reduced to $O(md_c^{max})$. If some of the FS inequalities are violated then these inequalities are added to the formulation and the LP is resolved including the new inequalities.

ALPD stops when the current optimal solution x^* satisfies all FS inequalities. If x^* is integral then it is the ML codeword, otherwise an error is returned. Note that putting LPD in an adaptive setting does not yield an improvement in terms of frame error rate since the same solutions are found. On the other hand, ALPD converges with fewer constraints than the LP decoder which has a positive effect on the computation time.

The advantages of LPD motivated researchers to find better approximations of the codeword polytope as part of MLD. One way is to tighten the fundamental polytope with new valid inequalities. Among some other generic techniques of cut generation, in [27] it is proposed to use FS inequalities derived from RPCs as potential cuts. We refer to these inequalities as RPC inequalities. Redundant parity-checks are obtained by adding a subset of rows of a possible parity-check matrix in $GF(2)$. These checks are redundant in

the sense that they do not alter the code (they may even degrade the performance of belief propagation [27]). However they induce RPC inequalities in the LP formulation which may cut off a particular non-integral optimal solution thus tightening the fundamental polytope. The RPC inequalities cutting off the non-integral optimal solutions are called RPC cuts [63]. Note that redundant parity-checks are dual codewords and there exist exponentially many redundant parity-checks as well as RPC inequalities. An open problem is to find redundant parity-checks efficiently such that the associated RPC inequalities include an RPC cut. This issue will be addressed in the next section.

To the best of the authors' knowledge three approaches for searching RPC cuts have been studied so far. First, using redundant parity-checks which result from adding any two rows of H (see [27]). Second, the approach in [63] where the redundant parity-checks are found from the fractional cycles in the Tanner graph. In the third approach, introduced in [53], 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 for.

6.3 Separation algorithm decoding

Our separation algorithm is based on the following formulation which we referred to as IPD1 in Chapter 4.

$$\begin{aligned} &\text{minimize } \lambda^T x && \text{(IPD1)} \\ &\text{subject to } Hx - 2z = 0 && \\ &\qquad x_j \in \{0,1\} && \text{for all } j \in \{1,\dots,n\} \\ &\qquad z_i \geq 0, \text{integer} && \text{for all } i \in \{1,\dots,m\}. \end{aligned}$$

IPD1 is an integer programming problem modeling an ML decoder. The auxiliary variable vector $z \in \mathbb{Z}^m$ ensures that the binary constraint $Hx \equiv 0 \bmod 2$ turns into a constraint over the real number field $\mathbb{R}$ which is much easier to handle. The number of constraints in this formulation is the same as the number of rows of a possible parity-check matrix.

Our approach is to solve the separation problem by iteratively adding new cuts $\Pi^T x \leq \Pi_0$ according to Definition 2.37 and solving the LP relaxation of IPD we refer to as relaxed integer programming decoding (RIPD).

$$\begin{aligned} &\text{minimize } \lambda^T x && \text{(RIPD)} \\ &\text{subject to } Hx - 2z = 0 && \\ &\qquad \Pi^T x \leq \Pi_0 \quad (\Pi, \Pi_0) \in \Lambda && \\ &\qquad 0 \leq x_j \leq 1 && \text{for all } j \in \{1,\dots,n\} \\ &\qquad z_i \geq 0 && \text{for all } i \in \{1,\dots,m\}. \end{aligned}$$

Note that in the initialization step there are no cuts of type $\Pi^T x \leq \Pi_0$, i.e., $\Lambda = \emptyset$. If RIPD has an integral solution $(x^*, z^*) \in \mathbb{Z}^{n+m}$ then x^* is the ML codeword. Otherwise we generate cuts of the type $\Pi^T x \leq \Pi_0$ in order to exclude the non-integral solution found in the current iteration. We add these inequalities to the formulation and solve RIPD again. In a non-integral solution of RIPD x or z (or both) is non-integral. If $x \in \mathbb{Z}^n$ and $z \in \mathbb{R}^m \setminus \mathbb{Z}^m$ then we add Gomory cuts (see [55]) which is a generic cut generation technique used in integer programming. Surprisingly, in this case Gomory cuts can be shown to correspond to FS inequalities.

Theorem 6.2. *Let $(x^*, z^*) \in \mathbb{Z}^n \times \mathbb{R}^m$ be the optimal solution of RIPD such that $z_i^* \in \mathbb{R} \setminus \mathbb{Z}$ for some $i \in I$. Then a Gomory cut which is violated by (x^*, z^*) is the FS inequality*

$$\sum_{j \in S} x_j - \sum_{j \in N_i \setminus S} x_j \leq |S| - 1 \tag{6.2}$$

where S is an odd cardinality subset of N_i such that $S = \left\{ j \in N_i \mid x_j^* = 1 \right\}$.

Proof. We apply the general method known as Gomory's cutting plane algorithm (see e.g. [55]) to our special case. Gomory cuts are derived from the rows of the simplex tableau[1] in order to cut off non-integral LP solutions and find the optimal solution to the integer linear programming problems. Consider RIPD at any step:

$$\begin{aligned} \text{minimize} \quad & \lambda^T x && \text{(RIPD)} \\ \text{subject to} \quad & Hx - 2z = 0 \\ & Ax \leq b \\ & 0 \leq x \leq 1 \\ & z \geq 0. \end{aligned}$$

where $\lambda, x \in \mathbb{R}^n,\ H \in \{0,1\}^{m \times n},\ z \in \mathbb{R}^m,\ A \in \{-1,0,1\}^{\eta \times n}$ for some $\eta \in \mathbb{N}_0$ and $b \in \mathbb{N}_0^{\lambda}$. Note that η is the number of constraints added iteratively until the current step, i.e., $\eta = |\Lambda|$. The $\eta \times n$ matrix A is the coefficient matrix of the iteratively added constraints, i.e., $\Pi^T x \leq \Pi_0 \quad (\Pi, \Pi_0) \in \Lambda$. (Note that $A \in \{-1,0,1\}^{\eta \times n}$ holds in each iteration since the iteratively added constraints turn out to be forbidden set type of inequalities.)

We denote the right hand sides of these constraints with the vector b. RIPD

[1] Details on the simplex algorithm can be found in classical books about linear programming, e.g., [60].

in standard form can be written as follows:

$$\begin{aligned}
&\text{minimize } \lambda^T x && \text{(RIPD)} && (6.3)\\
&\text{subject to } z - \bar{H}x = 0 && && (6.4)\\
&\qquad\qquad x + s^1 = 1 && && (6.5)\\
&\qquad\qquad Ax + s^2 = b && && (6.6)\\
&\qquad\qquad z \geq 0,\ x \geq 0,\ s \geq 0. && && (6.7)
\end{aligned}$$

where $\bar{H} := \frac{1}{2}H$, $s = (s^1, s^2) \in \mathbb{R}^{n+\eta}$. For ease of notation we rewrite (6.3)-(6.7) as

$$\begin{aligned}
&\text{minimize } \bar{\lambda}^T y && && (6.8)\\
&\text{subject to } Py = q && && (6.9)\\
&\qquad\qquad y \geq 0. && && (6.10)
\end{aligned}$$

Note that

$$\begin{aligned}
\bar{c}^T &= (\bar{\lambda}_1, \ldots, \bar{\lambda}_m, \bar{\lambda}_{m+1}, \ldots, \bar{\lambda}_{m+n}, \bar{\lambda}_{m+n+1}, \ldots, \bar{\lambda}_{m+2n+\lambda})\\
&= (0, \ldots, 0, \lambda_1, \ldots, \lambda_n, 0, \ldots, 0),\\
y^T &= (y_1, \ldots, y_m, y_{m+1}, \ldots, y_{m+n}, y_{m+n+1}, \ldots, y_{m+2n+\eta})\\
&= (z_1, \ldots, z_m, x_1, \ldots, x_n, s_1^1, \ldots, s_\eta^2) \text{ and}\\
q^T &= (q_1, \ldots, q_m, q_{m+1}, \ldots, q_{m+n}, q_{m+n+1}, \ldots, q_{m+n+\eta})\\
&= (0, \ldots, 0, 1, \ldots, 1, b_1, \ldots, b_\eta).
\end{aligned}$$

The constraint matrix P has $m+n+\eta$ rows and $m+2n+\eta$ columns. We denote the α^{th} row of P with $P_{\alpha,\cdot}$ where $\alpha \in \{1, \ldots, m+n+\eta\}$ and β^{th} column of P with $P_{\cdot,\beta}$ where $\beta \in \{1, \ldots, m+2n+\eta\}$. The component in row α and column β is denoted by $P_{\alpha,\beta}$. Additionally, we define the α^{th} unit vector as $e_{\cdot,\alpha} \in \mathbb{R}^{m+n+\eta}$. Thus, we rewrite P as

$$P = \left[e_{\cdot,1} \ldots e_{\cdot,m}, P_{\cdot,m+1} \ldots P_{\cdot,m+n}, e_{\cdot,m+1} \ldots e_{\cdot,m+n+\eta}\right].$$

The first m columns of the constraint matrix P are the unit vectors corresponding to the variables $\{z_1, \ldots, z_m\}$. Likewise, the last $n+\eta$ columns are the unit vectors corresponding to the slack variables $\{s_1^1, \ldots, s_\eta^2\}$.

The first m linear equations of $Py = q$ are of the form:

$$z_i - \frac{1}{2} \cdot \sum_{j \in N_i} x_j = 0 \text{ for all } i \in \{1, \ldots, m\}.$$

Let $y^* = (z^*, x^*, s^*) \in \mathbb{R}^{m+2n+\eta}$ be the optimal solution to (6.3)-(6.7). By assumption we have $x^* \in \{0,1\}^n$. For $i \in \{1, \ldots, m\}$, $z_i^* = \frac{1}{2}k_i$, where

$$k_i = \left|\{j \in N_i : x_j^* = 1\}\right|.$$

Obviously, $k_i \in \mathbb{N}_0$. If k_i is even, i.e., an even number of variable nodes are set to 1 in the neighborhood of the check node i, then $z_i^* \in \mathbb{N}_0$ holds. Otherwise, z_i^* is an odd multiple of $\frac{1}{2}$. We then consider the Gomory cut for this row i.

For the optimal solution y^* we can partition P into a basis submatrix P_B and a non-basis submatrix P_N, i.e., $P = [P_B\ P_N]$. Let B and N denote the index sets of the columns of P belonging to P_B and P_N, respectively. An $(m + n + \lambda) \times (m + n + \lambda)$ basis matrix, P_B, corresponding to the optimal solution y^* can be constructed as follows. First we take the columns $e_{.,1}, \ldots, e_{.,m}$ which are the identity vectors corresponding to the variables $\{z_1, \ldots, z_m\}$ into P_B. Second for $j = 1, \ldots, n$, we include the column $P_{.,m+j}$ if $x_j^* = 1$ or $P_{.,m+n+j}$ if $s_j^* = 1$ in P_B. There exists n such columns due to (6.5) and the fact that x is integral. Finally we take the columns $e_{.,m+n+1}, \ldots, e_{.,m+n+\lambda}$ corresponding to the slack variables which are written for the iteratively added constraints. The variables corresponding to the columns in the basis matrix are called basic variables. The remaining columns of P form the non-basis submatrix P_N. The columns of P_N are the columns $P_{.,m+j}$, $j = 1, \ldots, n$, for which $x_j^* = 0$ and the unit vectors $e_{.,m+j}$,$j = 1, \ldots, n$, for which $s_j^* = 0$. The variables corresponding to the columns in P_N are called non-basic variables.

The Gomory cut for row i of P is given by the inequality

$$\sum_{h \in N} (\bar{p}_{ih} - \lfloor \bar{p}_{ih} \rfloor)\, y_h \geq (\bar{q}_i - \lfloor \bar{q}_i \rfloor) \tag{6.11}$$

where $\bar{p}_{ih} = (P_B^{-1})_{i,.} \cdot (P_N)_{.,h}$, and $\bar{q}_i = (P_B^{-1})_{i,.} \cdot q$. Note that in this special case $i \leq m$ since only z^* has non-integral components. In the following, we investigate the structure of $(P_B^{-1})_{i,.}$, $(P_N)_{.,h}$, $\bar{p}_{ih}$, and $\bar{q}_i$.

For a fixed i, it can easily be verified that the entries $(P_B^{-1})_{i,l}$, $l = 1, \ldots, m + n + \eta$ of $(P_B^{-1})_{i,.}$ are given as

$$(P_B^{-1})_{i,l} \quad = \quad \begin{cases} 1, & \text{if } l = i \\ \frac{1}{2}, & \text{if } P_{i,l} = -\frac{1}{2}, x_j^* = 1, \\ & l = m + j, j \in \{1, \ldots, n\} \\ 0 & \text{otherwise} \end{cases}$$

(This can be verified by observing the changes on row i when we append an $(m + n + \lambda) \times (m + n + \lambda)$ identity matrix to P_B and perform the Gauss-Jordan elimination on the appended matrix in order to get P_B^{-1}. Alternatively this can be verified by checking that $P_B \cdot P_B^{-1} = I$.)

Having found $(P_B^{-1})_{i,.}$, $\bar{q}_i$ is then computed by

$$\bar{q}_i = (P_B^{-1})_{i,.} \cdot q \tag{6.12}$$

$$= q_i + \frac{1}{2} \sum_{\{j \in N_i : x_j^* = 1\}} q_{m+j} \tag{6.13}$$

$$= 0 + \frac{1}{2} \sum_{\{j \in N_i : x_j^* = 1\}} 1. \tag{6.14}$$

Thus, we showed that $\bar{q}_i$ is $\frac{1}{2}$ times the number of basic x variables in row i. Since z_i is not an integer, the number of basic x variables in row i is odd. It follows that in this special case the right hand side of the Gomory cut, $\bar{q}_i - \lfloor \bar{q}_i \rfloor$, is always $\frac{1}{2}$.

Next, we compute $\bar{p}_{ih} = (P_B^{-1})_{i,.} \cdot (P_N)_{.,h}$ for $h \in N$. The columns of P_N are the columns of P corresponding to non-basic x components, i.e., $x_j^* = 0$, and non-basic s components, i.e., $s_j^* = 0$, $j \in \{1, \ldots, n\}$. If $(P_N)_{.,h} = P_{.,m+j}$ such that $x_j^* = 0$, then for a fixed value of h, the entries of $(P_N)_{.,h}$, $(P_N)_{\phi,h}$, $\phi = 1, \ldots, m+n+\lambda$ are given as

$$(P_N)_{\phi,h} = \begin{cases} -\frac{1}{2}, & \text{if } P_{\phi,m+j} = -\frac{1}{2} \text{ and } \phi \leq m \\ 1, & \text{if } \phi = m + j = h \\ 0 & \text{otherwise.} \end{cases}$$

If $(P_N)_{.,h} = P_{.,m+n+j}$ such that $s_j^* = 0$, then $(P_N)_{.,h}$ is the unit vector $e_{.,m+j} \in \mathbb{R}^{m+n+\lambda}$.

For the case that $(P_N)_{.,h} = P_{.,m+j}$ where $x_j^* = 0$, the only position where both $(P_B^{-1})_{i,.}$ and $(P_N)_{.,h}$ may have non-zero entries is position i. For all other positions $l = 1, \ldots, m+n+\lambda$ and $l \neq i$ either $(P_B^{-1})_{i,l} = 0$ or $(P_N)_{l,h} = 0$. This implies

$$\bar{p}_{ih} = (P_B^{-1})_{i,.} \cdot (P_N)_{.,h} = \begin{cases} -\frac{1}{2}, & \text{if } P_{i,h} = -\frac{1}{2} \\ 0, & \text{if } P_{i,h} = 0. \end{cases}$$

For the case that $(P_N)_{.,h} = e_{.,m+j}$ where $s_j^* = 0$, position $m+j$ is the only position where both $(P_B^{-1})_{i,.}$ and $(P_N)_{.,h}$ may have a non-zero entry. This means, $\bar{p}_{ih} = (P_B^{-1})_{i,.} \cdot (P_N)_{.,h} = \frac{1}{2}$ for all non-basic s variables corresponding to the basic x variables in row i. The index set S is the set of indices of the basic x variables in row i. Let N^x and N^s be a partition of the index set of the columns of P_N, i.e., $N = N^x \cup N^s$, such that N^x is the index set of non-basic x variables in row i whereas N^s is the index set of non-basic s variables corresponding to

the basic x variables in row i. We can write the Gomory cut as

$$\begin{aligned}
&\sum_{h\in N} \left(\bar{p}_{ih} - \lfloor \bar{p}_{ih} \rfloor\right) y_h \geq \frac{1}{2} \\
&\Leftrightarrow \sum_{j\in N^x} \left(-\frac{1}{2} - \left\lfloor -\frac{1}{2} \right\rfloor\right) x_j + \sum_{j\in N^s} \left(\frac{1}{2} - \left\lfloor \frac{1}{2} \right\rfloor\right) s_j \geq \frac{1}{2} \\
&\Leftrightarrow \sum_{j\in N^x} \frac{1}{2} x_j + \sum_{j\in N^s} \frac{1}{2} s_j \geq \frac{1}{2}
\end{aligned} \tag{6.15}$$

It can be verified that N^x is equal to the set $N_i \setminus S$ and N^s is equal to the set S. Thus

$$\Leftrightarrow \sum_{j\in N_i\setminus S} x_j + \sum_{j\in S} s_j \geq 1. \tag{6.16}$$

$$\Leftrightarrow \sum_{j\in N_i\setminus S} x_j + \sum_{j\in S} (1 - x_j) \geq 1. \tag{6.17}$$

Since inequality (6.17) is the FS inequality obtained from the configuration $S = \left\{j \in N_i \mid x_j^* = 1\right\}$, the proof is concluded. □

Given an optimal solution of RIPD, (x^*, z^*) with $x_j^* \in \{0, 1\}$ for all $j \in J$ and $z_i^* \in \mathbb{R} \setminus \mathbb{Z}$ for at least one $i \in I$ we can efficiently derive Gomory cuts with the cut generation algorithm 1 (CGA1).

Cut generation algorithm 1 (CGA1)

Input: (x^*, z^*) such that x^* integral, z^* non-integral.
Output: Gomory cut(s).

1: **for** $i = 1$ to m **do**
2: **if** $k_i = 2z_i^*$ is odd **then**
3: Set configuration $S := \left\{j \in N_i \mid x_j^* = 1\right\}$.
4: Construct constraint (6.2).
5: **end if**
6: **end for**
7: Terminate.

This algorithm has a computational complexity of $O(md_c^{max})$ because at most m values have to be checked until a violated parity-check constraint is identified and $O(d_c^{max})$ is the complexity of constructing (6.2). We determine the violated parity-checks using the indicator variables z, i.e., a parity check i is violated if the z_i^* value is non-integral. Having identified a violated

parity-check constraint i (if there exists any) we construct (6.2) easily by setting the coefficient of x_j for $\{j \in N_i : x_j^* = 1\}$ to $+1$, the coefficient of x_j for $\{j \in N_i : x_j^* = 0\}$ to -1 and $|S| = k_i$.

Next we consider the situation that $0 < x_j^* < 1$ for some $j \in J$. Although it is still possible to derive a Gomory cut, CGA1 is not applicable since Theorem 6.2 holds only for integral x^*. For non-integral x^* we propose the cut generation algorithm 2 (CGA2). The idea of this algorithm is motivated by Proposition 6.1 and Proposition 6.3.

Proposition 6.3. *Let x^* be a non-integral optimal solution of relaxed integer programming decoding (RIPD) and $x^* \in conv(C_i)$. Then there are at least two indices $j, k \in N_i$ such that $0 < x_j < 1$ and $0 < x_k < 1$. In other words check node i cannot be adjacent to only one non-integral valued variable node.*

Proof. If $x^* \in \text{conv}(C_i)$ then it can be written as a convex combination of two or more extreme points of $\text{conv}(C_i)$. Next we make use of an observation given in the proof of Proposition 1 in [19]. Assume that check node i is adjacent to exactly one non-integral variable node. This implies that there are two or more extreme points of $\text{conv}(C_i)$ which differ in only one bit. Extreme points of $\text{conv}(C_i)$ differ however, in at least two bits since they all satisfy parity-check i which contradicts the assumption. □

A given binary linear code C can be represented with some alternative, equivalent parity-check matrix which we denote by $\hat{H}$. Any such alternative parity-check matrix for C is obtained by performing elementary row operations on H. Proposition 6.1 is valid for any $\hat{H}$. Likewise Proposition 6.3 holds as well for the parity-check nodes $i \in \{1, \ldots, m\}$ of the Tanner graph representing $\hat{H}$. Note that the rows of $\hat{H}$ may also be interpreted as redundant parity-checks with respect to parity-check matrix H. Hence, FS inequalities derived from the rows of an alternative parity-check matrix are RPC inequalities.

Our algorithm uses the following two ideas based on the cardinality of $\mathcal{A}_i := \{j \in J : \hat{H}_{i,j} = 1, 0 < x_j^* < 1)\}$. In other words, $\mathcal{A}_i$ is the index set of non-integral valued x components of the optimal solution $(x^*, z^*) \in \mathbb{R}^{n+m}$ in $\hat{H}_{i,.}$. We distinguish the cases where $|\mathcal{A}_i| = 1$ and $|\mathcal{A}_i| > 1$.

Consider the case where $|\mathcal{A}_i| = 1$. Given a non-integral optimum x^* of RIPD, in CGA2 we search for a parity-check which is adjacent to exactly one non-integral valued variable node. If we find such a parity-check we know due to Proposition 6.3 that x^* cannot be in the convex hull of this particular parity-check. Furthermore due to Proposition 6.1 there exists an RPC inequality which cuts off x^*. This observation is elaborated in the following theorem.

Theorem 6.4. *Let $(x^*, z^*) \in \mathbb{R}^n \times \mathbb{R}^m$ be the optimal solution of the current RIPD formulation such that x^* is non-integral. Let $\hat{H}$ be a parity-check matrix representing*

code C. Suppose there exists $i \in I$ such that x_j^* is non-integral for exactly one $j \in \hat{N}_i$. Let $k_i = \left|\{h \in \hat{N}_i | x_h^* = 1\}\right|$. Then, if k_i is odd, the inequality

$$\sum_{h \in \hat{N}_i:\, x_h^*=1} x_h - x_j - \sum_{h \in \hat{N}_i : x_h^*=0} x_h \leq k_i - 1, \tag{6.18}$$

or, if k_i is even, the inequality

$$\sum_{h \in \hat{N}_i:\, x_h^*=1} x_h + x_j - \sum_{h \in \hat{N}_i : x_h^*=0} x_h \leq k_i \tag{6.19}$$

is a valid inequality which is violated by x^*.

Proof. We have to show that:

1. For k_i odd [even] the inequality (6.18) $[(6.19)]$ is violated by x^*.
2. For k_i odd [even] the inequality (6.18) $[(6.19)]$ is satisfied for all $x \in C$.

The i^{th} row of the reconstructed matrix $\hat{H}$ is obtained by performing elementary row operations in $GF(2)$ on the rows of the original H matrix. Therefore it holds that $\hat{H}_{i,\cdot} x = 0 \bmod 2$ for all $x \in C$. We show the proof for k_i odd. If k_i is even the proof is analogous.

1) Let k_i be an odd number. For x^*, since $0 < x_j^* < 1$ the left hand side of (6.18) is larger than the right hand side, thus x^* violates (6.18).

2) Suppose k_i is odd and $x \in C$. Our aim is to show that (6.18) is satisfied by x. First we define

$$\delta_i(x) = \sum_{j \in \hat{N}_i} x_j.$$

We define $a_j \in \{-1, +1\}$ such that we can rewrite (6.18) as follows.

$$\sum_{j \in \hat{N}_i} a_j x_j \leq k_i - 1 \text{ where } a_j \in \{-1, 1\}. \tag{6.20}$$

We also define the index sets

$$S^+ = \{j \in \hat{N}_i : a_j = 1\} \text{ with } \left|S^+\right| = k_i \text{ and,}$$
$$S^- = \{j \in \hat{N}_i : a_j = -1\} \text{ with } \left|S^-\right| = \left|\hat{N}_i\right| - k_i.$$

Case 1 $\delta_i(x) \leq k_i - 1$:

$$\sum_{j \in \hat{N}_i} a_j x_j \leq k_i - 1 \text{ is fulfilled.}$$

Case 2 $\delta_i(x) \geq k_i + 1$: At most k_i of indices $j \in \hat{N}_i$ where $x_j = 1$ can be in S^+. Thus there is at least one index $j \in \hat{N}_i$ with $x_j = 1$ in S^-. Consequently

$$\sum_{j \in \hat{N}_i} a_j x_j \leq k_i - 1.$$

Case 3 $\delta_i(x) = k_i$: If there is at least one index $j \in S^-$ with $x_j = 1$ then

$$\sum_{j \in \hat{N}_i} a_j x_j \leq k_i - 1.$$

Otherwise all $j \in \hat{N}_i$ with $x_j = 1$ are in S^+. Then for row i, $\hat{H}_{i,.}x = 1 \bmod 2$ since k_i is odd. This yields $x \notin C$, a contradiction. □

If $|\mathcal{A}_i| = 1$, (6.18) or (6.19) cuts off the non-integral optimum. However, it is possible that each row $i \in I$ of $\hat{H}$ has at least two $j \in \hat{N}_i$ such that x_j^* is non-integral. Next we consider some special cases where $|\mathcal{A}_i| > 1$. These cases are specified in Proposition 6.5 and Proposition 6.6. Let x^* be non-integral and j^{max} denote the index of the maximum non-integral component of x^* in $\mathcal{A}_i$, i.e., $x_{j^{max}}^* := \max\{x_j^* : j \in \mathcal{A}_i\}$.

Proposition 6.5. *Let* $k_i = \left|\{h \in \hat{N}_i | x_h^* = 1\}\right|$. *If* k_i *is odd and* $\sum_{j \in \mathcal{A}_i} x_j^* < 1$, *then the inequality*

$$\sum_{h \in \hat{N}_i:\, x_h^* = 1} x_h - \sum_{j \in \mathcal{A}_i} x_j - \sum_{h \in \hat{N}_i : x_h^* = 0} x_h \leq k_i - 1, \tag{6.21}$$

is a valid cut which cuts off x^*.

Proof. For $i = 1, \ldots, m$, $\hat{H}_{i,.}$ is a dual codeword, or redundant parity-check. Thus $\hat{H}_{i,.}x \equiv 0 \bmod 2$ for all $x \in C$ and inequality (6.21) is valid for C (see proof of Theorem 6.4, part 2). It is assumed that the term $\sum_{j \in \mathcal{A}_i} x_j^*$ is less than 1. Thus the sum of the terms on the left hand side of inequality (6.21) is greater than $k_i - 1$ and x^* violates inequality (6.21). □

Proposition 6.6. *Let* $k_i = \left|\{h \in \hat{N}_i | x_h^* = 1\}\right|$. *If* k_i *is even and* $x_{j^{max}}^* > \sum_{j \in \mathcal{A}_i \setminus \{j^{max}\}} x_j^*$, *then the inequality*

$$\sum_{h \in \hat{N}_i:\, x_h^* = 1} x_h + x_{j^{max}} - \sum_{j \in \mathcal{A}_i \setminus \{j^{max}\}} x_j - \sum_{h \in \hat{N}_i:\, x_h^* = 0} x_h \leq k_i, \tag{6.22}$$

is a valid cut which cuts off x^*.

Proof. The proof of the validity of inequality (6.22) is analogous to the proof of Proposition 6.5. Note that the cardinality of the set $\{h \in \hat{N}_i : x_h^* = 1\} \cup \{j^{max}\}$ is an odd number. It remains to show that x^* violates (6.22). Since $x_{j^{max}}^* > \sum_{j \in \mathcal{A}_i \setminus \{j^{max}\}} x_j^*$ the sum of the terms on the left hand side of inequality (6.22) is greater than k_i and x^* violates inequality (6.22). □

Searching for all RPC cuts is intractable in general. Hence, we propose the construct-$\hat{H}$ algorithm which facilitates the search for RPC cuts. We transform matrix H into an equivalent matrix $\hat{H}$ by elementary row operations (addition is performed in $GF(2)$). Our aim is to represent code C with an alternative parity-check matrix $\hat{H}$, so that in row $\hat{H}_{i,\cdot}$ there exists exactly one $j \in \hat{N}_i$ where x_j^* is non-integral. Construct-$\hat{H}$ algorithm attempts to convert columns j of H with $x_j^* \notin \mathbb{Z}$ into unit vectors. Note that at most m columns of H are converted.

Construct-$\hat{H}$ algorithm

Input: (x^*, z^*) such that x^* non-integral

Output: $\hat{H}$.

```
Set l = 1.
for j = 1 to n do
   if x_j^* ∈ (0,1) then
      if l ≤ m then
         Do elementary row operations until H_{l,j} = 1 and H_{i,j} = 0 for all
         i ∈ {1,...,m} \ {l}.
         Set l = l + 1.
      end if
   end if
   if l > m then
      Terminate.
   end if
end for
Terminate.
```

$\hat{H}$ can be obtained with a complexity of $O(m^2 n)$. Construct-$\hat{H}$ algorithm is useful in the following sense. Suppose $i \in I$ is a check node adjacent to exactly one variable node $j \in J$ such that x_j^* is non-integral. If $\hat{H}$ has such a row i then due to Theorem 6.4 an RPC cut which cuts off the fractional optimal solution can be derived. If no such row exists, then we use Propositions 6.5 and 6.6 to generate RPC cuts. These theoretical findings are implemented in cut generation algorithm 2.

The complexity of CGA2 is $O(mn)$ since in the worst case each entry of $\hat{H}$ has to be visited once.

Cut generation algorithm 2 (CGA2)

Input: $(x^*, z^*) \in \mathbb{R}^{n+m}$ such that x^* non-integral, $\hat{H}$.
Output: New FS inequality or error.

1: **for** $i = 1$ to m **do**
2: Calculate k_i, $\mathcal{A}_i$, j^{max}.
3: **if** $|\mathcal{A}_i| = 1$ **then**
4: Construct (6.18) [(6.19)] for k_i odd [even], terminate.
5: **else if** $|\mathcal{A}_i| \geq 2$, k_i odd, $\sum_{j \in \mathcal{A}_i} x_j^* < 1$ **then**
6: Construct (6.21), terminate.
7: **else if** $|\mathcal{A}_i| \geq 2$, k_i even, $x_{j^{max}}^* > \sum_{j \in \mathcal{A}_i \setminus \{j^{max}\}} x_j^*$ **then**
8: Construct (6.22), terminate.
9: **end if**
10: **end for**
11: If no cuts are found, output error.

Proposition 6.7. *The complexity of CGA2 is* $O(mn)$.

We are now able to formulate our separation algorithm called separation algorithm decoding (SAD). In the first iteration, x^* can be found by applying hard decision decoding, e.g., set x_j to 1 for all $j \in J$ such that $c_j < 0$. In all of the following iterations RIPD does not necessarily have an optimal solution with integral x^*. If the vector (x^*, z^*) is integral then the optimal solution to IPD is found. If x^* is integral but z^* is non-integral we apply CGA1 to construct FS inequalities. Although adding any FS inequality suffices to cut off the non-integral solution (x^*, z^*), we add all FS inequalities induced by all non-integral z_i based on the argument that they may be useful in future iterations. If x^* is non-integral first Construct-$\hat{H}$ algorithm is employed. Then we check in CGA2 if there exists a row $\hat{H}_{i,:}$ such that Theorem 6.4, Proposition 6.5, or Proposition 6.6 holds. In this case there exists an RPC cut which cuts off x^*. If such a row does not exist, then CGA2 outputs an error. In $\hat{H}$ there may exist several rows from which we can derive new RPC cuts. If this is true, we add all new RPC cuts to the formulation RIPD with the same reasoning as before. SAD stops if either (x^*, z^*) is integral, which leads to an ML codeword, or CGA2 returns an error, which means no further cuts can be found.

Two strategies which may be used in the implementation of SAD are:

1. Add all valid cuts which can be obtained in one iteration.

2. Add only one of the valid cuts which can be obtained in one iteration.

There is a trade-off between strategies 1 and 2, since strategy 1 means fewer iterations with large LP problems and strategy 2 means more iterations with smaller LP problems. We empirically tested strategies 1 and 2 on various

Separation algorithm decoding (SAD)
Input: Vector of log-likelihood ratios λ, matrix H.
Output: Current optimal solution x^*.

1: Solve RIPD.
2: **if** (x^*, z^*) is integral **then**
3: Terminate.
4: **else**
5: **if** x^* is integral **then**
6: Call CGA1.
7: Add the constraints to formulation, go to 1.
8: **else**
9: Call construct-$\hat{H}$ algorithm.
10: Call CGA2.
11: **if** a cut is found **then**
12: Add the constraints to formulation, go to 1.
13: **else**
14: Terminate.
15: **end if**
16: **end if**
17: **end if**

LDPC and BCH codes. For all codes we tested, strategy 1 outperformed strategy 2 in terms of running time and error-correcting performance.

The relationship between our approach and other approaches based on RPC cuts (e.g. [27], [63]) can be described as follows. The latter approaches improve LPD by first investigating the set of FS inequalities derived from the original parity-check matrix. Because the polytope which can be obtained from the dual code is tighter than the fundamental polytope in a second step, these approaches try to find candidates for cuts resulting from other dual codewords by some strategies.

In SAD, we do not elaborate on the satisfaction of all FS inequalities of the original parity-check matrix. This has a positive effect on the overall complexity of our improved LP decoding approach. Nevertheless, CGA1 takes into account a subset of these inequalities. With CGA2 we try to derive cuts from dual codewords which are not necessarily limited to the rows of the original parity-check matrix. Therefore, we alter a possible parity-check matrix by the construct-$\hat{H}$ algorithm and check for fulfillment of the conditions explained above. If some of these conditions are satisfied for some rows of the alternative parity-check matrix, then the corresponding inequality is proved to be a cut. Still, there are some similarities between these approaches such as the utilization of the same family of cuts (forbidden set type of inequalities) or the efficient generation of RPC cuts.

6.4 Numerical results

The purpose of this section is twofold: first, we compare the error-correcting performance of our separation algorithm with LPD [27], sum-product decoding, and the reference curve resulting from MLD. MLD curve is obtained by modeling and solving IPD using CPLEX 12.0 [1] as an IP solver. To enable the comparison of SAD with other improved LPD approaches, e.g., [11], [19], [63], the same or similar codes as considered in the literature are used in our experiments. Second, we analyze the complexity and the behavior of our algorithm by collecting statistics of the average number of cuts inserted and iterations performed for increasing block length. The codes considered are $(3, 6)$-regular LDPC codes with $n = 40, 80, 160, 200, 400$ and $(31, 10)$, $(63, 39)$, $(127, 99)$, $(255, 223)$ BCH codes. For all tests, transmission over the BIAWGNC is assumed. The frame error rates are calculated by counting 100 erroneous blocks. LPD via FS inequalities introduced in [27] cannot be used for high density codes since the number of constraints is exponential in the check node degree. This causes a prohibitive usage of memory in the phase of building the LP model. The adaptive approach of [63] overcomes this shortcoming and yet performs as LPD. Therefore, we used this method in the comparison of algorithms.

The performance of the four methods mentioned above was measured on the $(3, 4)$-regular LDPC codes with block length 100, 200, and Tanner's $(155, 64)$ group structured LDPC code [65]. The Frame Error Rate (FER) against signal to noise ratio (SNR) measured as E_b/N_0 is shown in Figures 6.1, 6.2, and 6.3. We used 100 iterations for SPD of $(3, 4)$-regular LDPC codes and Tanner's $(155, 64)$ LDPC code. For the $(3, 4)$-regular LDPC code of length 100 and Tanner's $(155, 64)$ LDPC code our algorithm approaches MLD performance up to $0.5dB$. For the $(3, 4)$-regular LDPC code of length 200, we were not able to compute the ML curve in reasonable time. The performance of SPD and LPD is very similar. For all three codes, the error-correcting performance of SAD is superior to SPD.

To demonstrate that our algorithm also performs well for dense codes, we selected the $(63, 39)$ BCH code and the $(127, 99)$ BCH code for our tests. The results for these codes are shown in Figure 6.4 and Figure 6.5. SPD does not work well for this type of codes due to the dense structure of their parity-check matrix. Our approach is one of the first attempts (see [20]) to decode dense codes using mathematical programming. The results obtained by our algorithm are substantially better than the results obtained by LPD and are only slightly worse than MLD.

The improvement of error-correcting performance comes at the cost of increased complexity, i.e., more cuts are inserted and more and larger LPs have to be solved. To study this trade-off between performance gain and increased complexity of SAD and to relate it to LPD, Table 6.1 shows a comparison of

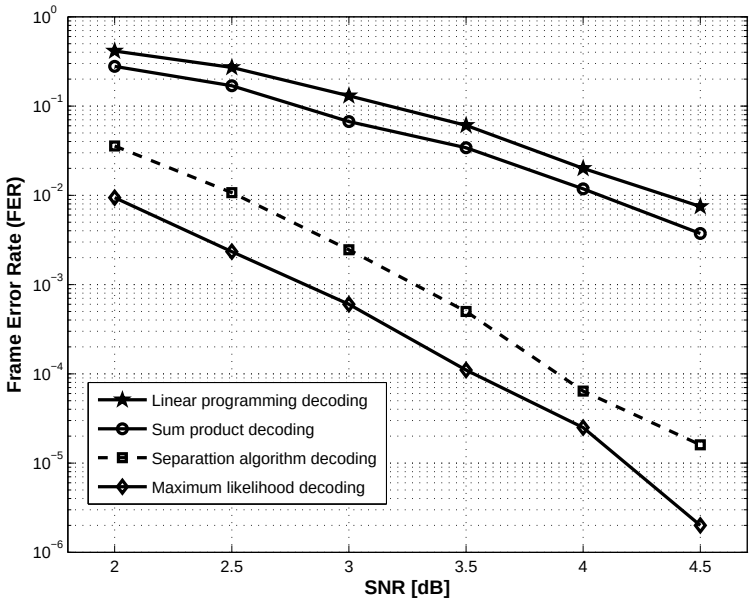


Figure 6.1: Decoding performance of (3,4)-regular LDPC code with block length 100.

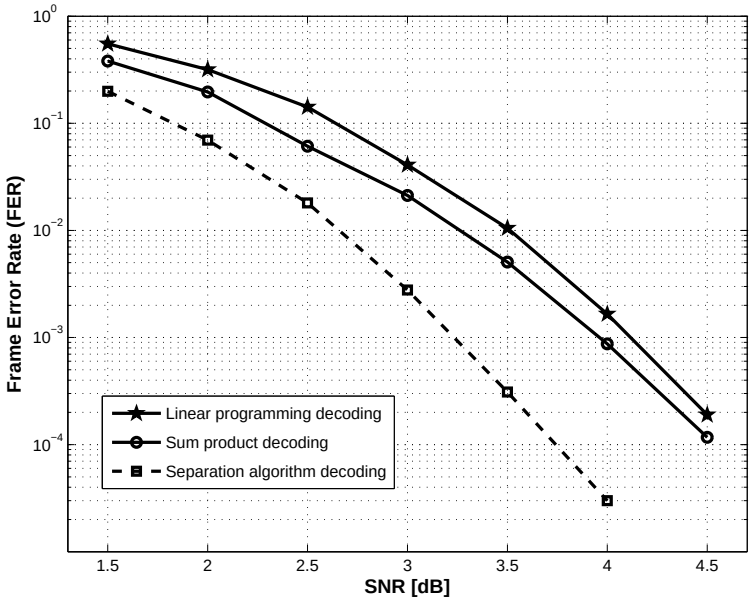


Figure 6.2: Decoding performance of (3,4)-regular LDPC code with block length 200.

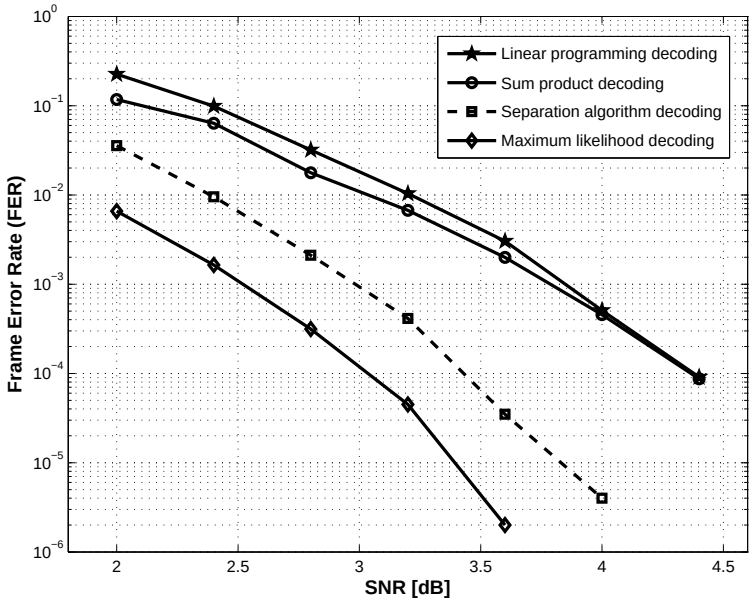


Figure 6.3: Decoding performance of Tanner's $(155, 64)$ LDPC code.

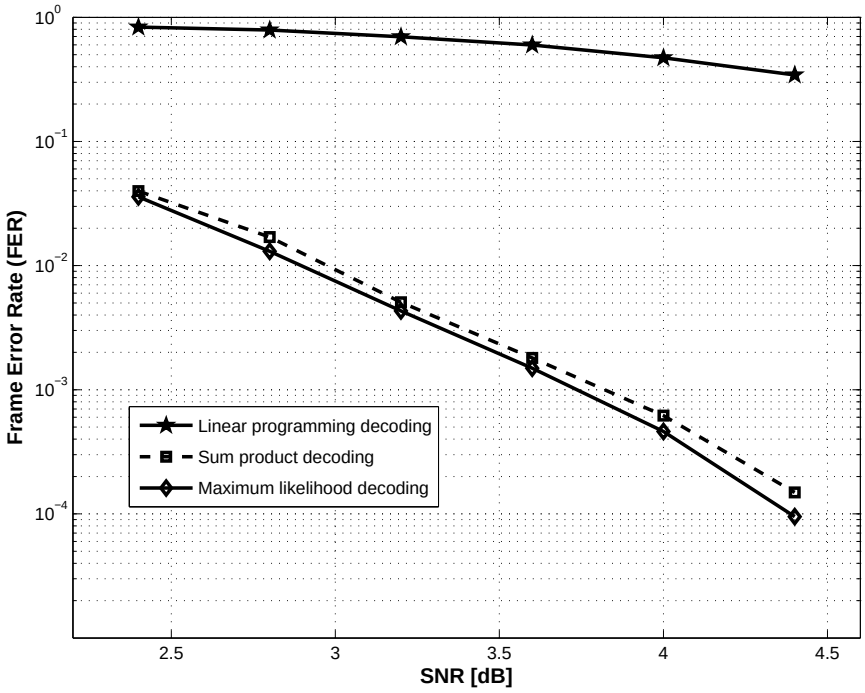


Figure 6.4: Decoding performance of the (63,39) BCH code.

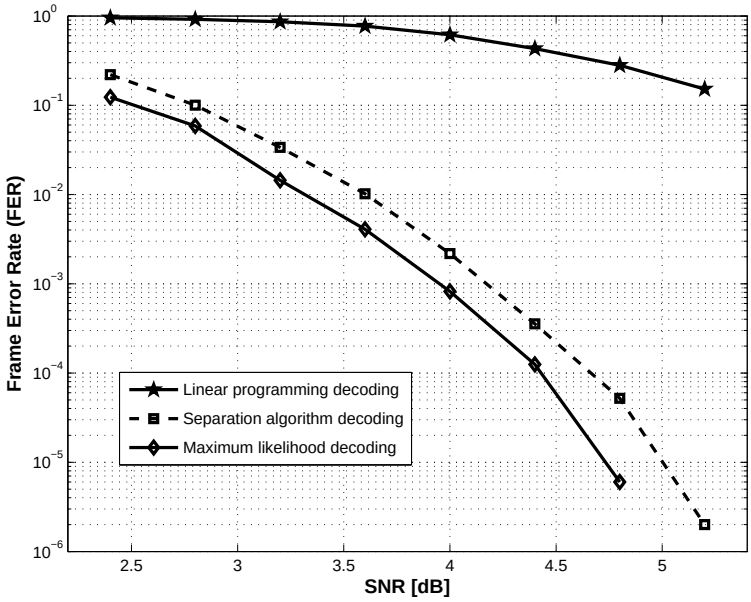


Figure 6.5: Decoding performance of the (127,99) BCH code.

these two methods with respect to the average number of iterations (i.e., number of linear programs solved) and the average number of cuts. These average values are found by solving 500 instances generated for the SNR value of 1.5dB. This low SNR value is chosen, since finding the ML codeword is more challenging for low SNR values due to the fact that the objective function coefficients become less reliable.

Compared to LPD, SAD needs more iterations and cuts to converge to an optimal solution. Observe that for codes with shorter block length, only the average number of cuts differs significantly. This can be explained by the construct-$\hat{H}$ algorithm which tries to modify a possible parity-check matrix such that a cut can be introduced for each row. As the block length increases (cf. (200,100) LDPC and (127,99) BCH in Table 6.1) the search space gets larger. Hence, the average number of cuts inserted and LPs solved increases to ten times that of LPD. The decoding performance of our approach is not affected by the check node degree. In fact, the application of the Construct-$\hat{H}$ algorithm to LDPC codes might lead to dense matrices.

An interesting question relates to the generation of cuts of a certain type in various phases of the decoding process. The distribution of cuts generated by CGA1 (employed when z^* values are non-integral) and CGA2 (employed when x^* values are non-integral), respectively, at different stages of SAD is presented in Figure 6.6. To collect the number of cuts generated by each of the two cut generation algorithms for the $(63, 39)$ BCH code, 1000 instances

Codes	LP decoding		Separation algorithm decoding	
	Cuts	Iterations	Cuts	Iterations
(40,20) LDPC	14.8	4.2	25.9	4.4
(80,40) LDPC	33.8	5.7	103.4	7.6
(160,80) LDPC	74.9	7.2	397.0	16.3
(200,100) LDPC	98.1	7.7	612.1	21.4
(400,200) LDPC	210.4	9.1	3023.0	72.7
(31,10) BCH	8.7	3.4	20.5	5.4
(63,39) BCH	25.2	4.3	143.2	13.9
(127,99) BCH	28.8	4.3	323.7	29.6
(255,223) BCH	32.2	4.2	416.2	34.6

Table 6.1: Average number of cuts and iterations in the adaptive implementation of LPD and SAD.

are solved. Figure 6.6 illustrates the distribution at a lower SNR of 1.5dB and a higher SNR of 4.0dB. The black and gray boxes represent the cumulative number of cuts generated by CGA1 and CGA2, respectively. The total number of iterations needed to solve an instance is divided into 3 phases: a starting phase, a middle phase, and an ending phase. For each of the 3 phases, the cuts generated by CGA1 and CGA2 are shown. To illustrate this, suppose that an instance is solved in $3T$ iterations. Then, the distribution of cuts in iterations 1 to T are shown under "Phase 1" in the figure. For both SNR, CGA1 is applied in the early iterations of SAD. At the end of the algorithm while converging to the optimal solution CGA2 is applied most of the time. Thus, it can be concluded that the complexity of SAD is dominated by construct-$\hat{H}$ algorithm.

6.5 Extensions of separation algorithm decoding

Given a non-integral optimal solution $(x^*, z^*) \in \mathbb{R}^{n+m}$ such that $0 < x^*_j < 1$ for some $j \in J$, if CGA2 is unsuccessful, i.e., no cuts are found, then the attempt to generate the ML codeword fails. The rate of decoding failures determines the performance of the decoding algorithm. In this section we extend SAD to improve its performance by some further valid cut search procedures.

Let (x^*, z^*) be a non-integral optimal solution of RIPD for which CGA2 returns an error. In this case we suggest the cut generation algorithm 3 (CGA3) where FS inequalities are searched after a sort operation. In CGA3 we transform the matrix $\hat{H}$ into an equivalent matrix $\bar{H}$ again by performing elementary row operations in GF(2). Let Π be a permutation of $J = \{1, \ldots, n\}$ such that $x^*_{\Pi(1)} \geq x^*_{\Pi(2)} \geq \ldots \geq x^*_{\Pi(n)}$. The motivation for constructing $\bar{H}$ is to obtain an alternative parity-check matrix in which the columns $\Pi(1), \ldots, \Pi(m)$

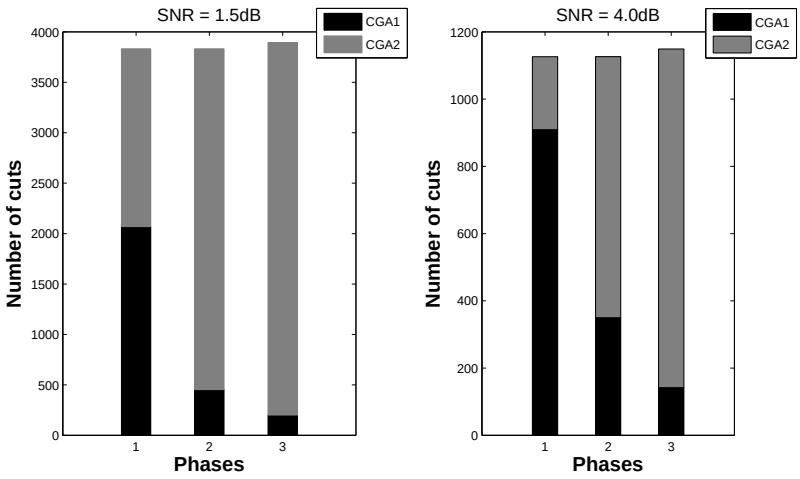


Figure 6.6: Distribution of cuts created by CGA1 and CGA2 for the (63,39) BCH code, SNR=1.5dB, 4.0dB.

of $\bar{H}$ correspond to unit vectors. Note that the columns $\Pi(1), \ldots, \Pi(m)$ are not necessarily linearly independent. Nevertheless we first concentrate on the case that these columns are linearly independent. Then $\bar{H}_{ij} = 1$ for $j = \Pi(i)$, $i = 1, \ldots, m$. Moreover, if $\bar{H}_{ij} = 1$ for $j \neq \Pi(i)$ then $j \in \mathcal{B}_i$ where $\mathcal{B}_i \subseteq \{\Pi(m+1), \ldots, \Pi(n)\}$, i.e., $\mathcal{B}_i$ is the index set of variable nodes other than $\Pi(i)$ in the neighborhood of check node i. The interpretation is that x^*_j is relatively small if $j \in \mathcal{B}_i$ and $x^*_{\Pi(i)}$ is relatively large. Therefore it may be the case that

$$x^*_{\Pi(i)} > \sum_{j \in \mathcal{B}_i} x^*_j. \tag{6.23}$$

It follows that the forbidden set inequality

$$x_{\Pi(i)} - \sum_{j \in \mathcal{B}_i} x_j \leq 0 \tag{6.24}$$

cuts off x^*. Note that inequalities of type (6.24) are also used in the algorithm proposed in [53] to improve the fractional distance.

Proposition 6.8. *Let $(x^*, z^*) \in \mathbb{R}^{n+m}$ be a non-integral optimal solution of RIPD. If condition (6.23) is satisfied, then inequality (6.24) cuts off (x^*, z^*) and it is a valid cut.*

Proof. The proof of the validity of Inequality (6.24) is analogous to the proof of Proposition 6.5. It remains to show that x^* violates (6.24). Condition (6.23) implies that x^* violates inequality (6.24). □

Cut generation algorithm 3 (CGA3)

Input: Non-integral x^*, $\hat{H}$.
Output: New FS inequality or error.

1: Transform $\hat{H}$ to $\bar{H}$.
2: **for** $i = 1$ to m **do**
3: **if** $\bar{H}_{i,\Pi(i)} = 1$ and $x^*_{\Pi(i)} > \sum_{j \in \mathcal{B}_i} x^*_j$ **then**
4: Construct (6.24), terminate.
5: **end if**
6: **end for**
7: Output error.

Remark 6.9. *If columns $\Pi(1), \ldots, \Pi(m)$ are linearly dependent, then CGA3 is still applicable. However it is less likely that inequality (6.23) is fulfilled.*

The dominating computational operation in CGA3 is to transform matrix $\hat{H}$ into $\bar{H}$. This operation is accomplished in $O(m^2 n)$ which is same as Step 9 of SAD. Thus the worst case complexity of CGA3 does not exceed the worst case complexity of SAD. We refer to SAD enhanced with CGA3 as extended separation algorithm (ESA). This algorithm is given below.

6.5.1 The minimum distance inequalities

In this section we introduce a new type of valid inequalities, called minimum distance inequalities. These inequalities have the advantage that they can be quickly generated from the columns of a parity-check matrix. Minimum distance inequalities employ the minimum distance $d(C)$ of a binary linear code C. Although the problem of finding the minimum distance of a code is in general intractable, for several codes this measure is known, e.g., some codes in the BCH code class. For some codes the minimum distance can be computed by solving the integer programming formulation suggested in Section 4.2. The minimum distance inequalities follow from Theorem 6.10.

Theorem 6.10. *Let $\bar{x} \in C$ and $J = \{1, \ldots, n\}$. Partition J into two sets K and Q such that $j \in K$ if $\bar{x}_j = 1$ and $j \in Q$ if $\bar{x}_j = 0$. Let $\hat{k} \in K$ be an arbitrary index and $w(\bar{x})$ denotes the weight of $\bar{x}$. Then*

$$(1 - w(\bar{x}))x_{\hat{k}} + \sum_{k \in K \setminus \{\hat{k}\}} x_k - \sum_{q \in Q} x_q \leq w(\bar{x}) - d(C) \tag{6.25}$$

is a valid inequality.

Proof. We show that inequalities of type (6.25) are satisfied by all codewords. We consider all possible codeword configurations and show that the maxi-

Extended separation algorithm (ESA)
Input: Vector of log-likelihood ratios λ, matrix H.
Output: Current optimal solution x^*.

Solve RIPD.
if (x^*, z^*) integral **then**
  Terminate.
else
  if x^* integral **then**
    Call CGA1.
    Add the constraints to formulation, go to 1.
  else
    Call construct-$\hat{H}$ algorithm.
    Call CGA2.
    if output is error **then**
      Call CGA3.
      if a cut is found **then**
        Add the constraints to formulation, go to 1.
      else
        Terminate.
      end if
    else
      Add the constraints to formulation, go to 1.
    end if
  end if
end if

mum value which can be obtained on the left hand side of inequality (6.25) is $w(\bar{x}) - d(C)$.

1. For $x = 0$, the inequality (6.25) is satisfied since $w(\bar{x}) - d(C) \geq 0$.
2. For $x = \bar{x}$, the inequality (6.25) is satisfied due to our construction.
3. For any $x \in C$ such that $x_{\hat{k}} = 1$ inequality (6.25) is satisfied. Note that the maximum value we can obtain on the left hand side of (6.25) is $w(\bar{x}) - 1$. If the $\hat{k}^{th}$ bit in a codeword x is 1, i.e., $x_{\hat{k}} = 1$, then the sum of the terms on the left hand side cannot be positive.
4. In the rest of the cases we consider $x \in C$ such that $x_{\hat{k}} = 0$. For any $x \in C$ with $x_k = 1$, for all $k \in K \setminus \{\hat{k}\}$, at least $(d(C) - 1)$ (1 bit differs already since $\bar{x}_{\hat{k}} = 1$ and $x_{\hat{k}}$) bits of x differs from $\bar{x}$ due to the definition of the minimum distance of a code. The indices of these different bits should be in Q. It follows that the maximum value we can obtain on the left hand side of (6.25) can be $(w(\bar{x}) - 1) - (d(C) - 1) = w(\bar{x}) - d(C)$.
5. For any $x \in C$, $x \neq 0$, with $x_k = 0$, for all $k \in K \setminus \{\hat{k}\}$, the sum of the terms on the left hand side of (6.25) is strictly negative.
6. For any $x \in C$ with $x_l = 1$ for all $l \in L$, $L \subset K \setminus \{\hat{k}\}$, $L \neq \emptyset$, the left hand side of (6.25) is at most $|L|$. This implies $x_o = 0$ for all $o \in K \setminus \{\hat{k} \cup L\}$. Thus, x differs from $\bar{x}$ in a total of $(w(\bar{x}) - 1 - |L|)$ bit positions.
   - Case 1: If $w(\bar{x}) - 1 - |L| \geq d(C) - 1$, then w$(\bar{x}) - d(C) \geq |L|$ and (6.25) is satisfied.
   - Case 2: If $w(\bar{x}) - 1 - |L| < d(C) - 1$, then at least $(d(C) - 1) - (w(\bar{x}) - 1 - |L|)$ bits in the set Q have to be 1 due to the definition of the minimum distance of a code. Thus we have to subtract $(d(C) - 1) - (w(\bar{x}) - 1 - |L|)$ from $|L|$. It follows that inequality (6.25) is satisfied since the maximum value we can obtain on the left hand side of (6.25) can only be $w(\bar{x}) - d(C)$.

□

Example: For $(7, 4, 3)$ Hamming code with a parity-check matrix H

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

we arbitrarily choose the codeword $[1, 1, 0, 0, 1, 1, 0]$, and $\hat{k} = 2$. We derive the inequality

$$-3x_2 + x_1 + x_5 + x_6 - x_3 - x_4 - x_7 \leq 1. \tag{6.26}$$

Consider the cost vector $\lambda^T = (2, 2, 5, 2, -3, -1, 4)$ and suppose the 0 codeword is sent. The RIPD optimum is $(x^*)^T = (0.5, 0, 0, 0, 1, 0.5, 0.5)$. Obviously, the minimum distance inequality (6.26) cuts off the non-integral optimal solution. Indeed, it is observed by straightforward checking that the minimum distance type inequalities define facets of the $(8, 4, 4)$ extended Hamming code. Thus, this set of inequalities may be utilized to improve the current version of SAD.

It is important to develop efficient methods to search for minimum distance inequalities which are likely to cut off non-integral optimal solutions of RIPD. The cut generation algorithm 4 (CGA4) introduced below makes use of the observation that linearly dependent columns of a parity-check matrix build a codeword. A minimum distance inequality is likely to cut off the non-integral optimal solution x^* if the positive terms on the left hand side of (6.25) are large and the negative terms are small. Therefore we aim to choose an index $\hat{k}$ for which $x^*_{\hat{k}} = 0$ and collect the largest valued components of x^* in the set K.

Let (x^*, z^*) be a non-integral optimal solution of RIPD. In CGA4 we first transform a possible parity-check matrix $\hat{H}$ into $\bar{H}$ as we have done in CGA3. We again try to obtain unit vectors in the columns corresponding to the indices of maximal x^* components. Let $\Omega \subset J$ be the index set of x variables with $x^*_j = 0$ for $j \in \Omega$. We fix $\hat{k} \in \Omega$ and iteratively vary i between 1 and m in the following: if the entry $\bar{H}_{i,\hat{k}}$ equals 1, then we try to find a column of $\bar{H}$ corresponding to the i^{th} unit vector. If the columns of $\Pi(1), \ldots, \Pi(m)$ are linearly independent, then the i^{th} unit vector is given by the column $\Pi(i)$ of $\bar{H}$ due to our construction. If this is the case, then the index $\Pi(i)$ is assigned to set K.

The remaining indices form the set Q. Using K and Q we construct a codeword $\bar{x}$ with $\bar{x}_k = 1$ for $k \in K$ (note that $\hat{k} \in K$) and $\bar{x}_q = 0$ for $q \in Q$. The weight of $\bar{x}$ is given by $w(\bar{x}) = \sum_{i \in I} \bar{H}_{i,\hat{k}} + 1$. If

$$\sum_{k \in K} x^*_k > \mathrm{w}(\bar{x}) - d(c) + \sum_{q \in Q} x^*_q, \tag{6.27}$$

then the minimum distance inequality cuts off x^*. We add the constraint to the formulation and resolve RIPD. CGA4 continues until a minimum distance inequality is found or the set Ω is exhausted.

6.6 Conclusion and further research

In this chapter, we proposed an improved LPD method for binary linear block codes. Instead of solving the optimization problem, we solved the separation problem.

The indicator z variables in IPD1 yield an immediate recognition of parity violations and efficient generation of cuts for the case where the current LP

Cut generation algorithm 4 (CGA4)

Input: non-integral x^*, $\hat{H}$.
Output: Minimum distance inequality or error.

1: Transform $\hat{H}$ to $\bar{H}$.
2: Find the set $\Omega \subset J$.
3: **while** $\Omega \neq \emptyset$ **do**
4: Choose $\hat{k} \in \Omega$, construct $\bar{x}$.
5: **if** (6.27) is satisfied **then**
6: Construct (6.25), terminate.
7: **else**
8: Delete ω from Ω.
9: **end if**
10: **end while**
11: Output error.

solution is integral in x variables. We used on the one hand the FS inequalities of [27] which are a subset of all possible Gomory cuts. On the other hand we showed how to generate new cuts based on redundant parity-checks efficiently. It is known that RPC cuts improve LPD via tightening the fundamental polytope. In our approach, once we ensure that Theorem 6.4, Proposition 6.5, or Proposition 6.6 hold for some row $\hat{H}_{i,.}$, $i \in I$, we can immediately find the configuration S and thus the RPC cut.

These theoretical improvements are supported with empirical evidence. Compared to LPD and SPD our algorithm is superior in terms of frame error rate for all the codes we have tested. In contrast to SPD, our approach is applicable to codes with dense parity-check matrices and offers a possibility to decode such codes.

We presented several algorithms to search for violated FS inequalities or RPC cuts in Section 6.3 and Section 6.5. Further algorithms can be found in [63], [62], [79]. More research has to be done on efficient algorithms dedicated to finding FS type inequalities from the dual code. Another important question is the trade-off between the computational effort required to find new valid cuts and the decoding performance.

In general, it is not possible to describe the codeword polytope completely by using only FS type of inequalities. Therefore, valid inequalities and cuts other than FS type inequalities are of great interest. We made a first attempt towards a new class of inequalities, so called minimum distance inequalities, which can be derived from the columns of a parity-check matrix. The minimum distance inequalities are valid cuts and an algorithm to derive minimum distance cuts from alternative code representations is developed. The polyhe-

dral analysis of the codeword polytope should be continued. It is interesting to identify the inequality classes defining the facets of the codeword polytope. The minimum distance inequalities may offer new perspective in this direction.

The worst case complexities of CGA1, CGA2, and Construct-$\hat{H}$ algorithm are stated in Section 6.3. However, the number of calls to CGA1 and CGA2 cannot be estimated hence we were not able to present an analytical overall evaluation of SAD. This remains as an open issue.

In all improved (in terms of error-correcting performance) LP decoding approaches known to us, a general purpose LP solver is used to solve LP subproblems. A special variant of the simplex method developed for the LP decoding problem or interior-point type algorithms may lower the overall complexity of the LP based decoding approaches. The structural properties of the FS inequalities identified in [62] can be used for this purpose.

Chapter 7

Decoding of LTE turbo codes

7.1 Introduction

Typically, turbo codes and LDPC codes are decoded in an iterative manner where probabilistic information is exchanged between corresponding component decoders. This is known as iterative message passing decoding (IMPD). In the last 10 years, linear programming decoding (LPD) introduced by Feldman et al. [27] has become an interesting alternative for decoding of binary linear block codes. LPD is a polynomial time decoding approach with some desirable properties such as the maximum likelihood (ML) certificate property: if LPD outputs a codeword, then it is the ML codeword. Moreover, this approach allows finite length analysis due to the geometry of linear programming (LP).

LPD is first formulated for repeat accumulate (RA) codes in [25]. Then it is applied to low density parity-check (LDPC) codes in [27]. After the introduction of LPD for LDPC codes, most of the research on LPD has focused on LDPC codes. The error correcting performance of LPD is inferior to sum-product decoding (SPD). For turbo-like codes, LPD is typically used for calculating error bounds especially for RA(2) codes (see [32] and references therein). A subgradient algorithm for LPD of RA(2) was proposed in [23].

In this chapter, we propose a new decoding method for linear programming based decoding of LTE turbo codes [1]. From the general scheme IPD8 given in Section 4.2 we derive an integer programming (IP) formulation for LTE turbo codes. Then we study the linear programming (LP) relaxation of the IP formulation. Motivated from constrained shortest path (CSP) problems [38] and equal flow problems [49] we develop a two-step algorithm called best agreeable path decoding (BAPD) which relies on the combinatorial structure of the decoding problem. Relaxing the side constraints by Lagrangian relaxation, yields a shortest path problem which can be efficiently solved. We solve

[1] 3rd generation partnership project, http://www.3gpp.org

the related Lagrangian dual problem and close the duality gap (if it exists) by a k-th shortest path algorithm. Under some assumptions, we can guarantee to output the ML codeword.

This chapter is organized as follows. The graphical representation of LTE turbo codes is explained in Section 7.2. In Section 7.3 we derive the formulation LTEIP from IPD8. The BAPD algorithm is described in Section 7.4. In Section 7.5 the error correcting performance of BAPD is compared to LPD, Log-MAP component decoding, and maximum likelihood decoding (MLD). The chapter is concluded with some further research ideas in Section 7.6.

7.2 Graphical representation of LTE turbo codes

LTE turbo codes are binary linear block codes with block length $n = 3k + 12$ including tail bits where k denotes the number of information bits. LTE turbo codes are constructed from two component convolutional codes, each having 8 states and rate $R = \frac{1}{2}$. The codewords are obtained by combining information bits and parity bits returned by two identical trellis encoders. However, one of the trellises encodes a permuted (interleaved), $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$, version of the information bits. This is illustrated in Figure 7.1. The bit vector $u \in \{0, 1\}^k$ denotes the information bits. The encoder outputs $y^0 = u$, and parity bits $y^1 \in \{0, 1\}^{k+6}$, $y^2 \in \{0, 1\}^{k+6}$ returned by encoder trellis 1 and 2. The overall code rate is approximately[2] $\frac{1}{3}$.

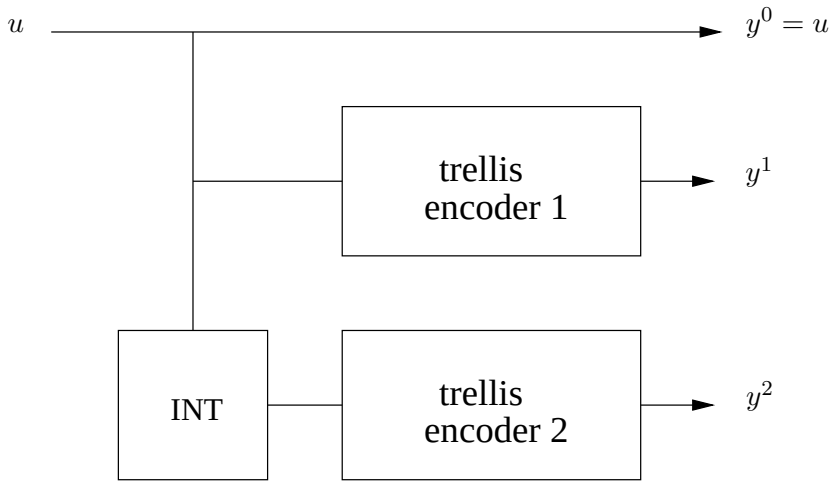


Figure 7.1: Encoder scheme of LTE turbo codes.

Next, the structure of the trellis graph used in encoding/decoding of LTE turbo codes is explained. This trellis graph is an acyclic, directed graph. One

[2] 12 tail bits become negligible as the block length increases.

From state	Information bit	To state	Parity bit
0	0	0	0
0	1	4	1
1	0	4	0
1	1	0	1
2	0	5	1
2	1	1	0
3	0	1	1
3	1	5	0
4	0	2	1
4	1	6	0
5	0	6	1
5	1	2	0
6	0	7	0
6	1	3	1
7	0	3	0
7	1	7	1

Table 7.1: State transition table

segment of the trellis is denoted by t. Each trellis segment corresponds to an information bit. There are two vertex layers in the segments. The vertices model different states. At each vertex of the trellis graph there are at most 2 incoming edges and at most 2 outgoing edges. An edge starts from a state which was left and ends in a state which is entered next. Encoding is done according to some state transition table. For LTE turbo codes discussed in this chapter Table 7.1 is used[3]. Depending on the current state and the value of the information bit, the next state and the related parity bit are read from the state transition table. The starting and ending state is 0. In order to ensure that the trellis terminates (ends in state 0) 3 tail information bits and 3 tail parity bits are used. The edges are labeled by the parity bits returned by the encoder.

Example 7.1. *An example component trellis encoder which takes* 4 *information bits as input and returns a bit sequence of length* 14 *is shown in Figure 7.2. Note that the last* 3 *information bits are tail information bits. At any state, if the information bit is* 0 *then the trellis encoder follows an input-0 edge (solid line), otherwise an input-1 edge (dashed line) is followed. In other words, if in segment* $t \in \{1, \ldots, k\}$ *an input-1 edge is used then* $u_t = y_t^0 = 1$.

Using Table 7.1 and Figure 7.2, it can be verified that an information word 1010 *is encoded to* 10100111100101 *(the path shown in bold) in this component trellis. Bits*

[3]Related polynoms are: $g_0 = 15$ (forward polynom in octal representation) and $g_1 = 13$ (backward polynom in octal representation) see [66].

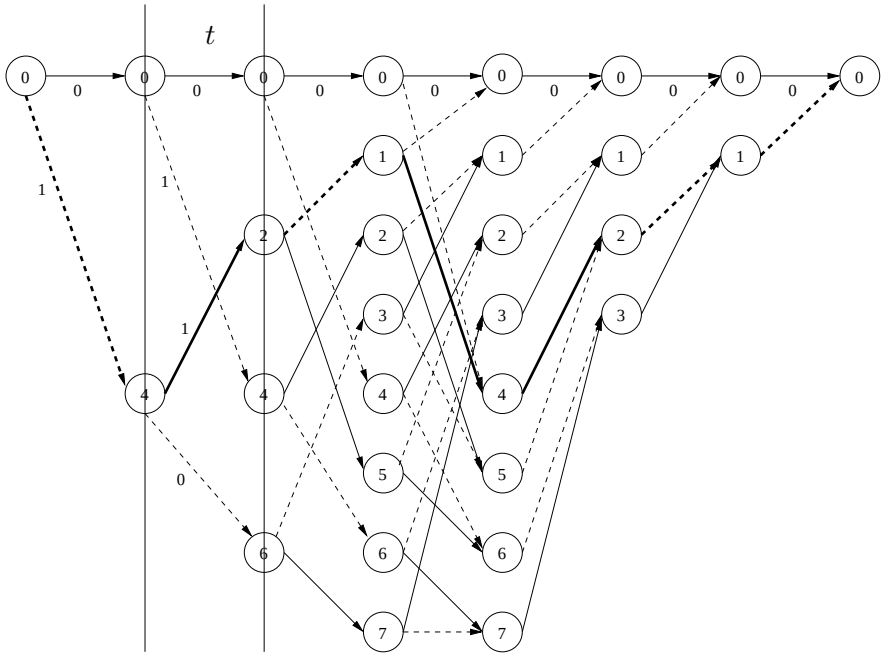


Figure 7.2: Trellis graph (trellis 1) with 8 states. To ensure that the encoder trellis terminates, 3 tail information bits are used. The number of tail parity bits which are produced by 3 tail information bits is also 3.

are distributed as follows: 1010 *are information bits,* 011 *are information tail bits,* 1100 *are parity bits, and* 101 *are parity tail bits.*

The second trellis encoder has the exact same structure as the first one. However it encodes a permuted (interleaved), $\Pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$, version of the information bits.

Example 7.2. *To complete the encoding example given in Example 7.1 we use the same information bits* 1010 *and assume a permutation* $\Pi : \{1, 2, 3, 4\} \to \{4, 3, 2, 1\}$. *Thus the second trellis encoder encodes the information word* 0101. *It can again be verified that* 101001111001011000110100 *is the resulting codeword.*

The distribution of bits in general is illustrated in Figure 7.3.

k info. bits	3 tail info. bits Trellis 1	k parity bits Trellis 1	3 tail parity bits Trellis 1	3 tail info. bits Trellis 2	k parity bits Trellis 2	3 tail parity bits Trellis 2

Figure 7.3: Codeword of length $3k + 12$.

7.3 An IP formulation for LTE turbo codes

Feldman [23] intoduced a general scheme which can be applied to derive IP formulations for turbo-like codes. In Section 4.2 we described IPD8 based on this scheme. The IP formulation, LTEIP, is the application of IPD8 on LTE turbo codes. We refer to Section 4.2.5 for the notation used in the following IP formulation.

We set $|\mathcal{L}| = 3$ in IPD8. From the first leaf, k information bits are output. These bits are denoted by the vector $y^0 \in \{0,1\}^k$. From each of the component trellises, 3 tail bits and $k+3$ parity bits are returned, i.e., $y^1, y^2 \in \{0,1\}^{k+6}$. The permutation used in interleaver is denoted by Π. To keep the notation uniform in this chapter, we denote the trellis graph 1 by $G^1 = (V^1, E^1)$ and the trellis graph 2 by $G^2 = (V^2, E^2)$ where $V$and E are the index sets of vertices and edges, respectively. In this chapter, the binary input additive white Gaussian noise channel (BIAWGNC) is assumed and the vector of log likelihood ratios (LLR) is denoted by $\lambda \in \mathbb{R}^n$ as before.

$$
\begin{aligned}
\text{minimize} \quad & (\lambda^1)^T y^0 + (\lambda^2)^T y^1 + (\lambda^3)^T y^2 && \text{(LTEIP)} \\
\text{subject to} \quad & \sum_{e \in \text{ out}(v^{\text{start},1})} f_e^1 = 1 && \\
& \sum_{e \in \text{ in}(v^{\text{end},1})} f_e^1 = 1 && \\
& \sum_{e \in \text{ out}(v)} f_e^1 = \sum_{e \in \text{in}(v)} f_e^1 && \text{for all } v \in V^1 \setminus \{v^{\text{start},1}, v^{\text{end},1}\} \\
& y_t^0 = \sum_{e \in I_t} f_e^1 && \text{for all } t \in \{1, \ldots, k\} \\
& y_t^1 = \sum_{e \in I_t} f_e^1 && \text{for all } t \in \{k+1, \ldots, k+3\} \\
& y_t^1 = \sum_{e \in O_{t-k-3}} f_e^1 && \text{for all } t \in \{k+4, \ldots, 2k+6\} \\
& \sum_{e \in \text{ out}(v^{\text{start},2})} f_e^2 = 1 && \\
& \sum_{e \in \text{ in}(v^{\text{end},2})} f_e^2 = 1 && \\
& \sum_{e \in \text{ out}(v)} f_e^2 = \sum_{e \in \text{in}(v)} f_e^2 && \text{for all } v \in V^2 \setminus \{v^{\text{start},2}, v^{\text{end},2}\} \\
& y_t^0 = \sum_{e \in I_{\Pi(t)}} f_e^2 && \text{for all } t \in \{1, \ldots, k\} \\
& y_t^2 = \sum_{e \in I_t} f_e^2 && \text{for all } t \in \{k+1, \ldots, k+3\} \\
& y_t^2 = \sum_{e \in O_{t-k-3}} f_e^3 && \text{for all } t \in \{k+4, \ldots, 2k+6\}.
\end{aligned}
$$

Choosing the appropriate formulation for an IP problem is of crucial importance since it affects the computation time and the sizes of the solvable instances. In our numerical experiments we observed that LTEIP performs significantly better than the IP formulations derived from the parity-check matrices of the codes (such formulations were given in Chapter 4). The reason for this is that the network flow constraints provide a tighter relaxation of the convex hull of the codewords than the relaxation resulting from a parity-check matrix [23]. Due to this reason, also LPD of LTE turbo codes is performed using the LP relaxation of LTEIP.

In what follows, we concentrate on the Lagrangian relaxation of LTEIP. For

the ease of notation, LTEIP is rewritten first. Suppose that the second component trellis graph is concatenated to the first one as shown in Figure 7.4. Let $G^c = (V^c, E^c)$ denote the concatenated graph where the set of vertices and edges are denoted by V^c and E^c, respectively. The edge connecting trellis 1 and trellis 2, i.e., $(v^{\text{end},1}, v^{\text{start},2}) \in E^c$, has 0 cost (see Figure 7.4). The consistency constraints induced by the interleaver can be interpreted as: in G^c if a path follows an input-1 edge in segment t it has to follow an input-1 edge in segment $(k + 4 + \Pi(t))$[4], $t \in \{1, \ldots, k\}$.

Definition 7.3. *A $v^{start,1} - v^{end,2}$ path of G^c which satisfy the interleaver consistency constraints is called an agreeable path.*

There is a one-to-one correspondence between codewords and agreeable paths. In our approach, we search for the agreeable path with minimum cost. The cost of an edge $e \in E^c$ is given by the sum of the LLR of related information and parity bits.

Example 7.4. *For example in the first segment of the trellis shown in Figure 7.2, the first input-1 edge (dashed line) has cost $\lambda_{(v',v'')} = \lambda_1 + \lambda_{(k+4)}$. The addition of an interleaver and a second trellis encoder complicates the MLD of LTE turbo codes. The resulting problem can be interpreted as a side constrained shortest path problem (SCSPP).*

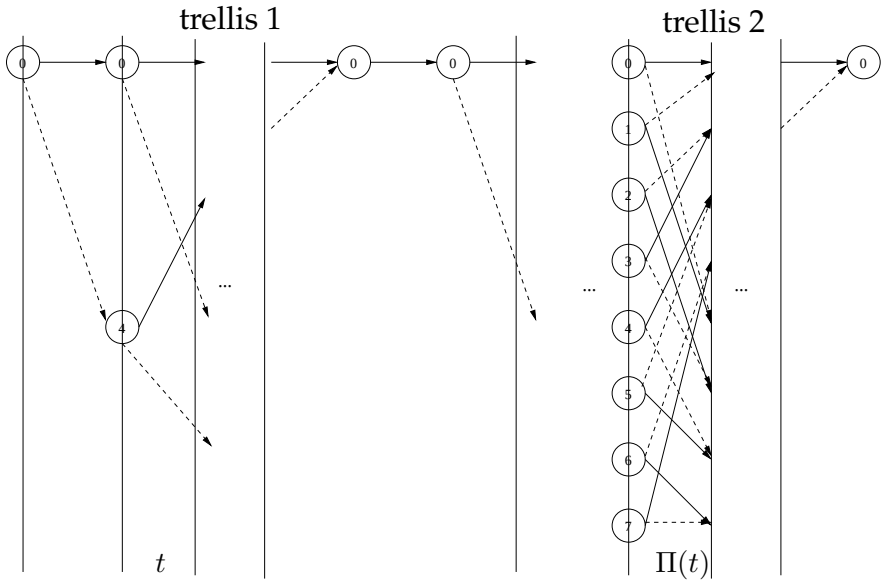


Figure 7.4: Constrained shortest path problem

[4]Segments $k + 1$, $k + 2$, $k + 3$ correspond to information bits used to terminate the encoder trellis 1. Segment $k + 4$ corresponds to the 0 cost edge connecting trellis 1 and 2.

An integer programming formulation of this SCSPP is given below. Again, we refer to the formulation as LTEIP. A shortest path problem is a special case of a network flow problem (see for example [2]) where one unit of flow is sent from the start vertex $v^{\text{start},1} \in V^c$ to an end vertex $v^{\text{end},2} \in V^c$. The variables x_e, $e \in E^c$ model the flows. If $x_e = 1$ then 1 unit of flow is passed on e. The constraints (7.2), (7.3), and (7.4) are conservation of flow constraints, i.e., one unit of flow is sent from start vertex, in all intermediate vertices the sum of flows on incoming edges is equal to the sum of flows on outgoing edges, and finally one unit of flow is received at the end vertex. The interleaver consistency is ensured by the set of constraints (7.5). For $t \in \{1, \ldots, k\}$, I_t is the index set of flow variables on the input-1 edges in segment t.

$$
\begin{aligned}
z = \text{minimize} \quad & \sum_{e \in E^c} \lambda_e x_e && \text{(LTEIP)} \\
\text{subject to} \quad & x_e \in \{0,1\} && \text{for all } e \in E^c && (7.1)\\
& \sum_{e \in \text{ out}(v^{\text{start},1})} x_e = 1 && && (7.2)\\
& \sum_{e \in \text{ out}(v)} x_e = \sum_{e \in \text{in}(v)} x_e && \text{for all } v \in V^c \setminus \{v^{\text{start},1}, v^{\text{end},2}\} && (7.3)\\
& \sum_{e \in \text{ in}(v^{\text{end},2})} x_e = 1 && && (7.4)\\
& \sum_{e \in I_t} x_e = \sum_{e \in I_{k+4+\Pi(t)}} x_e && \text{for all } t \in \{1, \ldots, k\}. && (7.5)
\end{aligned}
$$

Without the constraints (7.5), LTEIP models a shortest path problem on G^c, i.e., the edges $e \in E^c$ for which $x_e = 1$ form the shortest path from $v^{\text{start},1}$ to $v^{\text{end},2}$. We use x to denote the vector of flow variables and corresponding paths. The meaning will be clear from the context.

Constraints (7.2) - (7.4) with the relaxation of integrality constraints in (7.1), i.e., $x_e \in [0,1]$ for all $e \in E^c$, describe an integral polytope which we refer to as the path polytope and denote by X. Thus, the LP-relaxation of LTEIP without the set of constraints (7.5) is equivalent to an LP which has integral optimal solutions. Shortest path problems are usually not solved by general linear programming methods but by combinatorial algorithms (see [2]). Including the interleaver consistency constraints in LTEIP, turns the shortest path problem into an SCSPP problem. Only paths satisfying the interleaver consistency constraints are eligible. These constraints also disturb the structure of the integral polytope described by network flow constraints. The resulting relaxed polytope has in general integral and non-integral vertices.

7.4 A Lagrangian relaxation based algorithm

A solution approach for SCSPP is based on Lagrangian relaxation (LR) by incorporating the side constraints into the objective function and then to solve the Lagrangian dual (LD) problem with a subgradient algorithm. In this section we apply well-known results from integer programming to SCSPP. For theoretical background on Lagrangian relaxation, Lagrangian dual, and subgradient optimization, [55] is referred. This approach is reasonable since the Lagrangian relaxation yields a shortest path problem which can be solved efficiently by combinatorial algorithms. In the LPD literature, relaxing the interleaver consistency constraints by Lagrangian relaxation and solving the Lagrangian dual is considered in [23] for repeat accumulate (RA) codes with rate $R = \frac{1}{2}$. Below we present a Lagrangian relaxation of LTEIP, LR(θ), where $\theta \in \mathbb{R}^k$ is the vector of Lagrangian multipliers, X is the path polytope, and $z_{LR}(\theta)$ is the objective function value.

$$z_{LR}(\theta) = \min_{x \in X} \sum_{e \in E^c} c_e x_e + \sum_{t=1}^{k} \theta_t \left(\sum_{e \in I_t} x_e - \sum_{e \in I_{k+4+\Pi(t)}} x_e \right).$$

For every $\theta \in \mathbb{R}^k$, LR(θ) is a relaxation of LTEIP. Thus, for every fixed θ, $z_{\text{LR}}(\theta) \leq z$, i.e., LR($\theta$) yields a lower bound. In the Lagrangian dual (LD) problem for the LTEIP the θ value maximizing the lower bound (LB) is searched. Thus, we solve

$$z_{LD} = \max_{\theta \in \mathbb{R}^k} z_{LR}(\theta).$$

An iterative solution approach for LD problems is the subgradient algorithm. In this approach, subgradients are iteratively computed and according to the formula

$$\theta^{l+1} = \theta^l + \omega_l s^l. \tag{7.6}$$

Lagrangian multipliers for the next iteration are calculated, where ω_l is a step length and s^l is a subgradient in iteration l. Then, LR(θ) is solved with the updated θ values. With an appropriate choice of the step length ω_l, i.e., $\omega_l > 0$, $\lim_{l \to \infty} \omega_l = 0$, $\sum_{s=1}^{\infty} \omega_l = \infty$, $\sum_{s=1}^{\infty} \omega_l^2 < \infty$, the subgradient algorithm converges.

We compute the subgradient $s^l \in \{-1, 0, 1\}^k$ by

$$s_t^l = \sum_{e \in I_t} x_e^l - \sum_{e \in I_{k+4+\Pi(t)}} x_e^l \text{ for all } t \in \{1, \ldots, k\} \tag{7.7}$$

where x^l denotes the shortest path in iteration l and we choose $\omega^l = \frac{a}{b+cl}$ as the step length where $a, c > 0$ and $b \geq 0$ are constants.

Proposition 7.5. *If some path x^l is an agreeable path then $s^l = 0$ and the path x^l corresponds to the ML codeword.*

Proof. Follows from standard results on Lagrangian duality. □

In practice, subgradient algorithms converge slowly. Thus, we set the maximum number of iterations allowed to l^{max}. Note that the path x^l is not necessarily an agreeable path but it implies a lower bound and we can use it to compute upper bounds (UB) in each iteration as follows. Following the shortest path in trellis 1, an estimation for u can be made. This estimation is found by (7.8) below. If the shortest path follows an input-1 edge in segment t, then $(\hat{u}_1^l)_t = 1$, otherwise $(\hat{u}_1^l)_t = 0$. Analogously, a second estimate can be calculated from trellis 2 by (7.9). We denote this estimation by $\hat{u}_2^l$. A third estimate denoted by $\hat{u}_3^l$ in iteration l is found by averaging the $\hat{u}_1^l$ estimates found in the preceding iterations. This is shown in (7.10). The fourth estimate $\hat{u}_4^l$ is computed analogous to $\hat{u}_3^l$ but based on trellis 2 (see Equation (7.11)).

For all $t \in \{1, \ldots, k\}$:

$$(\hat{u}_1^l)_t = \sum_{e \in I_t} x_e^l. \tag{7.8}$$

$$(\hat{u}_2^l)_t = \sum_{e \in I_{k+4+\Pi(t)}} x_e^l. \tag{7.9}$$

$$(\hat{u}_3^l)_t = \begin{cases} 1, & \text{if } \left[\frac{(l-1)}{l}(\hat{u}_3^{l-1})_t + \frac{1}{l}(\hat{u}_1^l)_t\right] > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{7.10}$$

$$(\hat{u}_4^l)_t = \begin{cases} 1, & \text{if } \left[\frac{(l-1)}{l}(\hat{u}_4^{l-1})_t + \frac{1}{l}(\hat{u}_2^l)_t\right] > 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{7.11}$$

Proposition 7.6. *Let $x^{ag(i)}$, $i = 1, \ldots, 4$, denote four agreeable paths found by encoding $\hat{u}_1^l, \hat{u}_2^l, \hat{u}_3^l, \hat{u}_4^l$.*

$$UB^l = \min_{i=1,\ldots,4} \left\{ \sum_{e \in E^c} \lambda_e x_e^{ag(i)} \right\} \geq \sum_{e \in E^c} \lambda_e x_e^{ML}$$

where x^{ML} is the path corresponding to the ML codeword.

Proof. Follows since each $x^{ag(i)}$, $i = 1, \ldots, 4$, corresponds to a feasible codeword. □

The subgradient algorithm referred to as LDD is given below. While calculating the step length, the values $a = 1$, $b = 0$, and $c = 0.25$ are used. These values were determined by performing numerical experiments.

The complexity of LDD is in $O(|V^c|)$. The dominating operation is computing a shortest path which can be performed generally in $O(|V| + |E|)$ in a

Lagrangian Dual Decoder (LDD)
Input: $G^c = (V^c, E^c)$, λ, l^{max}.
Output: UB, best feasible solution x^{best}.

Set UB$= 0$, LB$= -\infty$, $l = 1$, $\theta^1 = 0$, $x^{best} = 0$.
while $l < l^{max}$ **do**
  Solve LR(θ^l), find $z_{LR}(\theta^l)$, x^l, s^l.
  if s^l=0 **then**
    Construct the ML codeword using x^l.
  else
    LB $= \max\{\text{LB}, z_{LR}(\theta^l)\}$.
    Compute $\hat{u}_1^l$, $\hat{u}_2^l$, $\hat{u}_3^l$, $\hat{u}_4^l$.
    if UB$^l <$ UB **then**
      Set UB=UB^l.
      Find x^{best} using the agreeable path which yields UB.
    end if
    Set $\theta = \theta + \frac{a}{b+cl}s$.
    Set $l = l + 1$.
  end if
end while

directed acyclic graph $G = (V, E)$ [16]. If LDD terminates with $l = l^{max}$, there is a duality gap which we attempt to close by a k-th shortest path algorithm (kSPA) (see [42] and references therein for computing k-th shortest path). We search for the k-th shortest path separately in trellis 1 and 2 with the RECURSIVE ENUMERATION ALGORITHM (REA) given in [42]. In a graph $G = (V, E)$, if the shortest paths from the start vertex to all other vertices are known, REA finds the k-th shortest path in $O(|E| + k\,|V| \log(|E| \,/\, |V|))$ [42].

Next, we briefly explain REA. In the first step of the algorithm, a counter k is set to 1 and the shortest paths from v^{start} to all other vertices $v \in V$ are computed. For any vertex $v \in V$, $\pi^k(v)$ denotes the k-th shortest path from the start vertex v^{start} to v. In the second step, k is incremented by 1 and $\pi^k(v^{\text{end}})$ is found by recursively calling a procedure called next path algorithm (NPA) which is given below. REA calls NPA with the parameters v^{end} and k. The second step of REA is repeated by setting $k = k + 1$ until $\pi^k(v^{\text{end}})$ does not exist or a particular stopping condition is fulfilled.

NPA takes a vertex $v \in V$ and an integer k as input parameters. In this procedure, the set of vertices $u \in V$ for which an edge from u to v exists, is defined as $\Gamma^{-1}(v)$, i.e., $\Gamma^{-1}(v) = \{u \in V : (u, v) \in E\}$. The " $\cdot$ " operator denotes the concatenation of a path $\pi^k(u)$ with an edge $(u, v) \in E$, e.g., $\pi^1(u) \cdot (u, v^{\text{end}})$ denotes the path formed by the shortest path from v^{start} to u and the edge $(u, v^{\text{end}}) \in E$. Every vertex $v \in V$ is reached from a vertex $u \in \Gamma^{-1}(v)$. A k-th shortest path to vertex v is chosen from a candidate set of paths $C(v)$ which is

initialized and updated as in the next path algorithm given below.

NEXT PATH ALGORITHM [42]
Input: $v \in V$, k.
Output: $\pi^k(v)$ or a message: $\pi^k(v)$ does not exist.

1: **if** k=2 **then**
2: Initialize a set of candidates $C(v) = \{\pi^1(u) \cdot (u, v) : u \in \Gamma^{-1}(v), \pi^1(v) \neq \pi^1(u) \cdot (u, v)\}$.
3: **if** $v = v^{\text{start}}$ **then**
4: Go to 14.
5: **end if**
6: **end if**
7: Let $u \in \Gamma^{-1}(v)$, k' be the vertex and index such that $\pi^{k-1}(v) = \pi^{k'}(u) \cdot v$.
8: **if** $\pi^{k'+1}(u)$ has not already been computed **then**
9: Call NEXTPATH ALGORITHM with input parameters $u \in V$, $k' + 1$.
10: **if** $\pi^{k'+1}(u)$ exists **then**
11: Insert $\pi^{k'+1}(u) \cdot v$ in $C(v)$.
12: **end if**
13: **end if**
14: **if** $C(v) \neq \emptyset$ **then**
15: Select and delete the shortest path in $C(v)$, output it as $\pi^k(v)$.
16: **else**
17: $\pi^k(v)$ does not exist, output message: $\pi^k(v)$ does not exist.
18: **end if**

The edge costs we use in REA are the modified costs updated in iteration l^{max} of LDD. In our numerical experiments, we observed that using these modified costs lowers the termination value of the counter k.

For the ease of notation, we partition a path $x \in \{0, 1\}^{|E^c|}$ in G^c as $x = (x_1, 1, x_2)$ where x_1, x_2 denote subpaths in trellis 1 and trellis 2 respectively. Note that the flow value on the edge $(v^{\text{end},1}, v^{\text{start},2})$ is 1. For a k-th shortest path x_1^k in trellis 1, there is a subpath x_2 in trellis 2 such that the set of constraints (7.5) are satisfied. This subpath can be derived from the values in x_1^k. The same also holds for a k-th shortest path x_2^k in trellis 2. Let $(x_1^k, 1, x_2)$ denote the agreeable path in G^c derived from x_1^k and $(\Lambda(x_1^k) + \Lambda(x_2))$ be the associated cost. The agreeable path derived from x_2^k and the associated cost are defined analogously.

There are exponentially many paths in trellis 1 and 2. In the worst case, finding the agreeable path in G^c which corresponds to the ML codeword can be intractable. The k-th shortest path algorithm (kSPA) proposed in this chapter, terminates with the k-th shortest path in trellis 1 and 2 if the condition in Step 4 is satisfied or the termination value k is larger than the maximum

number of iterations allowed k^{max}. In each iteration, upper bounds for z are calculated. If kSPA terminates in iteration $k < k^{max}$ then the ML codeword is found. Otherwise x^{best}, which is checked or updated in iterations, is output as an ML codeword estimate.

---

k-th shortest path algorithm (kSPA)
Input: Trellis 1 and 2, k^{max}, UB, x^{best} found in LDD.
Output: Best feasible solution x^{best}.

1: Set $k = 1$, UB and x^{best} to the values output by LDD.
2: **while** $k < k^{max}$ **do**
3: Compute x_1^k, x_2^k by REA [42].
4: **if** $\Lambda(x_1^k) + \Lambda(x_2^k) >$ UB **then**
5: terminate, x^{best} is the ML codeword.
6: **end if**
7: Compute $(\Lambda(x_1^k) + \Lambda(x_2))$, $(\Lambda(x_1) + \Lambda(x_2^k))$.
8: **if** UB $< \min\{(\Lambda(x_1^k) + \Lambda(x_2)), (\Lambda(x_1) + \Lambda(x_2^k))\}$ **then**
9: Set UB=$\min\{(\Lambda(x_1^k) + \Lambda(x_2)), (\Lambda(x_1) + \Lambda(x_2^k))\}$.
10: Set x^{best} using the agreeable path which yields the new UB.
11: **end if**
12: $k = k + 1$.
13: **end while**

Theorem 7.7. *If $k^{max} = \infty$, then kSPA outputs the ML codeword in termination iteration k.*

Proof. For each path x in the concatenated graph G^c, there is a subpath x_1 in trellis 1 and a subpath x_2 in trellis 2 such that $x = (x_1, 1, x_2)$. We denote by $x^{ML} = (x_1^{ML}, 1, x_2^{ML})$ the path which corresponds to the ML codeword. Let x_1^{ML} be the k_1th shortest path in trellis 1. Analogously, let x_2^{ML} be the k_2th shortest path in trellis 2. The termination iteration k is the iteration for which the condition $\Lambda(x_1^k) + \Lambda(x_2^k) >$ UB is satisfied.

Suppose $\min\{k_1, k_2\} > k$. Let $k' \leq k$ be the iteration such that UB = $\min\{(\Lambda(x_1^{k'}) + \Lambda(x_2)), (\Lambda(x_1) + \Lambda(x_2^{k'}))\}$. kSPA terminates in iteration k. It follows that $\Lambda(x_1^k) + \Lambda(x_2^k) \geq$ UB and

$$\Lambda(x_1^{k_1}) + \Lambda(x_2^{k_2}) \geq \Lambda(x_1^k) + \Lambda(x_2^k) \geq \text{UB}. \tag{7.12}$$

This is however a contradiction to the assumption that x^{ML} is formed by the k_1th shortest path in trellis 1 and k_2th shortest path in trellis 2. Consequently, there should exist at least one pair k_1, k_2 such that $\min\{k_1, k_2\} \leq k$. Let $k^* = \min\{k_1, k_2\} \leq k$. In step 8 of kSPA an upper bound from k^*-th shortest path in trellis 1 and trellis 2 is computed. Thus the ML codeword is computed. □

Best Agreeable Path decoding (BAPD)
Input: Trellis 1 and 2, $\lambda \in \mathbb{R}^n$.
Output: Best feasible solution x^{best}.

1: Call LDD.
2: **if** LDD terminates at l^{max} **then**
3: Call kSPA.
4: **end if**

BAPD which is a combination of LDD and kSPA can be summarized as follows.

Theorem 7.8. *BAPD outputs the ML codeword if $l < l^{max}$ or $k < k^{max}$.*

Proof. Follows from Proposition 7.5 and Theorem 7.7. □

7.5 Numerical results

We compare the error correcting performance of BAPD on (n, k) LTE turbo codes with short block length (cf. (132,40) LTE turbo code, (228,72) LTE turbo code). The decoding algorithms we use are: Log-MAP component decoding, LPD, and MLD. LPD curve is obtained by solving the LP relaxation of LTEIP by CPLEX LP solver [1] whereas MLD curve is obtained by solving LTEIP by CPLEX IP solver. The signal to noise ratio (SNR) is measured as E_b/N_0. The frame error rates (FERs) are calculated by counting 100 erroneous blocks. Randomly generated codewords are sent through the BIAWGNC.

For the (132,40) LTE turbo code, if the maximum number of iterations allowed in LDD is $l^{max} = 100$ and the maximum number of iterations allowed in kSPA is $k^{max} = 500$, BAPD performs better (in FER) than LPD by more than 1 dB. Compared to Log-MAP component decoding, the error correcting performance of BAPD is superior to the error correcting performance of Log-MAP component decoding by approximately 0.4 dB. Moreover, the average computation time (measured in CPU time) of BAPD is lower than LPD up to a factor 10. These observations are illustrated in Figures 7.5 and 7.6.

Without changing the l^{max} and k^{max} values used for the (132,40) LTE turbo code, BAPD is also applied to (228,72) LTE turbo code. In comparison to LPD, the error correcting performance of BAPD remains superior up to 0.4 dB. However, BAPD performs worse than Log-MAP component decoding by 0.2 dB for the (228,72) LTE turbo code. This is due to the reason that increasing the block length improves the error correcting performance of message passing algorithms but worsens the performance of optimization based algorithms since the search space gets larger. At the cost of increasing decoding times, the error correcting performance of BAPD can provably be improved by increasing

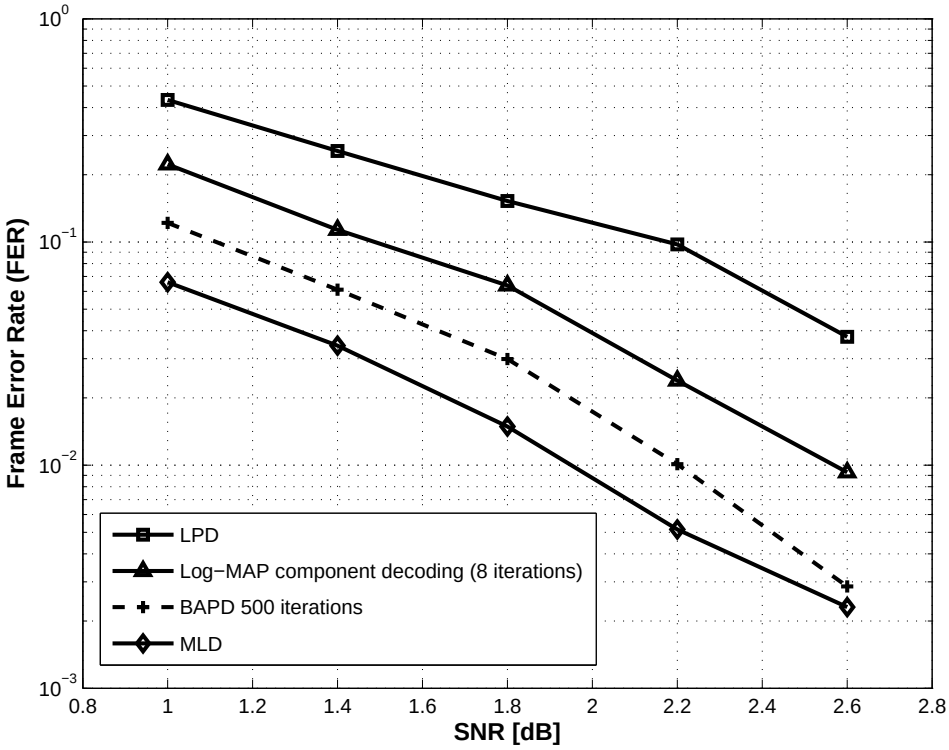


Figure 7.5: LPD, Log-MAP component decoding, MLD, BAPD for $(132, 40)$ LTE turbo code.

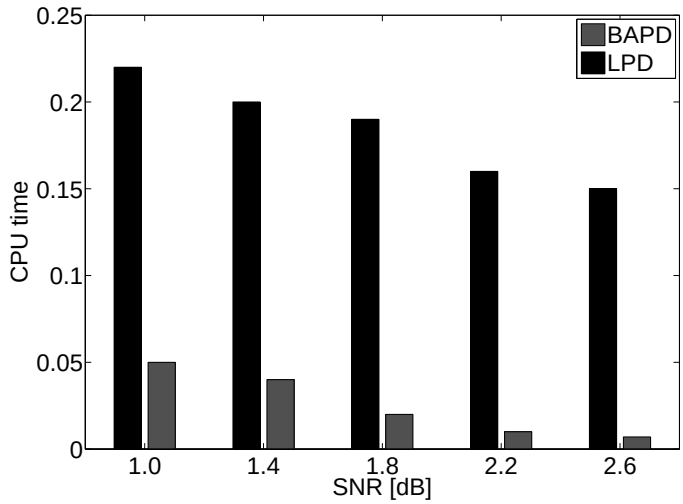


Figure 7.6: Decoding times of LPD and BAPD in CPU seconds.

k^{max}. To demonstrate this point we increased the k^{max} value and set it to 5000. The resulting curve is shown in Figure 7.7. It should be noted that at high SNR the error correcting performance of BAPD is superior to Log-MAP component decoding. The decoding times of LPD, BAPD $k^{max} = 500$, and BAPD $k^{max} = 5000$ are given in Figure 7.8. Note that even for $k^{max} = 5000$ the average decoding times of BAPD are less than the average decoding times obtained from LPD. Especially for high SNR values, BAPD with $k^{max} = 5000$ is faster than LPD by a factor of 10. To conclude, it may be a good strategy to increase k^{max} at high SNR.

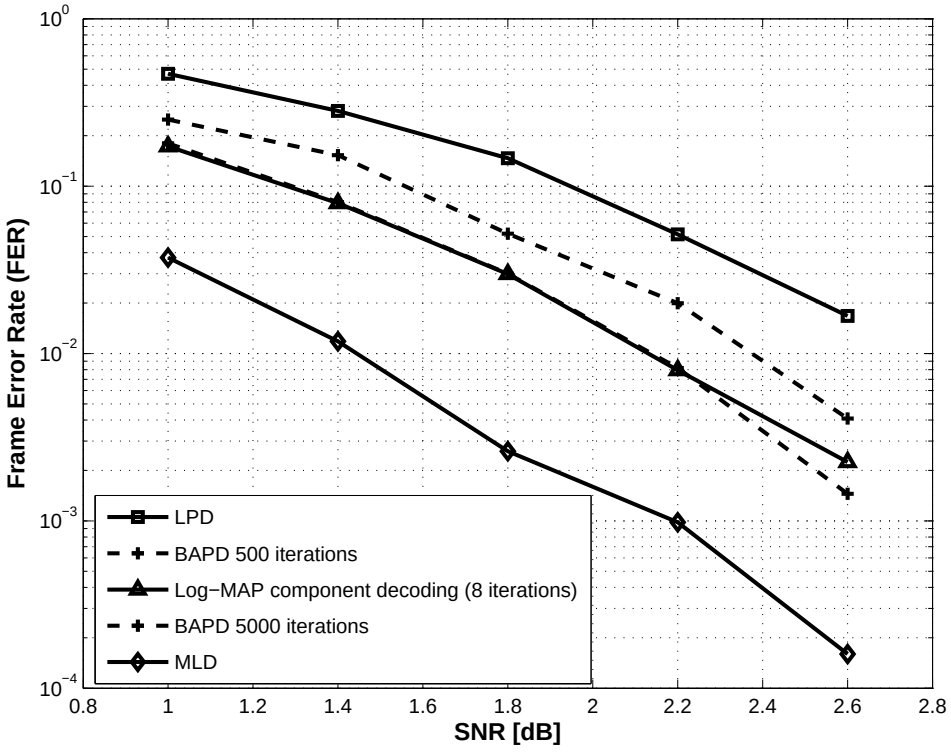


Figure 7.7: LPD, Log-MAP component decoding, MLD, BAPD for $(228, 72)$ LTE turbo code.

BAPD outputs the best agreeable path found during the iterations limited by l^{max} and k^{max}. If Theorem 7.8 is satisfied for some best agreeable path then it is also the ML codeword. Figure 7.9 demonstrates the ratio of ML codewords in all BAPD outputs. $(132, 40)$ LTE turbo code and $(228, 72)$ LTE turbo code are used again where k^{max} is set to 500 and 5000, respectively. At high SNR, it is observed that more than 98% of BAPD outputs are ML codewords.

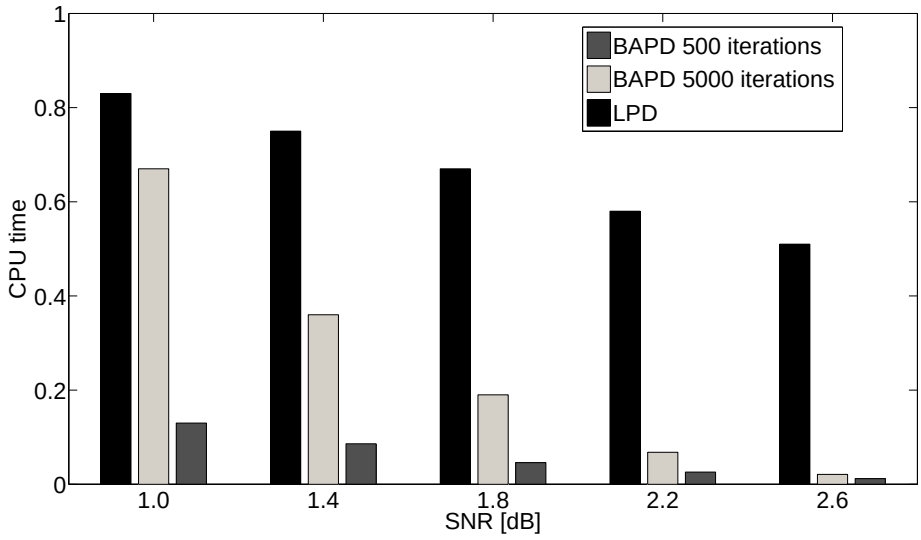


Figure 7.8: Decoding times of LPD and BAPD with $k^{max} = 500$, and BAPD with $k^{max} = 5000$ in CPU seconds.

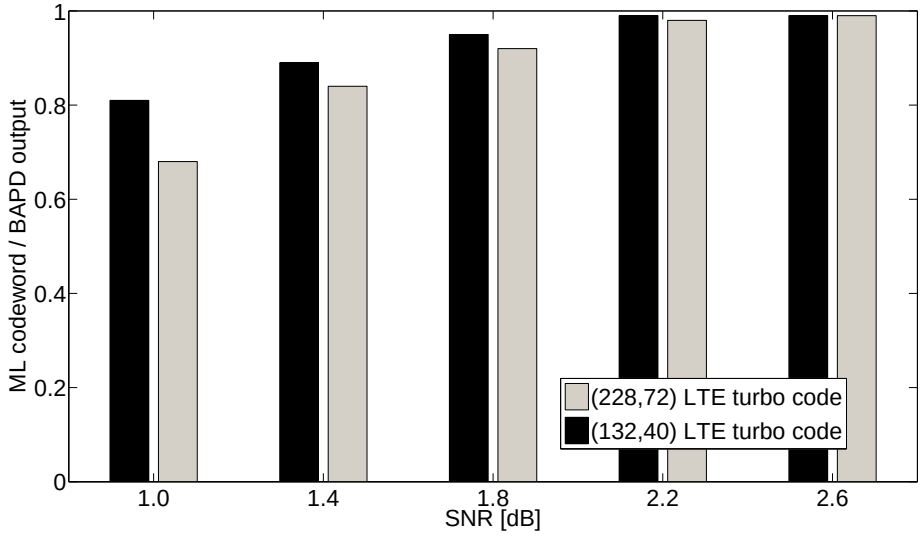


Figure 7.9: The ratio of ML codewords in all BAPD outputs.

7.6 Conclusion and further research

In this chapter we applied a two-step algorithm, BAPD, to solve the SCSPP defined on the concatenated graph G^c. In the first step, the Lagrangian dual of LTEIP is solved. To close the duality gap in the second step we apply a k-th shortest path algorithm. Under some assumptions it can be guaranteed that BAPD outputs the ML codeword. Promising numerical results are presented in Section 7.5.

The subgradient algorithm used in LDD will be studied further. Better step length parameters may be calculated using the underlying graph structure. Other efficient approaches to solve the LP relaxation of LTEIP, especially when the block length gets larger, will be investigated. Using appropriate data structures, efficiency of kSPA implementation can be improved. Combining standard turbo decoding with the approaches proposed in this chapter is another further research direction.

Chapter 8

Equality constraint shortest path problems

8.1 Introduction

In Chapter 7, we used a subgradient algorithm to solve the linear programming (LP) relaxation of a side constrained shortest path problem (SCSPP) with k equality constraints for $k \in \mathbb{Z}^+$. The underlying graph, $G = (V, E)$ with the set of vertices V and the set of edges E, was assumed to be an acyclic directed graph with positive or negative edge costs. Moreover, the equality constraints had a special structure modeling the interleaver consistency. A major drawback of the subgradient algorithm is its slow convergence in practice.

In this chapter, we study several special classes of side constrained shortest path problems (SCSPP) whose LP relaxation can be solved exactly by the combinatorial algorithms we propose. It is further assumed that the underlying graph is directed, has no cycles, and may contain edges with negative costs. The side constraints are assumed to be equality constraints with integral coefficients and zero right hand side. We refer to such problems as equality constraint shortest path problems (ECSPP(k)) where $k \in \mathbb{Z}^+$ specifies the number of side constraints.

Our study on ECSPP(k) may have implications for some other interesting problems, e.g., equality constraint knapsack problems can be transformed to equality constraint shortest path problems [52]. Although the algorithms we propose have exponential worst case complexity, in our numerical experiments we demonstrate that they are faster than the CPLEX LP solver up to a factor of more than 100.

In the formulation below, the path polytope (see Section 8.2) is denoted by X^{Path} where $c \in \mathbb{R}^{|E|}$, $x \in \{0,1\}^{|E|}$, and $g_t \in \mathbb{Z}^{|E|}$, $t = 1, \ldots, k$.

$$
\begin{array}{lll}
\text{minimize} & c^T x & \text{ECSPP}(k) \\
\text{subject to} & x \in X^{Path} & \\
& g_t^T x = 0 & \text{for all } t \in \{1, \ldots, k\} \\
& x \in \{0,1\}^{|E|}. &
\end{array}
$$

This chapter is organized as follows. The ECSPP problem with one side constraint is described in Section 8.2. In Section 8.3 we review some well-known and relevant results for multiple objective optimization (MOP). Then we propose a multiple objective linear optimization (MOLP) based solution algorithm to find an optimal solution of the LP relaxation of ECSPP(1) in Section 8.4. A special case of ECSPP(1) and the solution approaches to find an optimal solution of the LP relaxation of ECSPP(1) are also discussed in this section. Another MOLP based solution approach is used to solve the LP relaxation of ECSPP(2). The details of this approach are given in Section 8.5. Finally, the chapter is concluded with some further research ideas given in Section 8.6.

8.2 Solving the LP relaxation of ECSPP(1)

ECSPP(1) is the problem of finding a shortest path on a directed acyclic graph satisfying an equality constraint with right hand side equal to 0. Given a directed, acyclic graph $G = (V, E)$, a linear programming formulation of the shortest path problem on G can be written as [2] :

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{e \in E} c_e x_e & (8.1) \\
\text{subject to} & \displaystyle\sum_{e \in \text{ out}(v^{\text{start}})} x_e = 1 & (8.2) \\
& \displaystyle\sum_{e \in \text{ out}(v)} x_e = \sum_{e \in \text{in}(v)} x_e \qquad \text{for all } v \in V \setminus \{v^{\text{start}}, v^{\text{end}}\} & (8.3) \\
& \displaystyle\sum_{e \in \text{ in}(v^{\text{end}})} x_e = 1 & (8.4) \\
& x_e \geq 0 \qquad \text{for all } e \in E. & (8.5)
\end{array}
$$

This formulation models the situation that one unit flow is sent from a source vertex v^{start} to a target vertex v^{end}. The sets of incoming edges and outgoing edges of vertex $v \in V$ are denoted by in(v) and out(v), respectively. The variable x_e models the amount of flow on $e \in E$ whereas c_e is the cost of using this edge. Constraints (8.2)-(8.5) describe an integral polytope which we refer to as the path polytope and denote by X^{path}.

Having introduced the path polytope, the integer programming formulation of ECSPP(1) is given as:

$$\begin{aligned} &\text{minimize } && c^T x && \text{ECSPP(1)} \\ &\text{subject to } && x \in X^{Path} \\ & && g^T x = 0 \\ & && x \in \{0,1\}^{|E|}. \end{aligned}$$

where $c \in \mathbb{R}^{|E|}$, $g \in \mathbb{Z}^{|E|}$. Typically, the components g_e, $e \in E$ of a vector g reflect the delay or resource consumption on e.

Theorem 8.1. *ECSPP(1) is NP hard.*

Sketch of proof. The decision problem of ECSPP(1) is: given $k \in \mathbb{R}$, does there exist a v^{start}-v^{end} path modeled by $x \in \{0,1\}^{|E|}$ such that $c^T x \leq k$ and $g^T x = 0$? This problem is clearly in NP. To show that ECSPP(1) is NP hard, the decision problem of the subset sum problem [31] is reduced to the decision problem of ECSPP(1) analogous to the reduction of the decision problem of 0-1 knapsack problem to bicriterion shortest path problem in acyclic directed graphs (See [22, Theorem 9.2]). □

The LP relaxation of ECSPP(1) which is denoted by R-ECSPP(1) is:

$$\begin{aligned} &\text{minimize } && c^T x && \text{R-ECSPP(1)} && (8.6) \\ &\text{subject to } && x \in X^{Path} && && (8.7) \\ & && g^T x = 0. && && (8.8) \end{aligned}$$

In the literature, there exists several approaches, e.g., [59], [75], and [80] to solve the LP relaxation of a shortest path problem with one side constraint. This side constraint is of type $g^T x \leq \mathcal{G}$ where g_e, $e \in E$ and $\mathcal{G}$ are non-negative integers. The solution algorithm introduced in [59] is a multiple objective optimization based, efficient solution approach. We extend this approach so that it can also be used for R-ECSPP(1).

An alternative formulation for ECSPP(1), its LP relaxation, and the associated dual problem help for a better understanding of the proofs given in this chapter. Let P denote the set of all v^{start}-v^{end} paths. The cost of a path $p \in P$ is $c(p) = \sum_{e \in p} c_e$. The function $g(p)$ is defined as $g(p) = \sum_{e \in p} g_e$. An equality

constraint shortest path problem can also be formulated as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{p \in P} c(p)p && \text{(ECSPP(1))} \\
\text{subject to} \quad & \sum_{p \in P} p = 1 && \\
& \sum_{p \in P} g(p)p = 0 && \\
& p \in \{0, 1\} && \text{for all } p \in P.
\end{aligned}$$

The IP formulation above may have exponentially many variables since there is one variable for each v^{start}-v^{end} path in the graph. The LP relaxation is obtained by dropping the integrality constraints on the p variables.

$$\begin{aligned}
\text{minimize} \quad & \sum_{p \in P} c(p)p && \text{(R-ECSPP(1))} && (8.9) \\
\text{subject to} \quad & \sum_{p \in P} p = 1 && && (8.10) \\
& \sum_{p \in P} g(p)p = 0 && && (8.11) \\
& p \geq 0 && \text{for all } p \in P. && (8.12)
\end{aligned}$$

Our focus will be on the dual of the LP relaxation.

$$\begin{aligned}
\text{maximize} \quad & u && \text{(D-ECSPP(1))} \\
\text{subject to} \quad & u + vg(p) \leq c(p) && \text{for all } p \in P.
\end{aligned}$$

In the dual formulation, there are only two variables. On the other hand, the number of constraints may be exponential. Nevertheless, for a fixed $v \in \mathbb{R}$, the value of u can be found by computing a shortest path problem on $G = (V, E)$ where edge costs are modified.

$$u(v) = \min_{p \in P} \{c(p) - vg(p)\}. \tag{8.13}$$

The dual problem can also be stated as

$$\max_{v \in \mathbb{R}} u(v). \tag{8.14}$$

Definition 8.2. *A solution of Problem 8.14 is called an optimal multiplier* $v^* \in \operatorname{argmax}_{v \in \mathbb{R}} u(v)$.

The exact solution algorithm we propose for R-ECSPP(1) utilizes concepts from multiple objective optimization. In the next section, we recall some well-known, relevant results from this field of mathematical programming.

8.3 Multiple objective linear programming

An optimization problem with multiple objectives is called multiple objective optimization (MOP) problem. Usually the objective functions are conflicting and a single solution optimizing all objective functions simultaneously does not exist. Thus, the concept of Pareto optimality is used in MOP. A solution of a MOP problem is Pareto optimal if one objective function can not be improved without worsening another objective function. For a comprehensive review on MOP, [22] is referred. We concentrate on multiple objective linear programming (MOLP) problems.

Definition 8.3. *If the objective functions of a MOP are linear and the feasible set is a polyhedron, X, then the MOP problem is called a multiple objective linear programming (MOLP) problem.*

$$\begin{aligned} &\textit{minimize} \quad f(x) = (f_1(x), f_2(x), \ldots, f_Q(x)) && \textit{(MOLP)} \\ &\textit{subject to} \quad x \in X. \end{aligned}$$

In the context of MOP, $X \subseteq \mathbb{R}^{|E|}$ is referred to as the decision space, whereas $Y = \{y \in \mathbb{R}^Q : y = (f_1(x), f_2(x), \ldots, f_Q(x)), x \in X\}$ is called the objective space. Since no canonical ordering is defined in $\mathbb{R}^Q$, $Q \geq 2$, a componentwise ordering in the objective space is defined as follows.

Definition 8.4. *Let $y^1, y^2 \in \mathbb{R}^Q$. Then*

$$\begin{aligned} &y^1 \leqq y^2 :\Leftrightarrow y_i^1 \leqq y_i^2 \textit{ for all } i \in \{1, 2, \ldots, Q\} \\ &y^1 \leq y^2 :\Leftrightarrow y^1 \leqq y^2 \textit{ and } y^1 \neq y^2. \end{aligned}$$

Next, we give some definitions necessary for a better understanding of optimality in the presence of more than one objective function.

Definition 8.5. *The Pareto cone $\mathbb{R}^Q_{\leqq}$ is defined as $\mathbb{R}^Q_{\leqq} = \{y \in \mathbb{R}^Q : y \geqq 0\}$.*

Definition 8.6. *A decision vector $x \in X$ is called efficient (Pareto optimal) if there does not exist another decision vector $\bar{x}$ such that $f(\bar{x}) \leq f(x)$. The set of all efficient vectors is called the efficient set.*

$$X_E = \{x \in X : \nexists\, \bar{x} \in X : f(\bar{x}) \leq f(x)\}.$$

The image set of X_E is called the non-dominated set $Y_N = f(X_E)$.

Equivalently, the non-dominated set Y_N is the set of all non-dominated points.

Definition 8.7. *A vector $y \in \mathbb{R}^Q$ in the objective space is non-dominated if $Y \cap (y - \mathbb{R}^Q_{\leqq}) = \{y\}$.*

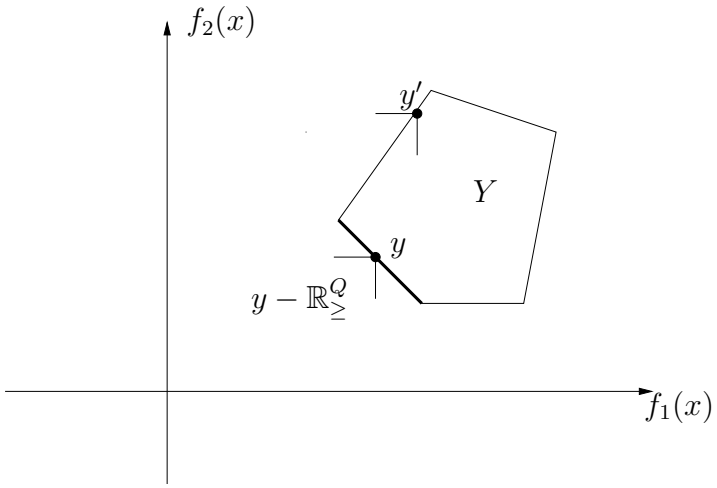


Figure 8.1: Dominated and non-dominated points of the objective space $Y \subseteq \mathbb{R}^2$.

Example 8.8. *Definition 8.7 is illustrated in Figure 8.1. $Y \subseteq \mathbb{R}^2$ is an example objective space. According to Definition 8.7, y' is a dominated point whereas y is a non-dominated point. Using Definition 8.7, it can be verified that non-dominated points in Y are located on the bold line.*

It is well-known for MOLP problems that the set of non-dominated points is identical to the non-dominated frontier which is defined as $Y_{NF} = \{y \in \text{conv}(Y_N) : \text{conv}(Y_N) \cap (y - \mathbb{R}^Q_{\geqq}) = \{y\}\}$. The objective space $Y \subseteq \mathbb{R}^2$ is a convex set for MOLP problems and if $Q = 2$, the non-dominated frontier can be regarded as the graph of a piecewise linear and convex function.

Example 8.9. *A non-dominated frontier is illustrated in Figure 8.2. The squares represent the breakpoints. These points are non-dominated extreme points of Y which are the images of the efficient extreme points in the decision space.*

Definition 8.10. *Let $0 < w_i < 1$ for all $i \in \{1, \ldots, Q\}$ and $\sum_{i=1}^{Q} w_i = 1$. The single objective optimization problem*

$$\begin{aligned} &\textit{minimize} \quad w_1 f_1(x) + w_2 f_2(x) + \ldots + w_Q f_Q(x) \qquad \textit{(WSSP(w))} \\ &\textit{subject to} \quad x \in X. \end{aligned}$$

is called the weighted sum scalarization problem (WSSP(w)).

The following results on the weighted sum scalarization of MOLP problems were stated in [22].

Theorem 8.11. *Let x^{ws} be the optimal solution of a WSSP(w) with $w_i > 0$, $i \in \{1, \ldots, Q\}$. Then x^{ws} is efficient, and $f(x^{ws})$ is non-dominated.*

Proposition 8.12. *Each efficient solution $x \in X_E$ of a MOLP problem can be found by solving a WSSP(w) with $w_i > 0$, $i \in \{1, \ldots, Q\}$.*

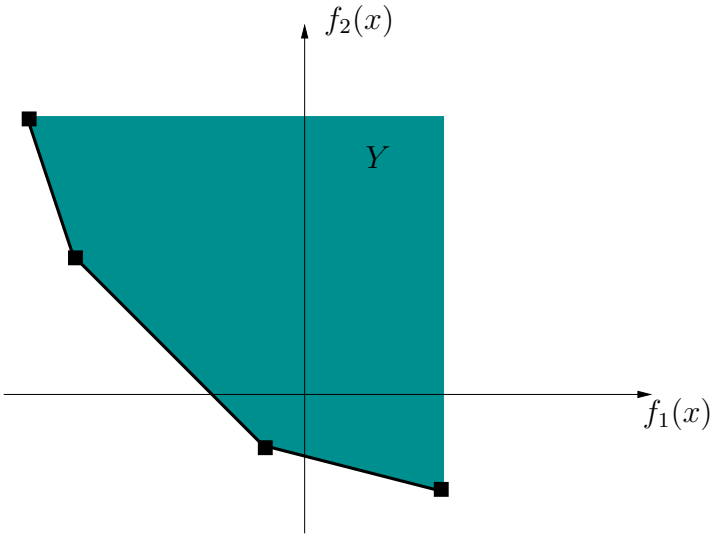


Figure 8.2: Non-dominated frontier of a MOLP problem where $Y \subseteq \mathbb{R}^2$.

8.4 Multiple objective linear programming approach to R-ECSPP(1)

In the approach introduced in this section we derive a multiple objective linear optimization problem which has an efficient solution being optimal for R-ECSPP(1). Before explaining the MOLP approach to R-ECSPP(1), we make some assumptions to exclude some trivial special cases.

Assumption 1: R-ECSPP(1) is feasible,

Assumption 2: Let $x^c = \text{argmin}\{c^T x : x \in X^{Path}\}$. Then $g^T x^c \neq 0$.

In the MOLP based approach, we consider X^{Path} as the decision space. The image of X^{path} in the objective space is denoted by Y^{Path}. We derive the first objective function and the horizontal axis from $g^T x$ according to the value of $g^T x_c^0$, $x_c^0 = \text{argmin}\{c^T x : x \in X^{Path}\}$ (see Proposition 8.20). The horizontal axis is denoted by $f_1(x)$. The vertical axis in the objective space is $f_2(x) = c^T x$.

Observation 8.13. *Let x^* denote the optimal solution of R-ECSPP(1). From a geometrical point of view, $f(x^*)$ must be on the $f_2(x)$-axis such that $c^T x^* = \min\{c^T x : x \in X^{Path}, f_1(x) = 0\}$. It also holds that $f(x^*)$ is located on some face of Y^{Path}.*

It can also be concluded that there are two paths x^1 and x^2 such that $f_1(x^1) < 0$, $f_1(x^2) > 0$, and $f(x^*) \in \text{conv}(\{f(x^1), f(x^2)\})$. The convex hull of $f(x^1)$ and $f(x^2)$ has either positive or negative slope.

Example 8.14. *In Figure 8.3 two objective spaces, Y^1 and Y^2 are illustrated. In both objective spaces the endpoints of the facet conv$(\{f(x^1), f(x^2)\})$ are labeled by y^1 and y^2. By $y^* = f(x^*)$, we denote the images of optimal solutions in the objective spaces.*

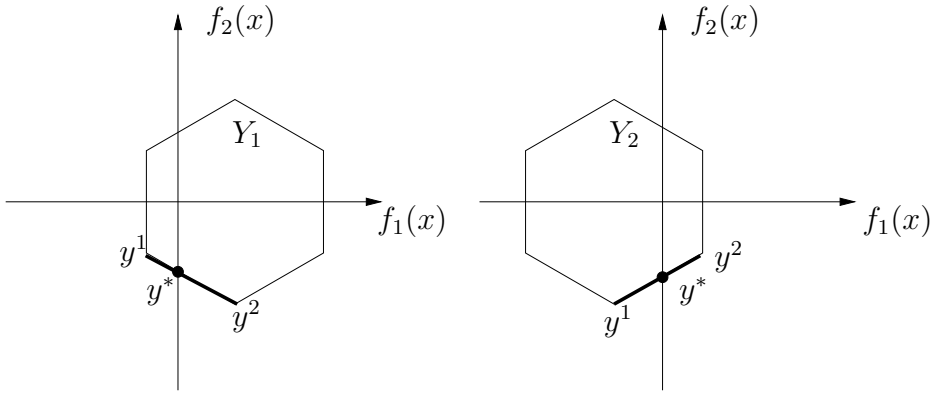


Figure 8.3: The non-dominated facet containing the image of the optimal solution of R-ECSPP(1).

The position of $f(x^*)$ can be specified using the notion of optimality from MOP. We first describe how to derive a second objective function from the constraint $g^T x = 0$. Problem 8.15, which can be solved in $O(|V| + |E|)$ [16] (since G is directed, acyclic) with a shortest path algorithm is considered first.

$$\min\{c^T x : x \in X^{Path}\}. \tag{8.15}$$

We denote the optimal solution of Problem 8.15 by x_c^0. Then the value $g^T x_c^0$ is computed.

Case 1:
If $g^T x_c^0 > 0$, then the points of intersection of Y^{Path} and $f_2(x)$-axis are on the left of $f(x_c^0)$. To find an efficient solution, x_g^0 such that $f_1(x_g^0) < 0$, Problem 8.16 is solved. A small constant $\epsilon > 0$ is chosen such that x_g^0 approximates the lexicographic optimum (see [22]) of $\min\{f(x) = (g^T x, c^T x) : x \in X^{Path}\}$.

$$\min\{g^T x + \epsilon c^T x : x \in X^{Path}\}. \tag{8.16}$$

The objective value $g^T x_g^0$ has to be non-positive according to Assumption 1. Due to convexity of Y^{Path}, the image of the optimal solution of R-ECSPP(1) is located in the triangle defined by the points $f(x_g^0)$, $f(x_c^0)$, and $(g^T x_g^0, c^T x_c^0)$.

Example 8.15. *An illustration of Case 1 is given in Figure 8.4.*

Proposition 8.16, relates Problem 8.17 to R-ECSPP(1).

Proposition 8.16. *Given the MOLP problem*

$$\min f(x) = (g^T x, c^T x) \text{ s.t. } x \in X^{Path}. \tag{8.17}$$

All non-dominated points in Y^{Path} for Problem 8.17 are located in the triangle defined by the points $f(x_g^0)$, $f(x_c^0)$, $(g^T x_g^0, c^T x_c^0)$.

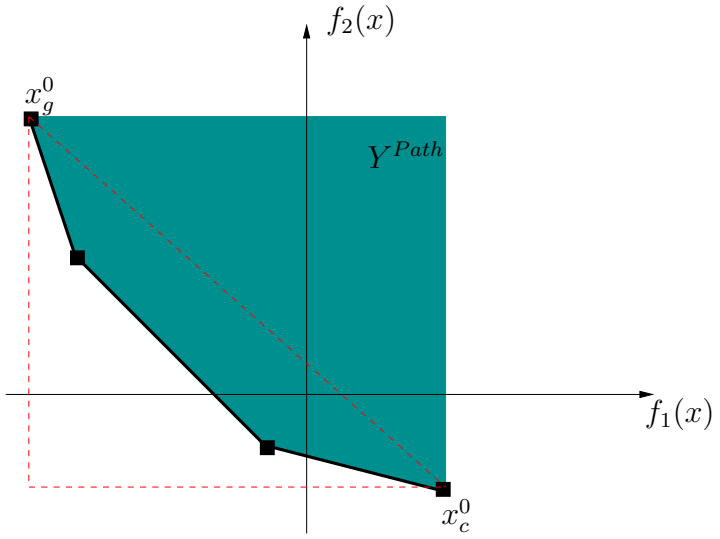


Figure 8.4: An illustration of the non-dominated frontier of Y^{Path} (bold line) for Case 1.

Proof. The proof follows from convexity of Y^{Path} and Definition 8.7. □

We argue that x^* can be found by investigating the non-dominated points of Y^{Path} for Problem 8.17 since $f(x^*)$ is a non-dominated point.

Proposition 8.17. *Let x^* be an optimal solution of R-ECSPP(1). The point $f(x^*) \in Y^{Path}$ is a non-dominated point for Problem 8.17.*

Proof. The proof follows from the convexity of Y^{Path} and the notion of dominance. □

Case 2:
If $g^T x_c^0 < 0$, then the points of intersection of Y^{Path} and $f_2(x)$-axis are on the right of $f(x_c^0)$. Consequently, we solve Problem 8.18.

$$\max\{g^T x - \epsilon c^T x : x \in X^{Path}\}. \tag{8.18}$$

Let x_g^0 denote the optimal solution of Problem 8.18. Again due to the convexity of Y^{Path}, the image of the optimal solution $f(x^*)$ is located in the triangle defined by the points $f(x_g^0)$, $f(x_c^0)$, $(g^T x_g^0, c^T x_c^0)$. However in this case, the objective value $g^T x_g^0$ is non-negative according to Assumption 1 and $f(x^*)$ is on a facet which has positive slope.

Example 8.18. *An illustration of Case 2 is given in Figure 8.5.*

There is a relation between $f(x^*)$ and the non-dominated points of the biobjective problem where the objective function $c^T x$ is minimized and the objective function $g^T x$ is maximized. Hence, Proposition 8.16 has to be slightly modified. Proposition 8.19 holds for Problem 8.19.

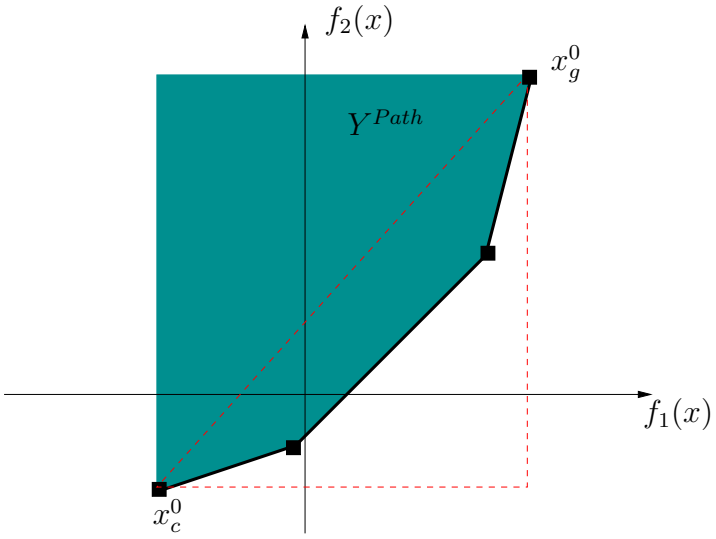


Figure 8.5: An illustration of the non-dominated frontier of Y^{Path} (bold line) for Case 2.

Proposition 8.19. *Given the MOLP problem,*

$$\min f(x) = (-g^T x, c^T x) \text{ s.t. } x \in X^{Path}. \tag{8.19}$$

All non-dominated points in Y^{Path} for Problem 8.19 are located in the triangle defined by the points $f(x_c^0)$, $f(x_g^0)$, $(g^T x_g^0, c^T x_c^0)$.

From the case distinction, it follows that Case 2 can be reduced to Case 1.

Proposition 8.20. *Case 2 can be reduced to Case 1 if both sides of the side constraint $g^T x = 0$ in R-ECSPP(1) are multiplied by -1. It is sufficient to investigate the non-dominated points of Problem 8.20*

$$\min f(x) = (f_1(x), f_2(x)) \text{ s.t. } x \in X^{Path} \tag{8.20}$$

where $f_1(x) = g^T x$ if $g^T x_c^0 > 0$ and $f_1(x) = -g^T x$ if $g^T x_c^0 < 0$, and $f_2(x) = c^T x$.

The main theorem of this section states that given R-ECSPP(1), an optimal solution x^* of R-ECSPP(1) is an efficient solution of Problem 8.20 where $f(x^*)$ is on the non-dominated facet $\text{conv}(\{f(x_g^k), f(x_c^k)\}) \in Y_{NF}^{Path}$ which intersects the $f_2(x)$-axis. The extreme points defining this facet are denoted by $f(x_g^k)$ and $f(x_c^k)$.

Theorem 8.21. *Let x^* be an optimal solution of R-ECSPP(1), it holds that $f(x^*) \in$ conv$(\{f(x_g^k), f(x_c^k)\})$.*

Proof. Let $\text{conv}(\{f(x_g^k), f(x_c^k)\})$ denote a non-dominated facet of Problem 8.20 which intersects the $f_2(x)$-axis. The extreme points of this facet, $f(x_g^k)$ and

$f(x_c^k)$ are breakpoints of the non-dominated frontier. Hence, they are the images of two extreme points x_g^k and x_c^k of X^{Path}. Let $(0, c^T x^*)$ denote the intersection point of conv$(\{f(x_g^k), f(x_c^k)\})$ and $f_2(x)$-axis. Due to the definition of convex hull, there exists k_1, k_2 such that

$$k_1, k_2 \geq 0, \tag{8.21}$$
$$k_1 + k_2 = 1, \tag{8.22}$$
$$k_1 f_1(x_g^k) + k_2 f_1(x_c^k) = 0, \tag{8.23}$$
$$k_1 c^T x_g^k + k_2 c^T x_c^k = c^T x^*. \tag{8.24}$$

The values of path variables in the Formulation (8.9)-(8.12) of R-ECSPP(1) are set so that the value of path p_1 modeled by x_g^k, is set to k_1 and the value of path p_2 modeled by x_c^k, is set to k_2. The values of all paths $p \in P \setminus \{p_1, p_2\}$ are set to 0. Obviously $(k_1, k_2, 0, \ldots, 0)^T \in [0,1]^{|P|}$ is a feasible solution of Problem (8.9)-(8.12).

Next, we consider the weighted sum scalarization of Problem 8.20.

$$\min\{w_1 f_1(x) + w_2 f_2(x) \text{ s.t. } x \in X^{Path}\}. \tag{8.25}$$

This problem can equivalently be written as

$$\min\{\frac{w_1}{w_2} f_1(x) + f_2(x) \text{ s.t. } x \in X^{Path}\}. \tag{8.26}$$

Suppose that the weight vector $w^* = (w_1^*, w_2^*)$ where $0 < w_1^* < 1$, $0 < w_2^* < 1$, $w_1^* + w_2^* = 1$ is chosen such that x_g^k (p_1) and x_c^k (p_2) are two optimal solutions of Problem 8.26.

Now we consider the dual of Formulation (8.9)-(8.12), D-ECSPP(1). If we set $v^* = -\frac{w_1^*}{w_2^*}$, then the paths p_1 and p_2 are the minimizers of Problem 8.27

$$u(v^*) = \min_{p \in P}\{c(p) - v * g(p)\}$$

and $(u(-\frac{w_1^*}{w_2^*}), -\frac{w_1^*}{w_2^*}) \in \mathbb{R}^2$ is dual feasible.

If we can show that $(k_1, k_2, 0, \ldots, 0)^T$ and $(u(-\frac{w_1^*}{w_2^*}), -\frac{w_1^*}{w_2^*})^T \in \mathbb{R}^2$ satisfy the complementary slackness conditions, then $(k_1, k_2, 0, \ldots, 0)^T$ is an optimal solution of R-ECSPP(1). The variable v is set to $-\frac{w_1^*}{w_2^*}$. Since p_1, p_2 are two shortest paths of Problem 8.27 with this particular value of v, for all $p \in P \setminus \{p_1, p_2\}$ it holds that

$$u < c(p) - vg(p). \tag{8.27}$$

It follows that the complementary slackness conditions are satisfied for all $p \in P \setminus \{p_1, p_2\}$ since their values are set to zero in the primal solution. The values

of paths p_1 and p_2 are greater than or equal to zero in the primal solution and the dual constraint satisfies

$$u - c(p) + vg(p) = 0, \text{ if } p = p_1 \text{ or } p = p_2. \tag{8.28}$$

So the complementary slackness conditions are also fulfilled for p_1 and p_2. Thus the solution considered above is an optimal solution of R-ECSPP(1). □

In the following, we describe a method, which we call non-dominated facet algorithm (NFA), to find conv$(\{f(x_g^k), f(x_c^k)\})$ for R-ECSPP(1). NFA is a variant of Algorithm 6 proposed in [59]. At the beginning of the algorithm, two points $f(x_g^0)$ and $f(x_c^0)$ are computed as described above in distinction of cases. If conv$(\{f(x_g^0), f(x_c^0)\})$ is a non-dominated facet intersecting the $f_2(x)$-axis, NFA stops. Otherwise there exists another non-dominated point located inside the triangle defined by the points $f(x_g^0)$, $f(x_c^0)$, $(f_1(x_g^0), c^T x_c^0)$ according to Proposition 8.16.

As stated in Section 8.3, non-dominated points in Y^{Path} can be found by solving the weighted sum scalarization of Problem 8.20. For any $0 < w_1 < 1$, $0 < w_2 < 1$, and $w_1 + w_2 = 1$ the optimal solution of Problem 8.25 is an efficient solution and its image is a non-dominated point. An optimal solution of the weighted sum problem is denoted by x_{ws}^0.

According to the value $f_1(x_{ws}^0)$, either $f(x_g^0)$ or $f(x_c^0)$ is replaced by $f(x_{ws}^0)$.

If $f_1(x_{ws}^0) = 0$ then x_{ws}^0 is the optimal solution of R-ECSPP(1), NFA terminates,

else if $f_1(x_{ws}^0) > 0$ then $f(x_c^0)$ is replaced by $f(x_{ws}^0)$.

else $f(x_g^0)$ is replaced by $f(x_{ws}^0)$.

The new pair of points is denoted by $f(x_g^1)$ and $f(x_c^1)$. In NFA, the above procedure is performed iteratively (l denotes the iteration counter) until the non-dominated facet intersecting $f_2(x)$-axis, conv$(\{f(x_g^k), f(x_c^k)\})$ is found. In each iteration, $f(x_g^l)$ is located on the left of $f_2(x)$-axis whereas $f(x_c^l)$ is located on the right. The weights for the weighted sum problem are chosen according to Proposition 8.22.

Proposition 8.22. *The minimization direction is orthogonal to conv$(\{f(x_g^l), f(x_c^l)\})$, $l = 0, 1, 2, \ldots$, and points in the interior of the triangle defined by the points $f(x_g^l)$, $f(x_c^l)$, and $(f_1(x_g^l), c^T x_c^l)$ if weights are set as*

$$w_1 = (c^T x_g^l - c^T x_c^l) \text{ and } w_2 = (f_1(x_c^l) - f_1(x_g^l)). \tag{8.29}$$

Proof. From elementary geometry, the equation $w_1 f_1(x) + w_2 f_2(x) = b$ defines a line orthogonal to vector (w_1, w_2). If $w_1 f_1(x) + w_2 f_2(x)$ corresponds to an objective function, then in a minimization problem the line $w_1 f_1(x) + w_2 f_2(x) = b$ is shifted in the opposite (w_1, w_2) direction until an optimal solution is found.

Let $w_1 = (c^T x_g^l - c^T x_c^l)$ and $w_2 = (f_1(x_c^l) - f_1(x_g^l))$. Then the line $w_1 f_1(x) + w_2 f_2(x) = b$ is parallel to conv$(\{f(x_g^l), f(x_c^l)\})$. Moreover, by shifting the line $w_1 f_1(x) + w_2 f_2(x) = b$ in the opposite (w_1, w_2) direction, an optimal solution of WSSP(w), which is in the triangle defined by the points $f(x_g^l)$, $f(x_c^l)$, and $(f_1(x_g^l), c^T x_c^l)$ is found. □

Theorem 8.23. *In iteration l, if $f(x_{ws}^l) \in$ conv$(f(x_g^l), f(x_c^l))$, then the convex hull of $\{f(x_g^l), f(x_c^l)\}$ is a non-dominated facet intersecting the $f_2(x)$-axis.*

Proof. By construction, there is an intersection point of conv$(\{f(x_g^l), f(x_c^l)\})$ and the $f_2(x)$-axis in each iteration $l = 0, 1, 2, \ldots$ For some l and a given weight vector w, if $f(x_{ws}^l) \in$ conv$(\{f(x_g^l), f(x_c^l)\})$ then all points in conv$(\{f(x_g^l), f(x_c^l)\})$ are the images of alternative optimal solutions of Problem 8.25. Thus, they are efficient solutions of Problem 8.20. It follows that conv$(\{f(x_g^l), f(x_c^l)\})$ is a non-dominated facet. □

Theorem 8.24. *NFA finds the non-dominated facet conv$(\{f(x_g^k), f(x_c^k)\})$ in finitely many steps.*

Proof. In Step 12 of NFA, a shortest path problem with modified edge costs is solved. Therefore x_{ws}^l is a path. If $f(x_{ws}^l) \in$ conv$(\{f(x_g^l), f(x_c^l)\})$ or $f_1(x_{ws}^l) = 0$ then NFA terminates. Otherwise $f(x_{ws}^l)$ is a non-dominated point such that $f_1(x_g^l) < f_1(x_{ws}^l) < f_1(x_c^l)$ and $c^T x_c^l < c^T x_{ws}^l < c^T x_g^l$. Since the values of x_g^l, x_{ws}^l, x_c^l, and g are integers, NFA terminates after a finite number of iterations. □

8.4.1 R-ECSPP(1) problems, a special case

In this section we concentrate on a special case of R-ECSPP(1) problems. We assume that the side constraint $g^T x = 0$ is in the form of an interleaver consistency constrains mentioned in Chapter 7. Although our study in this section has no immediate practical use in terms of decoding of binary linear codes due to the limitation of a single side constraint, a theoretical understanding may help in developing efficient solution approaches for the side constrained shortest path problem considered in Chapter 7.

Given a directed, acyclic graph $G = (V, E)$, let E_1, E_2 denote two edge sets.

Definition 8.25. *A path is called agreeable if it contains an edge from the sets E_1 and E_2 both or it contains no edge from the sets E_1 and E_2.*

In Problem (8.30) - (8.32), the minimum cost agreeable path is searched.

$$\text{minimize} \quad c^T x \tag{8.30}$$

$$\text{subject to} \quad \sum_{e \in E_1} x_e = \sum_{e \in E_2} x_e \tag{8.31}$$

$$x \in X^{Path} \cap \{0,1\}^{|E|} \tag{8.32}$$

Non-dominated facet algorithm (NFA)
Input: An equality constraint shortest path problem.
Output: Two paths x_g^k, x_c^k.

Set $l = 0$.
Compute $x_c^l = \text{argmin}\{c^T x : x \in X^{Path}\}$.
if $g^T x_c^l > 0$ **then**
  Set $f_1(x) = g^T x$.
else
  Set $f_1(x) = -g^T x$.
end if
Compute $x_g^l = \text{argmin}\{(1-\epsilon) f_1(x) + \epsilon f_2(x) : x \in X^{Path}\}$.
Set terminate = false.
repeat
  Set (w_1, w_2)=$((c^T x_g^l - c^T x_c^l), (f_1(x_c^l) - f_1(x_g^l)))$.
  Compute x_{ws}^l = $\text{argmin}\{w_1 f_1(x) + w_2 f_2(x) : x \in X^{Path}\}$.
  if $f_1(x_{ws}^l) = 0$ **then**
    Set terminate = true.
  else if $f_1(x_{ws}^l) > 0$ **then**
    if $f(x_{ws}^l) \in \text{conv}(\{f(x_g^l), f(x_c^l)\})$ **then**
      Set terminate =true.
    else
      Set $f(x_c^l)$=$f(x_{ws}^l)$.
    end if
  else
    if $f(x_{ws}^l) \in \text{conv}(\{f(x_g^l), f(x_c^l)\})$ **then**
      Set terminate = true.
    else
      Set $f(x_g^l)$=$f(x_{ws}^l)$.
    end if
  end if
  Set $l = l + 1$.
until terminate = true
Output x_g^l and x_c^l.

Note that in this problem, $g \in \{-1, 0, 1\}^{|E|}$. The set of all paths which contain an edge from E_1 is denoted by P_1 whereas the set of all paths which contain an edge from E_2 is denoted by P_2. The set of agreeable paths is $P^{AG} = (P_1 \cap P_2) \cup ((P \setminus P_1) \cap (P \setminus P_2))$. Furthermore, the set of paths where an edge in E_1 is followed but no edge from E_2 is used, is defined as $P' = P_1 \cap P \setminus P_2$. $P'' = P \setminus P_1 \cup P_2$ is defined analogously.

Lemma 8.26. *For all paths $p \in P'$ it holds that $g^T x^p = 1$ and for all paths $p \in P''$ it holds that $g^T x^p = -1$. Similarly, for all agreeable paths, $p \in P^{AG}$, $g^T x^p = 0$.*

As a result, in the objective space, non-dominated points corresponding to paths are located on lines $f_1(x) = -1$, $f_1(x) = 0$, or $f_1(x) = 1$.

Example 8.27. *Figure 8.6 illustrates an objective space where the image of the optimal solution of the shortest path problem (without constraint 8.31), $f(x_c^0)$, is on the line $f_1(x) = 1$. Two other possibilities are $f(x_c^0)$ is on $f_2(x)$-axis or $f(x_c^0)$ is on $f_1(x) = -1$.*

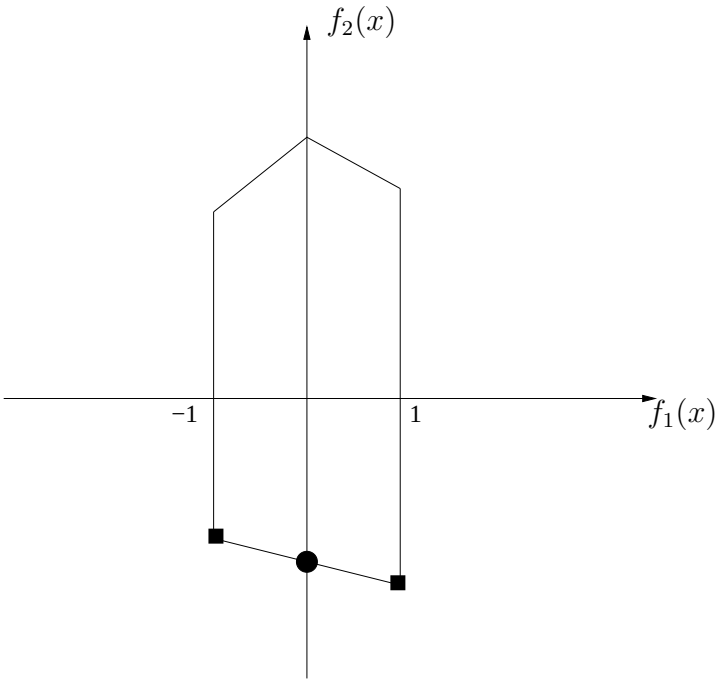


Figure 8.6: A special case of R-ECSPP(1) problems.

Given the special case of R-ECSPP(1). The non-dominated facet containing the image of an optimal solution of R-ECSPP(1) can be computed easily by NFA.

Theorem 8.28. *NFA finds the non-dominated facet for Problem (8.30) - (8.32) in the first iteration.*

Proof. Let x^* be an optimal solution of R-ECSPP(1). If $g^T x_c^0 \neq 0$, then $f(x^*) \in \text{conv}(\{f(x_g^k), f(x_c^k)\})$. After applying Steps 3 - 7 of NFA, it holds that $f(x_g^k)$ is the image of the minimum cost path on the left of the $f_2(x)$-axis and $f(x_c^k)$ is the image of the minimum cost path on the right of the $f_2(x)$-axis. Thus $f(x_c^k) = f(x_c^0)$ and $f(x_g^k) = f(x_{ws}^0)$. □

An alternative solution approach is a k-th shortest path algorithm based approach. To find a k-th shortest path, the algorithm proposed by Jimenez and Marzal [42] (see Chapter 7) can be used. In the following we first state an auxiliary lemma. Then prove that the special case of R-ECSPP(1) problem considered in this section, can be solved by a k-th shortest path algorithm based approach. This approach is also presented as an algorithm. R-ECSPP(1) is formulated as in (8.9)-(8.12). It is assumed that it has at least one feasible solution.

Lemma 8.29. *Let $P(v)$ denote the set of shortest paths $p \in P$ for a fixed multiplier v in Problem 8.13. If $P(v) \subset P'$ or $P(v) \subset P''$ then v is not an optimal multiplier.*

Proof. We show the proof for the case where $P(v) \subseteq P'$. For the case where $P(v) \subseteq P''$, the proof is analogous. For a fixed v, $u = \min_{p \in P}\{c(p) - vg(p)\}$ and $(u, v)^T$ are dual feasible due to construction. If there is a primal feasible solution such that complementary slackness conditions are satisfied, then $(u, v)^T$ is an optimal solution of D-ECSPP(1) and v is an optimal multiplier. Without loss of generality, let $P(v) = \{p_1, p_2, \ldots, p_{|P(v)|}\}$. For all $i \in \{|P(v)| + 1, |P(v)| + 2, \ldots, |P|\}$ it holds that $u + vg(p_i) - c(p_i) < 0$. The complementary slackness conditions imply that if $(u, v)^T$ is optimal, then there is a feasible solution $(p_1, p_2, \ldots)^T \in [0, 1]^{|P|}$ of R-ECSPP(1) such that $p_i = 0$ for all $i \in \{|P(v)|+1, |P(v)|+2, \ldots, |P|\}$. However in a primal feasible solution there should exist some $p_i > 0$, $i \in \{|P(v)| + 1, |P(v)| + 2, \ldots, |P|\}$ so that (8.11) is satisfied. This is a contradiction. □

An optimal multiplier v^* of Problem 8.14 can be found by the algorithm below. We refer to it as the find optimal multiplier algorithm (FOMA). The details of the algorithm are explained in the proof of Theorem 8.30. An LP optimum is constructed as described in Case 1 and Case 2 of the proof.

Theorem 8.30. *Find optimal multiplier algorithm terminates with an optimal multiplier $v^* \in \mathbb{R}$ of Problem 8.14.*

Proof. Let $v = 0$, then the set of shortest paths $P(0)$ can be found by applying a k-th shortest path algorithm. At the beginning the counter k is set to 0 and a shortest path p_0 is computed. The set $P(0)$ is initialized with p_0, i.e., $P(0) = \{p^0\}$. Then k is incremented to 1 and the next shortest path p_1 is computed. If $c(p_1) = c(p_0)$ then $P(0) = \{p_0, p_1\}$. This procedure is continued until $c(p_k) > c(p_0)$. If $P(0)$ is not a subset of P' or P'', then the following case distinction can be made. In both cases 0 is an optimal multiplier.

Find optimal multiplier algorithm (FOMA)
Input: The special case of R-ECSPP(1) problem described above.
Output: The optimal multiplier v^* for D-ECSPP(1).

Set $v = 0$.
Compute the set $P(0)$ by a k-th shortest path algorithm.
$z' = c(p^0)$, $p^0 \in P(0)$.
if $P(0) \subset P'$ **then**
  Compute $\hat{p} = \text{argmin}_{p \in P \setminus P'}\{c(p) + vg(p)\}$ and $\hat{z} = c(\hat{p})$.
  if $\hat{p} \in P^{AG}$ **then**
    Set $v = 0 - (\hat{z} - z')$, terminate.
  else if $\hat{p} \in P''$ **then**
    Set $v = 0 - (\frac{\hat{z} - z'}{2})$, terminate.
  end if
else if $P(0) \subset P''$ **then**
  Compute $\hat{p} = \text{argmin}_{p \in P \setminus P''}\{c(p) + vg(p)\}$ and $\hat{z} = c(\hat{p})$.
  if $\hat{p} \in P^{AG}$ **then**
    Set $v = (\hat{z} - z') - 0$, terminate.
  else if $\hat{p} \in P''$ **then**
    Set $v = (\frac{\hat{z} - z'}{2}) - 0$, terminate.
  end if
else
  Terminate.
end if

Case 1: there is an agreeable path $p_i \in P^{AG}$ in $P(0)$. In this case 0 is an optimal multiplier since a primal feasible solution and a dual feasible solution which satisfy the complementary slackness conditions can be constructed as follows. In the primal problem set $p_i = 1$ and all other path variables $p_j \in P \setminus p_i$ to 0. In the dual problem set $v = 0$ and $u = c(p_i)$. Then $p_i(u - c(p_i)) = 0$ and $p_j(u - c(p_j)) = 0$ for all $p_j \in P \setminus p_i$.

Case 2: There are at least two paths p', $p'' \in P(0)$ such that $p' \in P'$ and $p'' \in P''$. In this case 0 is an optimal multiplier since a primal feasible solution and a dual feasible solution which satisfy the complementary slackness conditions can be constructed as follows. In the primal problem set $p' = 0.5$, $p'' = 0.5$, and all other path variables $p_j \in P \setminus \{p', p''\}$ to 0. In the dual problem set $v = 0$ and $u = c(p') = c(p'')$. For this pair of solutions the complementary slackness conditions are satisfied.

Conversely if $P(0) \subset P'$ or $P(0) \subset P''$ then the following computations are performed. At this point, we assume that $P(0) \subset P'$. If $P(0) \subset P''$ the proof is analogous. Since $P(0) \subset P'$, it holds that

$$\min_{p \in P \setminus P'} \{c(p) + vg(p)\} > \min_{p \in P'} \{c(p) - v\} = u. \tag{8.33}$$

Let $\hat{p} = \text{argmin}_{p \in P \setminus P'} \{c(p) + vg(p)\}$, $\hat{z} = c(\hat{p})$ and $z' = \min_{p \in P} \{c(p)\}$. If $\hat{p} \in P^{AG}$ then by setting $v = 0 - (\hat{z} - z')$, else if $\hat{p} \in P''$ then by setting $v = 0 - (\frac{\hat{z} - z'}{2})$ the value of u can be increased. Solving Problem 8.13 with the updated multiplier v, either Case 1 or Case 2 holds for $P(v)$. □

8.4.2 Numerical results

We evaluate the performance of the MOLP based approach NFA, versus the solution approach based on the solution of R-ECSPP(1) (8.6)-(8.8) by the CPLEX LP solver. The performance measure considered for comparison of both approaches is computation time (in CPU seconds). We report the average number of iterations performed in NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver in CPU seconds. The experimental settings are described below.

- The directed acyclic graphs $G = (V, E)$ used in the numerical tests are the component trellis graphs introduced in Section 7.2 (see Example 7.1).
- The sizes of the instances are determined by the number of vertices. The number of edges in $G = (V, E)$ is approximately 2 times the number of vertices.
- The objective function coefficients c_e, $e \in E$, and the coefficients of the equality side constraints g_e, $e \in E$, are randomly chosen from a uniform distribution on the interval $[-M, M]$ where M is a positive integer.

- In the test scenarios, we either fix the value of M and vary the number of vertices or fix the number of vertices and vary the value of M.
- The number of instances solved for each setting (fixed number of vertices, fixed M value) is 100.

In the first test scenario, we fix the interval of coefficients to $[-50, 50]$ and vary the number of vertices from 8000 to 40000 by steps of 8000. The average number of iterations performed by NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus the number of vertices are reported in Table 8.1 and plotted in Figure 8.7.

Vertices	**NFA(iterations)**	**NFA(time)**	**CPLEX LP solver (time)**
8000	8.18	0.1154	4.2357
16000	9.14	0.206	14.0048
24000	9.91	0.3323	29.8796
32000	10.28	0.4491	51.2751
40000	10.61	0.7313	91.0647

Table 8.1: The average number of iterations performed in NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus number of vertices.

Our observations regarding the first test scenario are as follows:

- If the number of vertices increases, the average computation time of the CPLEX LP solver increases significantly.
- Increasing the number of vertices from 8000 to 40000 does not cause a substantial increase in the average number of iterations performed in NFA. The average number of iterations is around 10 for different numbers of vertices.
- If the number of vertices increases, there is also a small increase in the average computation time of NFA. In comparison to the increase in the average solution times of the LP solver, this increase is very small.
- NFA is faster than the CPLEX LP solver up to a factor of more than 100.

In the second test scenario, we fix the number of vertices to 40000 and vary M between the values $\{1, 5, 20, 50, 100\}$. The average number of iterations performed by NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus coefficient intervals $[-1, 1]$, $[-5, 5]$, $[-20, 20]$, $[-50, 50]$, and $[-100, 100]$ are reported in Table 8.2 and plotted in Figure 8.8.

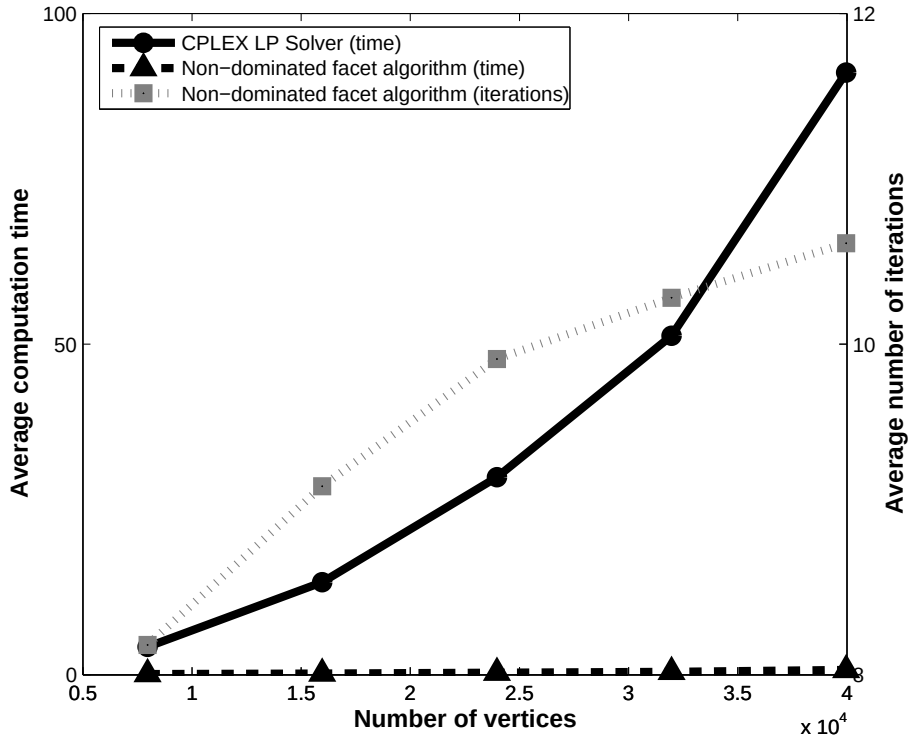


Figure 8.7: The average number of iterations performed in NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus number of vertices.

Intervals	**NFA(iterations)**	**NFA(time)**	**CPLEX LP solver (time)**
$[-1, 1]$	9.82	0.6544	69.7854
$[-5, 5]$	10.41	0.5844	77.5005
$[-20, 20]$	10.59	0.6806	86.6181
$[-50, 50]$	10.61	0.7313	91.0647
$[-100, 100]$	10.55	0.7369	90.7957

Table 8.2: The average number of iterations performed by NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus varying coefficient intervals.

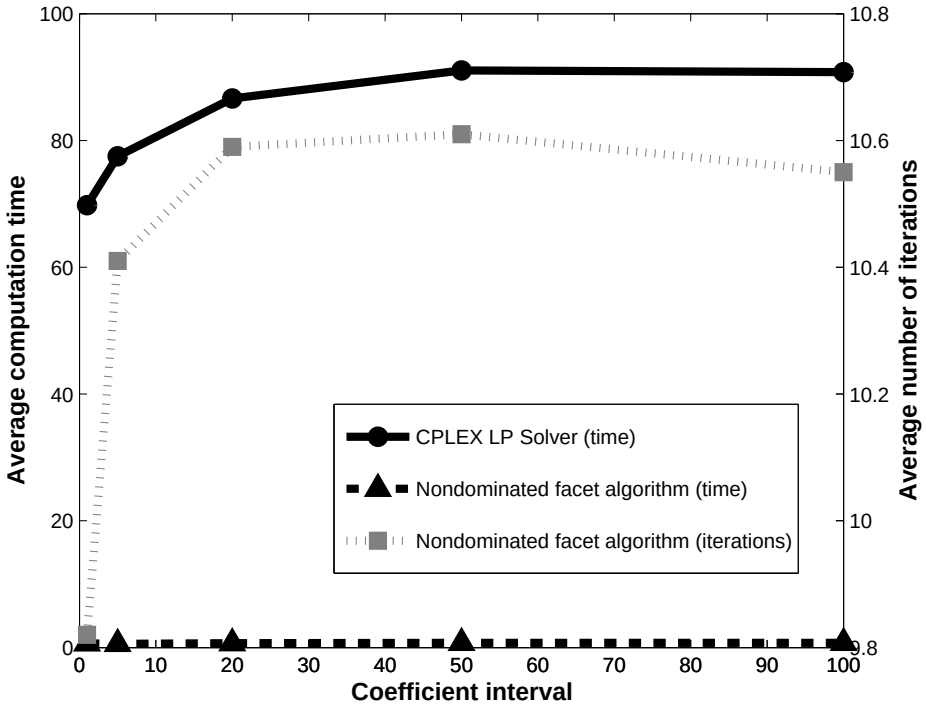


Figure 8.8: The average number of iterations performed by NFA, the average computation times of NFA, and the average computation times of the CPLEX LP solver versus varying coefficient intervals.

Our observations regarding the second test scenario are as follows:

- If the size of the coefficient interval increases, the average computation time of the CPLEX LP solver increases.
- The average computation times and the average number of iterations performed by NFA are not significantly affected by the change in the coefficient interval.
- NFA is faster than the CPLEX LP solver up to a factor of more than 100 for all chosen sizes of the coefficient interval.

In the third test scenario, we assume that the constraint $g^T x = 0$ is an interleaver consistency type of constraint. This special case of R-ECSPP(1) was discussed in Section 8.4.1. Let E_1 and E_2 be two disjoint edge sets in a component trellis graph explained in Chapter 7. In particular, they are the sets of input-1 edges from randomly chosen segments of the component trellis graph (see Example 7.1). Then we consider a side constraint of type

$$g^T x = \sum_{e \in E_1} x_e - \sum_{e \in E_2} x_e = 0. \tag{8.34}$$

The objective function coefficients are randomly chosen from a uniform distribution on the interval $[-10, 10]$.

The average computation times of NFA and the average computation times of the CPLEX LP solver versus the number of vertices are reported in Table 8.3 and plotted in Figure 8.9. Although the find optimal multiplier algorithm (FOMA) (see Section 8.4.1) is applicable to the special case of R-ECSPP(1), with our test settings this algorithm was impractical. Since FOMA applies a path enumerating approach, if the number of vertices increases, the number of paths enumerated rapidly gets very large and FOMA does not terminate in reasonable time. For this reason, the average computation times of FOMA are not reported here.

Our observations regarding the third test scenario are as follows:

- If the number of vertices increases, the average computation time of the CPLEX LP solver increases significantly.
- The average computation times of NFA are near to 0 for all different number of vertices tested.

Vertices	NFA(time)	CPLEX LP solver (time)
8000	0.0388	69.7854
16000	0.0765	77.5005
24000	0.1207	86.6181
32000	0.1357	91.0647
40000	0.1921	90.7957

Table 8.3: The average computation times of NFA, and the average computation times of the CPLEX LP solver versus number of vertices.

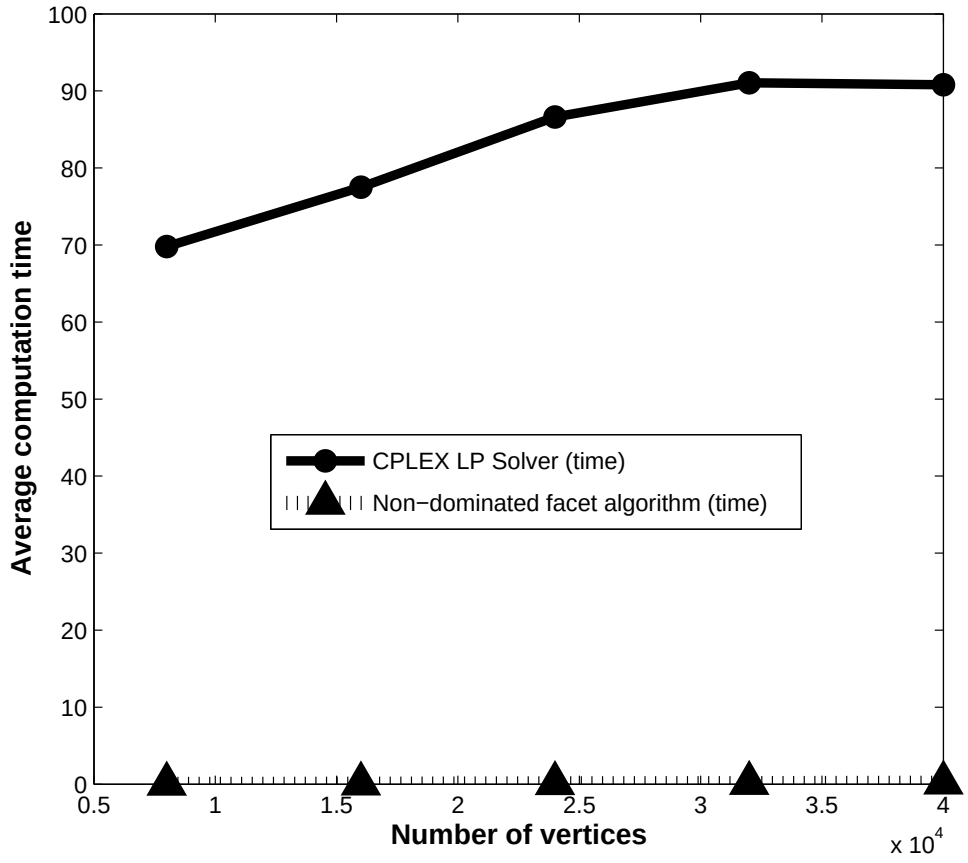


Figure 8.9: The average computation times of NFA, and the average computation times of the CPLEX LP solver versus number of vertices.

- NFA is faster than the CPLEX LP solver up to a factor of more than 100.
- The average computation times of the CPLEX LP solver and NFA with interleaver consistency type of constraint is lower than the average computation times of the CPLEX LP solver and NFA with arbitrary constraints considered in the first test scenario.

8.5 Multiple objective linear programming approach to R-ECSPP(2)

In this section, the MOLP based solution approach is extended to find an optimal solution of the LP relaxation of ECSPP(2). Given an acyclic directed graph $G = (V, E)$, $c \in \mathbb{R}^{|E|}$, $x \in \{0, 1\}^{|E|}$, and $g_t \in \mathbb{Z}^{|E|}$, $t = 1, 2$, the LP relaxation of an equality constraint shortest path problem with two side constraints is

$$\begin{aligned} &\text{minimize} && c^T x && \text{R-ECSPP(2)} && (8.35)\\ &\text{subject to} && x \in X^{Path} && && (8.36)\\ & && g_1^T x = 0 && && (8.37)\\ & && g_2^T x = 0. && && (8.38) \end{aligned}$$

As in Section 8.4, the complicating side constraints are rewritten in terms of objective functions. The original objective function of R-ECSPP(2) remains as the third objective function $f_3(x)$ whereas the linear functions $g_1^T x$ and $g_2^T x$ are considered as the first and the second objective functions $f_1(x)$ and $f_2(x)$, respectively. Consequently, the following MOLP is derived

$$\begin{aligned} &\text{minimize} && f(x) = (f_1(x), f_2(x), f_3(x)) && (8.39)\\ &\text{subject to} && x \in X^{Path}. && (8.40) \end{aligned}$$

Our study in this section is based on the geometric interpretation of Problem (8.39)-(8.40). The linear functions $f_1(x)$, $f_2(x)$, and $f_3(x)$ are introduced for the ease of notation.

The decision space is denoted by X^{Path} and the objective space is denoted by $Y^{Path} \subseteq \mathbb{R}^3$. At various steps of the solution approach, we work with the projection of the objective space Y^{Path} on the $f_1(x)$-$f_2(x)$ plane. We denote the projection by Y_{2d}^{Path}. Similar assumptions as in Section 8.4 are also made in this section. Assumption 1 and Assumption 2 exclude some trivial cases. Assumption 3 excludes the case that all points in Y^{Path} are on the same plane. This case is unlikely to occur and will not be considered in this chapter. Nevertheless we note that if all points in Y^{Path}, are on the same plane, then the solution approach would be to take a particular linear combination of $g_1^T x = 0$, $g_2^T x = 0$ and solve an R-ECSPP(1).

Assumption 1: R-ECSPP(2) is feasible,

Assumption 2: Let $x^c = \text{argmin}\{c^T x : x \in X^{Path}\}$. Then $g_1^T x^c$ and $g_2^T x^c$ are not both equal to zero.

Assumption 3: Y^{Path} is full-dimensional.

Let $x^* \in X^{Path}$ denote an optimal solution of R-ECSPP(2). In the MOLP based approach proposed in this section, we search for the facet of Y^{Path} which contains $f(x^*)$.

Observation 8.31. *Every optimal solution x^* satisfies $c^T x^* = \min\{f_3(x) : x \in X^{Path}, f_1(x) = 0, f_2(x) = 0\}$, i.e., $f(x^*)$ is located on the $f_3(x)$-axis with minimal value among all feasible points in $Y^{Path} \cap f_3(x)$.*

Example 8.32. *The geometrical interpretation stated in Observation 8.31 is illustrated exemplarily in Figure 8.10.*

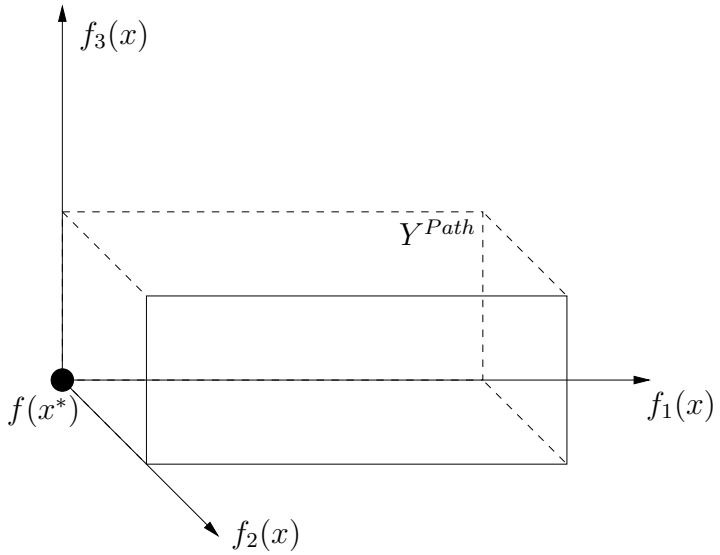


Figure 8.10: The image of the optimal solution of R-ECSPP(2), $f(x^*)$.

Definition 8.33. *A facet of Y^{Path} which contains $f(x^*)$ is called an optimal facet of Y^{Path}.*

In order to find $f(x^*)$ we propose a two stage solution approach. An optimal facet of Y^{Path} which contains $f(x^*)$ can be defined by three points p^1, p^2, and p^3 such that $(0,0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$, where p^1_{2d}, p^2_{2d}, and p^3_{2d} denote the projections of p^1, p^2, and p^3 on the $f_1(x)$-$f_2(x)$ plane. In the first stage of the solution approach, three initial points p^1, p^2, $p^3 \in Y^{Path}$ are found such that $(0,0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$ is satisfied. The first stage of the solution approach

is given in Section 8.5.1. As long as the points p^1, p^2, and p^3 are not contained in the optimal facet, a new point p^4 satisfying certain conditions substitutes one of the old points. This constitutes the second stage of the solution approach given in Section 8.5.2. It will be shown that this iterative procedure terminates with the $f(x^*)$ value after a finite number of iterations.

8.5.1 A procedure to find initial points

To find the initial points, we first find a longest path according to weight vector g_1, i.e., $x^1 = \text{argmin}\{-g_1^T x : x \in X^{Path}\}$. Let $p^1 = (g_1^T x^1, g_2^T x^1, c^T x^1) \in \mathbb{R}^3$ be the image of x^1 in the objective space Y^{Path}. The projection of p^1 onto the $f_1(x)$-$f_2(x)$ plane is denoted by $p^1_{2d} = (g_1^T x^1, g_2^T x^1)$.

In order to find $p^2 \in \mathbb{R}^3$, a shortest path x^2 according to weight vector g_1 is found, i.e., $x^2 = \text{argmin}\{g_1^T x : x \in X^{Path}\}$. Let $p^2 = (g_1^T x^2, g_2^T x^2, c^T x^2)$ be the image of x^2 in the objective space Y^{Path}. Projection of p^2 on the $f_1(x)$-$f_2(x)$ plane is denoted by $p^2_{2d} = (g_1^T x^2, g_2^T x^2)$. Since we assumed R-ECSPP(2) to be feasible, the values $g_1^T x^1$ and $g_1^T x^2$ cannot be both positive or both negative.

A third point is found as follows. We first define a line $rf_1(x) + sf_2(x) = t$ in $f_1(x)$-$f_2(x)$ plane, passing through the points p^1_{2d} and p^2_{2d}.

Lemma 8.34. *If the parameters r, t, $s \in \mathbb{R}$ are set as*

$$\begin{aligned} r &= g_2^T x^1 - g_2^T x^2, \\ s &= g_1^T x^2 - g_1^T x^1, \\ t &= r g_1^T x^1 + s g_2^T x^1, \end{aligned}$$

then equation $rf_1(x) + sf_2(x) = t$ defines a line passing through points p^1_{2d} and p^2_{2d}.

Proof. The proof follows from the definition of a line by two points. □

The line $rf_1(x) + sf_2(x) = t$ is shifted in the halfspace which is defined by $rf_1(x) + sf_2(x) = t$ and contains $(0, 0)$, until a boundary point of Y^{Path} is reached. The reason for doing this is that we want to compute a third point $p^3 \in \mathbb{R}^3$ such that $(0, 0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$.

If $t \geq 0$, then $(0, 0)$ is located in the halfspace $rf_1(x) + sf_2(x) \leq t$. Otherwise it is located in the halfspace $rf_1(x) + sf_2(x) > t$. Given two points $p^i = (g_1^T x^i, g_2^T x^i)$ and $p^j = (g_1^T x^j, g_2^T x^j)$, a procedure for finding an halfspace which contains points p^i, p^j, and $(0, 0)$, is described in the construct-halfspace algorithm.

From a MOP point of view, shifting the line $rf_1(x)+sf_2(x) = t$ in $h_0(p^1_{2d}, p^2_{2d})$ until a point on the boundary of Y^{Path}_{2d} is reached means: solve a shortest path problem in which the objective functions $f_1(x)$ and $f_2(x)$ are linearly combined

Construct-halfspace algorithm
Input: Two points $p^i = (g_1^T x^i, g_2^T x^i)$, $p^j = (g_1^T x^j, g_2^T x^j)$ on the $f_1(x)$-$f_2(x)$ plane.
Output: The halfspace $h_0(p^i, p^j)$.

1: Compute $r = g_2^T x^i - g_2^T x^j$.
2: Compute $s = g_1^T x^j - g_1^T x^i$.
3: Compute $t = r g_1^T x^i + s g_2^T x^i$.
4: **if** $t \geq 0$ **then**
5: $\quad h_0(p^i, p^j) = r f_1(x) + s f_2(x) \leq t$.
6: **else if** $t < 0$ **then**
7: $\quad h_0(p^i, p^j) = r f_1(x) + s f_2(x) > t$.
8: **end if**

in the weighted sum scalarization. The weight vector (w_1, w_2) (see 8.10) is given by

$$(w_1, w_2) = \begin{cases} (r, s) \text{ if } t \geq 0, \\ (-r, -s) \text{ if } t < 0 \end{cases} \tag{8.41}$$

A shortest path $x^3 \in X^{Path}$ and a third point p^3 are found as follows.

$$x^3 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x : x \in X^{Path}\}, \tag{8.42}$$
$$p^3 = (g_1^T x^3, g_2^T x^3, c^T x^3). \tag{8.43}$$

Theorem 8.35. *Given two points p_{2d}^1 and p_{2d}^2. If weights (w_1, w_2) of the weighted sum scalarization problem given in (8.42) are chosen as in (8.41), then $p_{2d}^3 = (g_1^T x^3, g_2^T x^3)$ is a boundary point of Y_{2d}^{Path} and $p_{2d}^3 \in h_0(p_{2d}^1, p_{2d}^2)$.*

Proof. Due to Theorem 8.11, an optimal solution x^3 of the weighted sum scalarization problem given in (8.42) is an efficient solution and its image in Y_{2d}^{Path} is a non-dominated point. It follows that $p_{2d}^3 = (g_1^T x^3, g_2^T x^3)$ is on the boundary of Y_{2d}^{Path}.

Next we show that $p_{2d}^3 \in h_0(p_{2d}^1, p_{2d}^2)$. Let $r f_1(x) + s f_2(x) = t$ denote the line passing through p_{2d}^1 and p_{2d}^2 where r, s and t are defined as in Lemma 8.34. We make the following case distinction.
Case 1:
Suppose $t \geq 0$. Then $w_1 = r$ and $w_2 = s$. It follows that

$$r g_1^T x^3 + s g_2^T x^3 \leq r0 + s0 \leq t, \tag{8.44}$$

since x^3 is an optimal solution of the weighted sum scalarization problem and point $(0, 0) \in Y_{2d}^{Path}$ is the image of some feasible solution in X^{Path}.

Case 2:
Suppose $t < 0$. Then $w_1 = -r$ and $w_2 = -s$. It follows that

$$x^3 = \text{argmax}\{rg_1^T x + sg_2^T x : x \in X^{Path}\} \Rightarrow$$
$$rg_1^T x^3 + sg_2^T x^3 \geq r0 + s0 > t,$$

since x^3 is an optimal solution of the weighted sum scalarization problem above and point $(0,0) \in Y_{2d}^{Path}$ is the image of some feasible solution in X^{Path}. □

Example 8.36. *An illustration of the procedure to find points p^1, p^2, p^3 is shown in Figures 8.11(a) to 8.11(d).*

Example 8.37. *Note that conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$) does not necessarily contain the point $(0,0)$. Such a case is illustrated in Figure 8.12.*

If $(0,0) \notin \text{conv}(\{p_{2d}^1, p_{2d}^2, p_{2d}^3\})$, then two points p_{2d}^1, p_{2d}^3 are chosen and the halfspace $h_0(p_{2d}^1, p_{2d}^3)$ is constructed. If $p_{2d}^2 \in h_0(p_{2d}^1, p_{2d}^3)$, then points p_{2d}^2 and p_{2d}^3 are selected and it is checked if $p_{2d}^1 \in h_0(p_{2d}^2, p_{2d}^3)$. Note that $p_{2d}^3 \in h_0(p_{2d}^1, p_{2d}^2)$ by construction. If $p_{2d}^2 \notin h_0(p_{2d}^1, p_{2d}^3)$ then the line $rf_1(x) + sf_2(x) = t$ passing through p_{2d}^1 and p_{2d}^3 is shifted in $h_0(p_{2d}^1, p_{2d}^2)$ until a boundary point of Y_{2d}^{Path} is reached. A new convex hull is constructed by keeping points p_{2d}^1, p_{2d}^3 and setting p_{2d}^2 to the new boundary point. An analogous procedure is applied if $p_{2d}^1 \notin h_0(p_{2d}^2, p_{2d}^3)$. The exact case distinction is given in find initial points algorithm (FIPA). The procedure above is repeated until $(0,0) \in \{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$. It is shown in Theorem 8.39 that FIPA finds three points p^1, p^2, and p^3 such that $(0,0) \in \text{conv}(\{p_{2d}^1, p_{2d}^2, p_{2d}^3\})$ in finitely many steps.

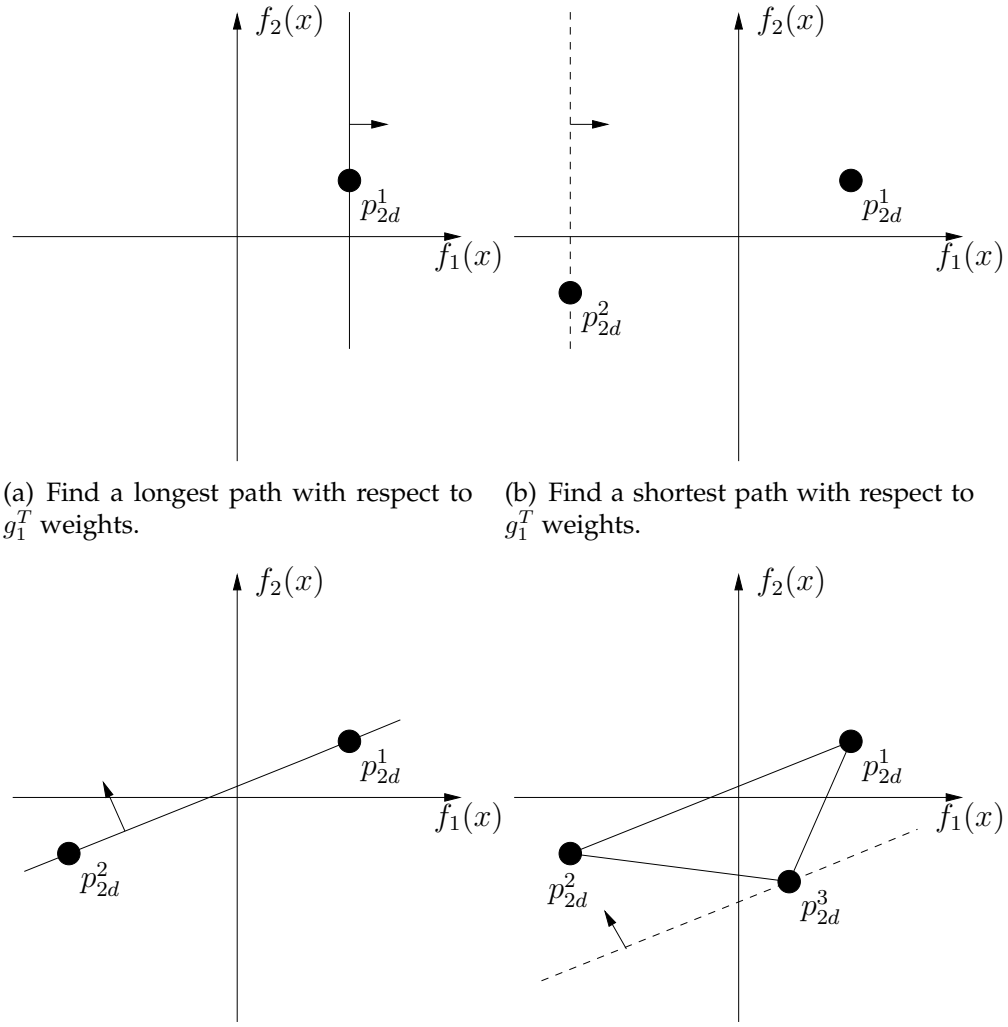
Example 8.38. *In the example illustrated in Figure 8.13, the points p_{2d}^2 and $(0,0)$ lie in different halfspaces defined by the line passing through p_{2d}^1 and p_{2d}^3. Thus, the line defined by the points p_{2d}^1 and p_{2d}^3 is shifted in the direction of $(0,0)$ until point p_{2d}^4 on the boundary of Y_{2d}^{Path} is reached. After that p_{2d}^2 is set to p_{2d}^4. With the new extreme points it holds that $(0,0) \in \{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$.*

Theorem 8.39. *At the termination step of find convex hull algorithm (FIPA), it holds that $(0,0) \in$ conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$).*

Proof. Let $\mathcal{K}$ denote the set of images of feasible paths on the boundary of Y_{2d}^{Path}. Since it is assumed that R-ECSPP(2) is feasible, $(0,0) \in \text{conv}(\mathcal{K})$. The following observations can be made on $\mathcal{K}$.

Observation 1: $|\mathcal{K}| < \infty$ since there is a finite number of paths in $G = (V, E)$.

Observation 2: There exists three points p_{2d}^1, p_{2d}^2, and $p_{2d}^3 \in \mathcal{K}$ such that $(0,0) \in \text{conv}(\{p_{2d}^1, p_{2d}^2, p_{2d}^3\})$.



(a) Find a longest path with respect to g_1^T weights.

(b) Find a shortest path with respect to g_1^T weights.

(c) Find a line passing through p^1_{2d} and p^2_{2d}.

(d) Shift the line in the halfspace which contains $(0, 0)$.

Figure 8.11: A procedure to find initial points p^1, p^2, p^3.

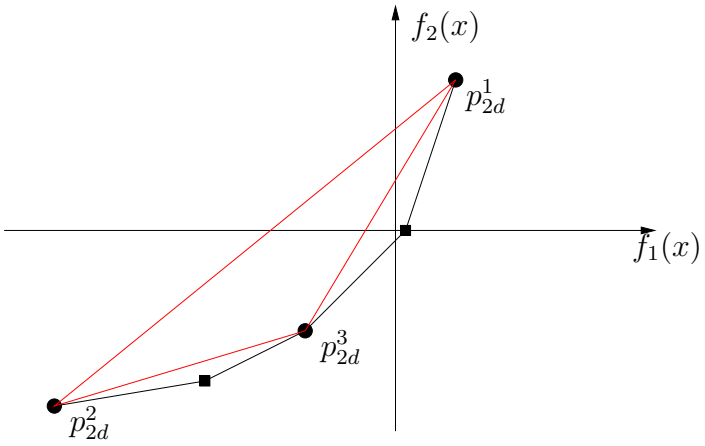


Figure 8.12: Initial triangle for R-ECSPP(2) problems.

Find initial points algorithm (FIPA)

Input: R-ECSPP(2) problem.

Output: Three points p^1, p^2, p^3 such that $(0,0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$.

Find $x^1 = \text{argmin}\{-g_1^T x : x \in X^{Path}\}$.
Find $p^1 = (g_1^T x^1, g_2^T x^1, c^T x^1)$.
Find $x^2 = \text{argmin}\{g_1^T x : x \in X^{Path}\}$.
Find $p^2 = (g_1^T x^2, g_2^T x^2, c^T x^2)$.
Call construct-halfspace algorithm with p^1_{2d} and p^2_{2d}.
Compute (w_1, w_2) according to (8.41).
Find $x^3 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x : x \in X^{Path}\}$.
Find $p^3 = (g_1^T x^3, g_2^T x^3, c^T x^3)$.
while $(0,0) \notin \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$ **do**
  Call construct-halfspace algorithm with p^1_{2d} and p^3_{2d}.
  if $p^2_{2d} \notin h_0(p^1_{2d}, p^3_{2d})$ **then**
    Compute (w_1, w_2) according to (8.41).
    Find $x^4 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x : x \in X^{Path}\}$.
    Set $p^2 = (g_1^T x^4, g_2^T x^4, c^T x^4)$.
  else
    Call construct-halfspace algorithm with p^2_{2d} and p^3_{2d}.
    if $p^1_{2d} \notin h_0(p^2_{2d}, p^3_{2d})$ **then**
      Compute (w_1, w_2) according to (8.41).
      Find $x^4 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x : x \in X^{Path}\}$.
      Set $p^1 = (g_1^T x^4, g_2^T x^4, c^T x^4)$.
    end if
  end if
end while

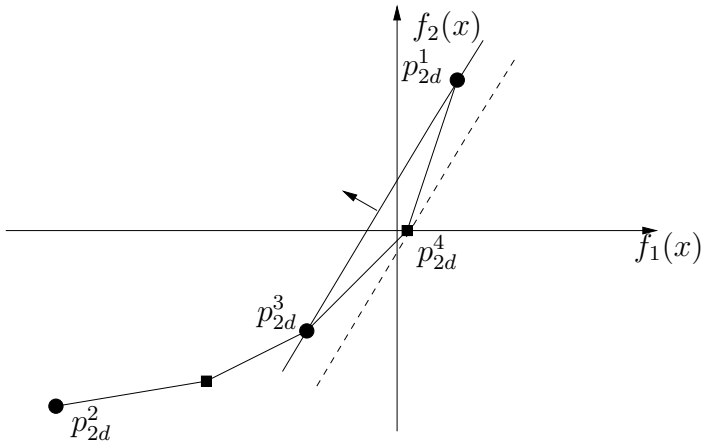


Figure 8.13: A procedure for finding a triangle which contains point $(0, 0)$.

Observation 3: FIPA computes feasible paths whose images are in $\mathcal{K}$ when computing x^i, $i \in \{1, 2, 3, 4\}$ at steps 1, 3, 7, 13, and 19 (due to Theorem 8.11).

This proof relies on the idea that the set of images of paths on the boundary which need to be considered in any subsequent iteration is reduced in each iteration. Given $\{p^1_{2d}, p^2_{2d}, p^3_{2d}\} \in \mathcal{K}$. If $(0, 0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$, then FIPA terminates. Assume therefore the opposite and that we enter the while loop for the first time. It holds that $p^2_{2d} \notin h_0(p^1_{2d}, p^3_{2d})$ or $p^1_{2d} \notin h_0(p^2_{2d}, p^3_{2d})$ since $p^3_{2d} \in h_0(p^1_{2d}, p^2_{2d})$ by construction (see Theorem 8.35).

We consider the case that $p^2_{2d} \notin h_0(p^1_{2d}, p^3_{2d})$ (the other case works analogously). Then a feasible path x^4 and p^4 such that $p^4_{2d} \in h_0(p^1_{2d}, p^3_{2d}) \cap \mathcal{K}$ is found at steps 13 and 14. This follows from Observation 3 and Theorem 8.35.

Let $\mathcal{K}^1 = \mathcal{K} \cap h_0(p^1_{2d}, p^3_{2d})$. Then, it holds that

$$(0, 0) \in \text{conv}(\mathcal{K}^1) \cap h_0(p^1_{2d}, p^3_{2d}) \tag{8.45}$$

$$\left|\mathcal{K}^1\right| < |\mathcal{K}| \tag{8.46}$$

since at least one boundary point p^2_{2d} is an element of $\mathcal{K}$ but not an element of $\mathcal{K}^1$. Moreover Observation 2 applies to $\mathcal{K}^1$ as well.

This reasoning can be applied iteratively. Since cardinality of the convex set $\mathcal{K}^i$ is decreased by at least 1 in each iteration i and since Observation 2 holds for set $\mathcal{K}^i$, three points p^1, p^2, p^3 such that $(0, 0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$ must be found in less than $|\mathcal{K}| < \infty$ iterations. □

8.5.2 A procedure to find an optimal facet

The second stage of the MOLP based approach follows from Observation 8.31, i.e., $f(x^*)$ is located on the $f_3(x)$-axis with minimal value among all points in

$Y^{Path} \cap f_3(x)$. To find an optimal facet, we propose an iterative procedure in which a sequence of shortest path problems are solved.

Given the points output by FIPA, first the plane passing through points p^1, p^2, and p^3 is constructed. In elementary plane geometry, a plane is defined in several ways. We use the definition by a point and a normal vector.

Definition 8.40. *[18] Given a point $p^1 = (g_1^T x^1, g_2^T x^1, c^T x^1)$ and a normal vector $\eta = (r, s, t)$, the equation*

$$r(f_1(x) - g_1^T x^1) + s(f_2(x) - g_2^T x^1) + t(f_3(x) - c^T x^1) = 0 \Rightarrow$$
$$rf_1(x) + sf_2(x) + tf_3(x) = u,$$

where $u = rg_1^T x^1 + sg_2^T x^1 + tc^T x^1$, defines a plane which has normal η and contains p^1.

The following two lemmas follow from the equation of a plane [18].

Lemma 8.41. *A normal vector $\eta \in \mathbb{R}^3$ of a plane which contains three points p^1, p^2, and p^3 in $\mathbb{R}^3$ is given by*

$$\eta = (p^2 - p^1) \times (p^3 - p^1) \Rightarrow$$
$$((g_1^T x^2 - g_1^T x^1), (g_2^T x^2 - g_2^T x^1), (c^T x^2 - c^T x^1)) \times$$
$$((g_1^T x^3 - g_1^T x^1), (g_2^T x^3 - g_2^T x^1), (c^T x^3 - c^T x^1)).$$

where the $\times$ operator denotes the cross product.

Lemma 8.42. *According to the basic cross product relationships, the components of the normal vector $\eta = (r, s, t) \in \mathbb{R}^3$ are given as*

$$r = (g_2^T x^2 - g_2^T x^1)(c^T x^3 - c^T x^1) - (c^T x^2 - c^T x^1)(g_2^T x^3 - g_2^T x^1),$$
$$s = (c^T x^2 - c^T x^1)(g_1^T x^3 - g_1^T x^1) - (g_1^T x^2 - g_1^T x^1)(c^T x^3 - c^T x^1),$$
$$t = (g_1^T x^2 - g_1^T x^1)(g_2^T x^3 - g_2^T x^1) - (g_2^T x^2 - g_2^T x^1)(g_1^T x^3 - g_1^T x^1).$$

The plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ divides $\mathbb{R}^3$ into two halfspaces. We denote the halfspace which contains the point $(0, 0, -\infty)$ by $h_{neg}(p^1, p^2, p^3)$. Due to Observation 8.31, $f(x^*)$ is located on the $f_3(x)$-axis with minimal value among all points in $Y^{Path} \cap f_3(x)$. In an attempt to find $f(x^*)$ we shift the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ in $h_{neg}(p^1, p^2, p^3)$ (negative $f_3(x)$ direction) until a boundary point of Y^{Path} is reached. This is done by setting weights (w_1, w_2, w_3) as in (8.47)

$$(w_1, w_2, w_3) = \begin{cases} (r, s, t) \text{ if } t > 0, \\ (-r, -s, -t) \text{ if } t < 0 \end{cases} \tag{8.47}$$

and solving Problem 8.48

$$\min\{w_1 g_1^T x + w_2 g_2^T x + w_3 c^T x : x \in X^{Path}\}. \tag{8.48}$$

Theorem 8.43. *Given three points p^1, p^2, and $p^3 \in \mathbb{R}^3$. If weights (w_1, w_2, w_3) are chosen as in (8.47), then for the weighted sum scalarization problem in (8.49)*

$$x^4 = \text{argmin}\{w_1 f_1(x) + w_2 f_2(x) + w_3 f_3(x) : x \in X^{Path}\} \tag{8.49}$$

$p^4 = f(x^4)$ is a boundary point of Y^{Path} and $p^4 \in h_{neg}(p^1, p^2, p^3)$.

Proof. Due to Theorem 8.11, an optimal solution x^3 of the weighted sum scalarization problem given in (8.49) is an efficient solution and its image in Y^{Path} is a non-dominated point. It follows that $p^4 = f(x^4)$ is on the boundary of Y^{Path}.

If $t > 0$, then the normal vector $\eta = (r, s, t)$ points at positive $f_3(x)$ direction. It follows that $p^4 \in h_{neg}(p^1, p^2, p^3)$ since the minimization direction is the opposite direction of the direction that η points at.

If $t < 0$, then the normal vector $\eta = (-r, -s, -t)$ points at positive $f_3(x)$ direction. It follows that $p^4 \in h_{neg}(p^1, p^2, p^3)$ since the minimization direction is the opposite direction of the direction that η points at. □

Example 8.44. *In Figure 8.14 the points p^1, p^2 and p^3 are assumed to be those points returned by FIPA. After finding the components of the normal vector η, the plane which contains these points is shifted in negative $f_3(x)$ direction until the boundary point p^4 is reached.*

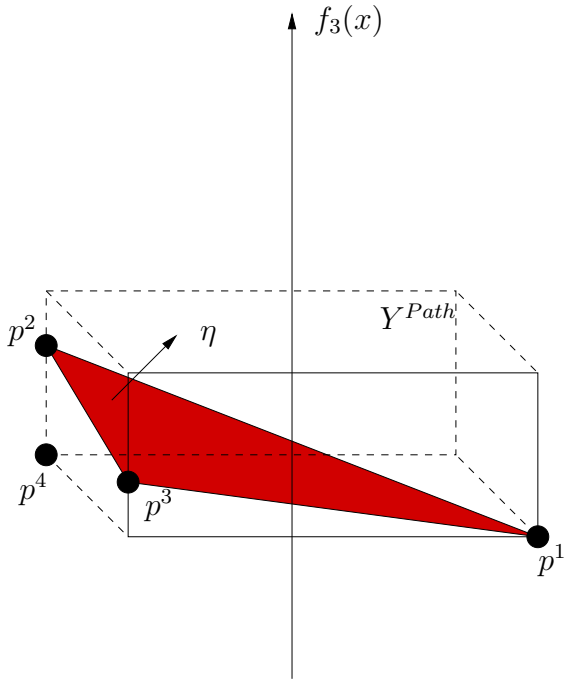


Figure 8.14: Finding a boundary point.

Let $x^4 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x + w_3 c^T x : x \in X^{Path}\}$ and $p^4 = f(x^4)$. If p^4 is on the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$, then the objective function value of R-ECSPP(2) is the $f_3(x)$ coordinate of the intersection point of $rf_1(x) + sf_2(x) + tf_3(x) = u$ with the $f_3(x)$-axis. Below, this claim is stated as a Theorem 8.45 and proved.

Theorem 8.45. *Let x^* be an optimal solution of R-ECSPP(2). If p^4 is located on the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ defined by points p^1, p^2, and p^3, then $f(x^*)$=$(0, 0, \frac{u}{t})$.*

Proof. The plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ contains the points

$$\begin{aligned} p^1 &= (g_1^T x^1, g_2^T x^1, c^T x^1), \\ p^2 &= (g_1^T x^2, g_2^T x^2, c^T x^2), \\ p^3 &= (g_1^T x^3, g_2^T x^3, c^T x^3), \\ p^4 &= (g_1^T x^4, g_2^T x^4, c^T x^4). \end{aligned}$$

Furthermore, $(0, 0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$. It follows from the definition of convex hull that there exists k_1, k_2, $k_3 \in \mathbb{R}$ such that

$$k_1, k_2, k_3 \geq 0, \tag{8.50}$$
$$k_1 + k_2 + k_3 = 1, \tag{8.51}$$
$$k_1 f_1(x^1) + k_2 f_1(x^2) + k_3 f_1(x^3) = 0, \tag{8.52}$$
$$k_1 f_2(x^1) + k_2 f_2(x^2) + k_3 f_2(x^3) = 0. \tag{8.53}$$

Next we extend Formulation (8.9)-(8.12) to include a second side constraint.

$$\text{minimize} \quad \sum_{p \in P} c(p)p \qquad \text{(ECSPP(2))} \tag{8.54}$$
$$\text{subject to} \quad \sum_{p \in P} p = 1 \tag{8.55}$$
$$\sum_{p \in P} g_1(p)p = 0 \tag{8.56}$$
$$\sum_{p \in P} g_2(p)p = 0 \tag{8.57}$$
$$p \geq 0 \qquad \text{for all } p \in P. \tag{8.58}$$

We set the values of path variables in the new formulation (8.54)-(8.58) so that the value of the path p_1 modeled by x^1 is set to k_1, the value of the path p_2 modeled by x^2 is set to k_2, and the value of path p_3 modeled by x^3 is set to k_3. The values of all paths $p \in P \setminus \{p_1, p_2, p_3\}$ are set to 0. Obviously $(k_1, k_2, k_3, 0, \ldots, 0)^T \in [0, 1]^{|P|}$ is a feasible solution of Problem (8.54)-(8.58).

Next, we consider the weighted sum scalarization of Problem (8.39)-(8.40). The weights are determined according to the value of t as in Equation (8.47).

$$\min\{w_1 f_1(x) + w_2 f_2(x) + w_3 f_3(x) \text{ s.t. } x \in X^{Path}\}. \tag{8.59}$$

If the image of the optimal solution of Problem 8.59, p^4, is on the plane $r f_1(x) + s f_2(x) + t f_3(x) = u$ which also contains points p^1, p^2, and p^3, then p^1, p^2, and p^3 are also images of alternative optimal solutions of Problem 8.59. This problem can equivalently be written as

$$\min\{\frac{w_1}{w_3} f_1(x) + \frac{w_2}{w_3} f_2(x) + f_3(x) \text{ s.t. } x \in X^{Path}\}. \tag{8.60}$$

Suppose that the weight vector $w^* = (w_1^*, w_2^*, w_3^*)$ where $0 < w_1^* < 1$, $0 < w_2^* < 1$, $0 < w_3^* < 1$, and $w_1^* + w_2^* + w_3^* = 1$ is chosen such that x^1 (p_1), x^2 (p_2), and x^3 (p_3) are three optimal solutions of Problem 8.60.

Now we consider the dual of R-ECSPP(2) (Formulation (8.54)-(8.58)), D-ECSPP(2).

maximize u	(D-ECSPP(2))
subject to $u + v_1 g_1(p) + v_2 g_2(p) \leq c(p)$	for all $p \in P$.

For fixed v_1 and v_2 values, it holds that

$$u(v_1, v_2) = \min_{p \in P}\{c(p) - v_1 g_1(p) - v_2 g_2(p)\}. \tag{8.61}$$

If we set $v_1 = -\frac{w_1}{w_3}$ and $v_2 = -\frac{w_2}{w_3}$ then the paths p_1, p_2, and p_3 are optimal solutions of Problem 8.61 and the vector $(u(-\frac{w_1}{w_3}, -\frac{w_2}{w_3}), -\frac{w_1}{w_3}, -\frac{w_2}{w_3}) \in \mathbb{R}^3$ is dual feasible.

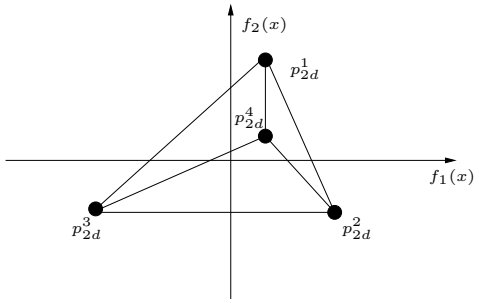
Analogously to the proof of Theorem 8.21, $(k_1, k_2, k_3, 0, \ldots, 0)^T \in [0,1]^{|P|}$ and $(u(-\frac{w_1}{w_3}, -\frac{w_2}{w_3}), -\frac{w_1}{w_3}, -\frac{w_2}{w_3}) \in \mathbb{R}^3$ satisfy complementary slackness conditions. Consequently, $(k_1, k_2, k_3, 0, \ldots, 0)^T \in [0,1]^{|P|}$ is an optimal solution of R-ECSPP(2). □

If p^4 is not located on the plane $r f_1(x) + s f_2(x) + t f_3(x) = u$, then point p^4 is projected onto the $f_1(x)$-$f_2(x)$ plane. The projection of p^4 may be located inside or outside conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$).

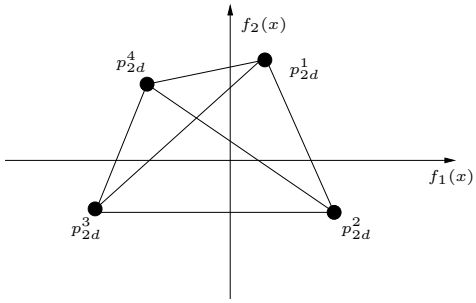
Example 8.46. *An illustration of the location of p_{2d}^4 w. r. t. conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^3\}$).*

In Observation 8.47, it is stated that the projection of p^4 is in one of the convex hulls which can be defined with 3 points from the set $\{p_{2d}^1, p_{2d}^2, p_{2d}^3, p_{2d}^4\}$.

Observation 8.47. *Let $\{p_{2d}^1, p_{2d}^2, p_{2d}^3, p_{2d}^4\}$ be 4 points on the $f_1(x)$-$f_2(x)$ plane, such that point $(0,0) \in$ conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^3, p_{2d}^4\}$). Then $(0,0)$ is located in one of the convex hulls: conv($\{p_{2d}^1, p_{2d}^2, p_{2d}^4\}$), conv($\{p_{2d}^1, p_{2d}^3, p_{2d}^4\}$), or conv($\{p_{2d}^2, p_{2d}^3, p_{2d}^4\}$).*



(a) Point p^4 is located in conv($\{p^1_{2d}, p^2_{2d}, p^3_{2d}\}$).



(b) Point p^4 is not located in conv($\{p^1_{2d}, p^2_{2d}, p^3_{2d}\}$).

If $(0,0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^4_{2d}\})$ then p^3 is set to p^4, else if point $(0,0) \in \text{conv}(\{p^1_{2d}, p^3_{2d}, p^4_{2d}\})$ then point p^2 is set to p^4, else point p^1 is set to p^4. With the new points, the procedure described above is repeated. Termination condition is stated in Theorem 8.45. The iterative procedure applied to find the optimal objective function value of R-ECSPP(2) is given in the find optimal facet algorithm (FOFA) below.

Remark 8.48. *Steps 20-22 of FOFA are necessary to avoid cycling which is discussed in the proof of Theorem 8.50.*

Remark 8.49. *The value of t computed as in Lemma 8.42 cannot be 0 because*

in Assumption 3 we assumed that Y^{Path} is full dimensional,

FIPA outputs three non-collinear points,

between steps 23 - 29 3 non-collinear points are selected.

Theorem 8.50. *Let x^* denote the optimal solution of R-ECSPP(2). The find optimal facet algorithm (FOFA) outputs $f(x^*)$ after a finite number of iterations.*

Proof. Let $\mathcal{K}$ denote the set of images of feasible paths on the boundary of Y^{Path}. The proof uses the idea that there exist finitely many triples $(p^1, p^2, p^3) \in \mathcal{K}$ such that $(0,0) \in \text{conv}(\{p^1_{2d}, p^2_{2d}, p^3_{2d}\})$ and FOFA works through these triples without choosing the same triple twice until the optimal facet is found. Let l be the iteration counter of FOFA. The triple (p^1, p^2, p^3) used to define the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ in iteration l is denoted by $(p^1, p^2, p^3)^l$.

Observation 1: $|\mathcal{K}| < \infty$ since there is a finite number of paths in $G = (V, E)$.

Observation 2: In each iteration l, the plane defined by $(p^1, p^2, p^3)^l$ contains a point $(0, 0, v^l) \in (Y^{Path} \cap f_3(x))$, i.e., the plane defined by $(p^1, p^2, p^3)^l$ intersects the $f_3(x)$ axis at the point $(0, 0, v^l)$.

Find optimal facet algorithm (FOFA)
Input: R-ECSPP(2) problem.
Output: The optimal objective function value of R-ECSPP(2).

Call FIPA.
Set terminate=false.
while terminate=false **do**
  Construct plane $rf_1(x) + sf_2(x) + tf_3(x) = u$.
  if $t > 0$ **then**
    Set $(w_1, w_2, w_3) = (r, s, t)$.
  else
    Set $(w_1, w_2, w_3) = (-r, -s, -t)$.
  end if
  Find $x^4 = \text{argmin}\{w_1 g_1^T x + w_2 g_2^T x + w_3 c^T x : x \in X^{Path}\}$.
  Find $p^4 = (g_1^T x^4, g_2^T x^4, c^T x^4)$.
  if $g_1^T x^4 = 0$ and $g_2^T x^4 = 0$ **then**
    Optimal objective function value is $c^T x^4$.
    Set terminate=true.
  end if
  if p^4 is on the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ **then**
    Optimal objective function value is $\frac{u}{t}$.
    Set terminate=true.
  end if
  if $(0,0) \in \text{conv}(\{p_{2d}^i, p_{2d}^j\})$, $i, j \in \{1,2,3\}$, $i \neq j$ **then**
    Reset indices such that $i = 1$, $j = 2$ and the index of the third point is 3.
  end if
  if $(0,0) \in \text{conv}(\{p_{2d}^4, p_{2d}^2, p_{2d}^3\})$ and $\{p_{2d}^4, p_{2d}^2, p_{2d}^3\}$ are non-collinear **then**
    Set $p^1 = p^4$.
  else if $(0,0) \in \text{conv}(\{p_{2d}^4, p_{2d}^1, p_{2d}^3\})$ and $\{p_{2d}^4, p_{2d}^1, p_{2d}^3\}$ are non-collinear **then**
    Set $p^2 = p^4$.
  else if $(0,0) \in \text{conv}(\{p_{2d}^4, p_{2d}^1, p_{2d}^2\})$ and $\{p_{2d}^4, p_{2d}^1, p_{2d}^2\}$ are non-collinear **then**
    Set $p^3 = p^4$.
  end if
end while

Observation 3: For increasing number of iterations l, the values of v^l are non-increasing. This is due to the fact that the new point p^4 found in iteration l is obtained by shifting the plane defined by $(p^1, p^2, p^3)^l$ in the negative $f_3(x)$ direction.

At Step 10 of any iteration l, the algorithm computes a feasible path x^4 such that $f(x^4) \in \mathcal{K} \cap h_{neg}(p^1, p^2, p^3)^l$, i.e., $f(x^4)$ is a boundary point of Y^{Path} and it is in the halfspace $h_{neg}(p^1, p^2, p^3)^l$ which contains p^1, p^2, p^3, and $(0, 0, -\infty)$. For a given x^4, the following cases can occur.
Case 1: If $g_1^T x^4 = 0$ and $g_2^T x^4 = 0$ then x^4 is an optimal solution of R-ECSPP(2) and ECSPP(2). The algorithm terminates.
Case 2: If $p^4 = (g_1^T x^4,\ g_2^T x^4,\ c^T x^4)$ is on the plane $rf_1(x) + sf_2(x) + tf_3(x) = u$ which is defined by $(p^1, p^2, p^3)^l$, then according to Theorem 8.45 $f(x^*) = (0, 0, \frac{u}{t})$ and x^* can be found by solving the equation system (8.50)-(8.53). The algorithm terminates.
Case 3: If Case 1 and Case 2 do not hold, then the boundary point $p^4 = f(x^4)$ is not on the plane defined by $(p^1, p^2, p^3)^l$. According to the case distinctions between Step 20 and Step 29, one of the points p^1, p^2, or p^3 is set to p^4. A new plane defined by $(p^1, p^2, p^3)^{l+1}$ is constructed in iteration $l + 1$. To show that a triple $(p^1, p^2, p^3)^l \in \mathcal{K}$ such that $(0, 0) \in \text{conv}(\{p_{2d}^1, p_{2d}^2, p_{2d}^3\})$ is considered at most once in FOFA, we make the following case distinction.
Case 3a: If $v^{l+1} < v^l$ then $v^{l+1} < v^m$ for all $m \in \{1, \ldots, l\}$ due to Observation 3. It follows that the triple from iteration $l + 1$ is considered for the first time in FOFA and the triples $(p^1, p^2, p^3)^m$ for all $m \in \{1, \ldots, l\}$ won't be considered in the following iterations.
Case 3b: If $v^{l+1} = v^l$, then this corresponds to the following special case. Let i and j be two distinct elements of the set $\{1, 2, 3\}$ such that $(0, 0)$ is in the convex hull of $\{p_{2d}^i, p_{2d}^j\}$. Depending on the position of p^4 the following case distinction can be made.

Case 3b-1: The third point of the triple denoted by p_{2d}^k and p_{2d}^4 are on the same side of the line passing through p_{2d}^i, $(0, 0)$, and p_{2d}^j. This implies $v^{l+1} = v^l$ because p^4 can only replace the third point of the triple between steps 23 - 29. Since the third point of the triple in iteration l, is not in $h_{neg}(p^1, p^2, p^3)^{l+1}$, triple $(p^1, p^2, p^3)^l$ will not be considered by FOFA a second time. Due to the fact that there exists finitely many points which satisfy the conditions described in Case 3b-1, after finitely many iterations all these triples are exhausted.

Case 3b-2: The third point of the triple denoted by p_{2d}^k and p_{2d}^4 are on the opposite sides of the line passing through p_{2d}^i, $(0, 0)$, and p_{2d}^j. If FOFA replaces p_{2d}^k with p_{2d}^4 in iteration $(l+1)$ and p_{2d}^4 with p_{2d}^k in iteration $(l+2)$, cycling occurs. To avoid cycling FOFA tries to first replace one of the points p_{2d}^i or p_{2d}^j. This is done at Steps between 20 - 22. Setting indices new, i.e., $i = 1$, $j = 2$ and the index of the third point to 3, it is guaranteed that first p^i or p^j is replaced by p^4 between steps 23 - 29. □

8.5.3 Numerical results

Using the same experimental settings as described in Section 8.4.2 we evaluate the performance of MOLP based approach FOFA, versus the solution approach where R-ECSPP(2) (8.35)-(8.38) is solved by the CPLEX LP solver. The performance measure considered for the comparison of both approaches is the computation time (measured in CPU seconds). We report the average number of iterations performed in FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver. At the beginning, FOFA calls the find initial points algorithm (FIPA). FIPA is an iterative algorithm but in our numerical experiments the average number of iterations performed in FIPA was always approximately 0. This means FIPA finds 3 initial points without performing significantly many iterations. Therefore, the iterations performed in FIPA are not reported here.

In the first test scenario, we fix the interval of coefficients to $[-50, 50]$ and vary the number of vertices from 8000 to 40000 by steps of 8000. The average number of iterations performed by FOFA, the average computation times of FOFA and the average computation times of the CPLEX LP solver versus the number of vertices are reported in Table 8.4 and plotted in Figure 8.15.

Vertices	**FOFA(iterations)**	**FOFA(time)**	**CPLEX LP solver (time)**
8000	21.95	0.1984	3.7671
16000	24.65	0.4401	13.1875
24000	25.95	0.7282	28.1804
32000	27.44	1.0442	48.9683
40000	27.53	1.2667	72.3677

Table 8.4: The average number of iterations performed in FOFA, the average computation times of FOFA and the average computation times of the CPLEX LP solver versus number of vertices.

Our observations regarding the first test scenario are as follows:

- If the number of vertices increases, the average computation time of the CPLEX LP solver increases significantly.

- If the number of vertices increases, there is also a small increase in the average computation time of FOFA. However, compared to the increase in the average computation time of the LP solver, this increase is very small.

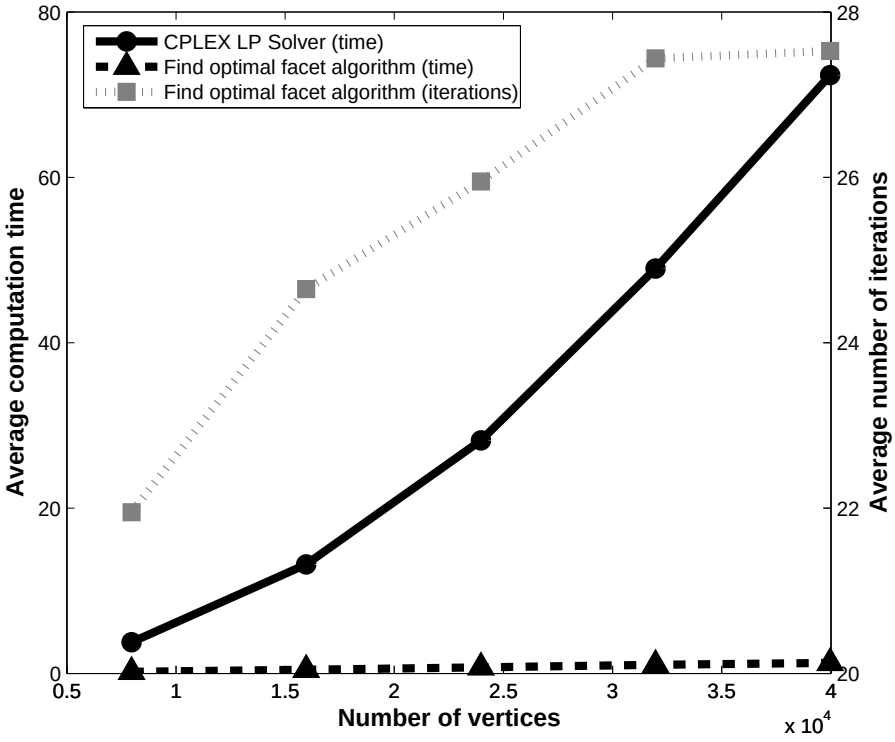


Figure 8.15: The average number of iterations performed in FOFA, the average computation times of FOFA and the average computation times of the CPLEX LP solver versus number of vertices.

- Increasing the number of vertices from 8000 to 40000 does not cause a substantial increase in the average number of iterations performed by FOFA. The average number of iterations performed by FOFA varies between 20 and 30.

- FOFA is faster than the CPLEX LP solver up to a factor of 70.

In the second test scenario, we fix the number of vertices to 40000 and vary M between the values $\{1, 5, 20, 50, 100\}$. The average number of iterations of FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus coefficient intervals $[-1, 1]$, $[-5, 5]$, $[-20, 20]$, $[-50, 50]$, and $[-100, 100]$ are reported in Table 8.5 and plotted in Figure 8.16.

Intervals	**FOFA(iterations)**	**FOFA(time)**	**CPLEX LP solver (time)**
$[-1, 1]$	11.42	0.6198	56.6427
$[-5, 5]$	17.62	0.832	93.6618
$[-20, 20]$	26.61	1.2401	83.2267
$[-50, 50]$	27.53	1.2667	72.3677
$[-100, 100]$	28.38	1.2919	69.8042

Table 8.5: The average number of iterations performed in FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus varying coefficient intervals.

Our observations regarding the second test scenario are as follows:

- If the size of the coefficient interval increases, the average computation time of FOFA and the average number of iterations performed by FOFA increase. The reason for this is probably that for a large coefficient interval, Y^{Path} has more vertices.

- We observed also in our numerical experiments which are not reported in this section that CPLEX LP solver has the largest average computation time for $M = 5$ when M is chosen from the set $\{1, 5, 20, 50, 100\}$. An explanation for this may be that there exist a lot of feasible solutions for the interval $[-5, 5]$. If the size of the coefficient interval increases, the number of solutions satisfying the equality side constraints decreases.

- FOFA is faster than the CPLEX LP solver up to a factor of more than 100.

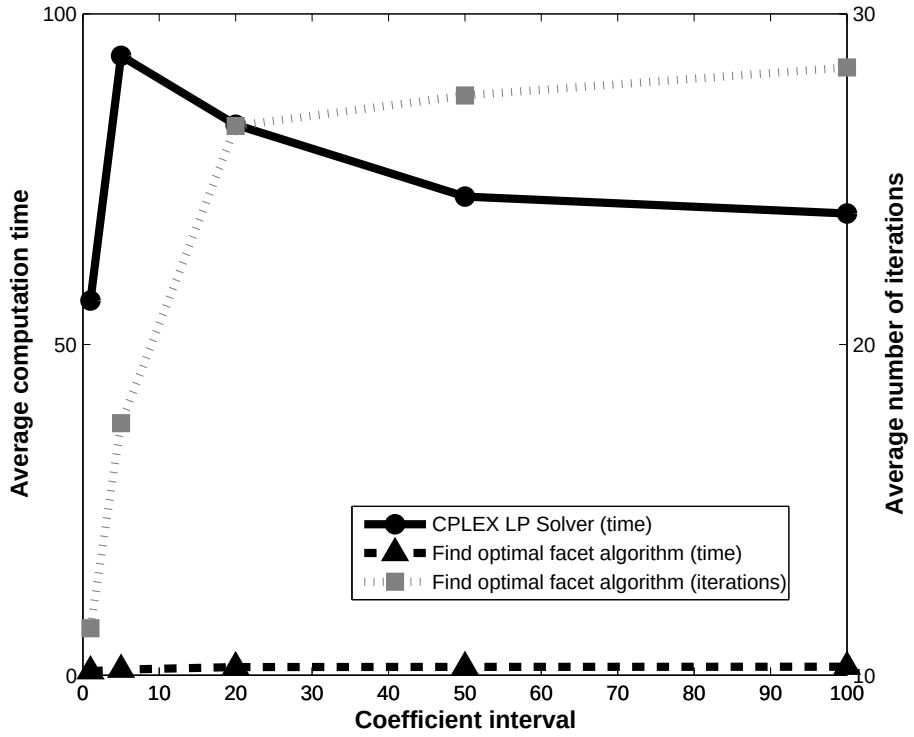


Figure 8.16: The average number of iterations performed in FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus varying coefficient intervals.

In the third test scenario, we assume that the constraints $g_1^T x = 0$ and $g_2^T x = 0$ are interleaver consistency type of constraints mentioned in Chapter 7). Let E_1, E_2, E_3, and E_4 be four disjoint edge sets in a component trellis graph explained in Chapter 7). In particular, they are the sets of input-1 edges from randomly chosen segments of the component trellis graph (see Example 7.1). Then we consider side constraints of type

$$g_1^T x = \sum_{e \in E_1} x_e - \sum_{e \in E_2} x_e = 0, \tag{8.62}$$

$$g_2^T x = \sum_{e \in E_3} x_e - \sum_{e \in E_4} x_e = 0. \tag{8.63}$$

The objective function coefficients are randomly chosen from a uniform distribution on the interval $[-10, 10]$. The average number of iterations of FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus the number of vertices are reported in Table 8.6 and plotted in Figure 8.17.

Vertices	**FOFA(iterations)**	**FOFA(time)**	**CPLEX LP solver (time)**
8000	4.37	0.049	1.7956
16000	4.37	0.112	7.4917
24000	4.16	0.1778	17.5222
32000	4.4	0.2439	31.6284
40000	4.25	0.3003	47.6228

Table 8.6: The average number of iterations performed in FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus number of vertices.

Our observations regarding the third test scenario are as follows:

- If the number of vertices increases, the average computation time of the CPLEX LP solver increases significantly.
- If the number of vertices increases, there is also a small increase in the average computation time of FOFA.
- If the number of vertices increases, there is no significant change in the average number of iterations performed by FOFA.
- FOFA is faster than the CPLEX LP solver up to a factor of more than 100.

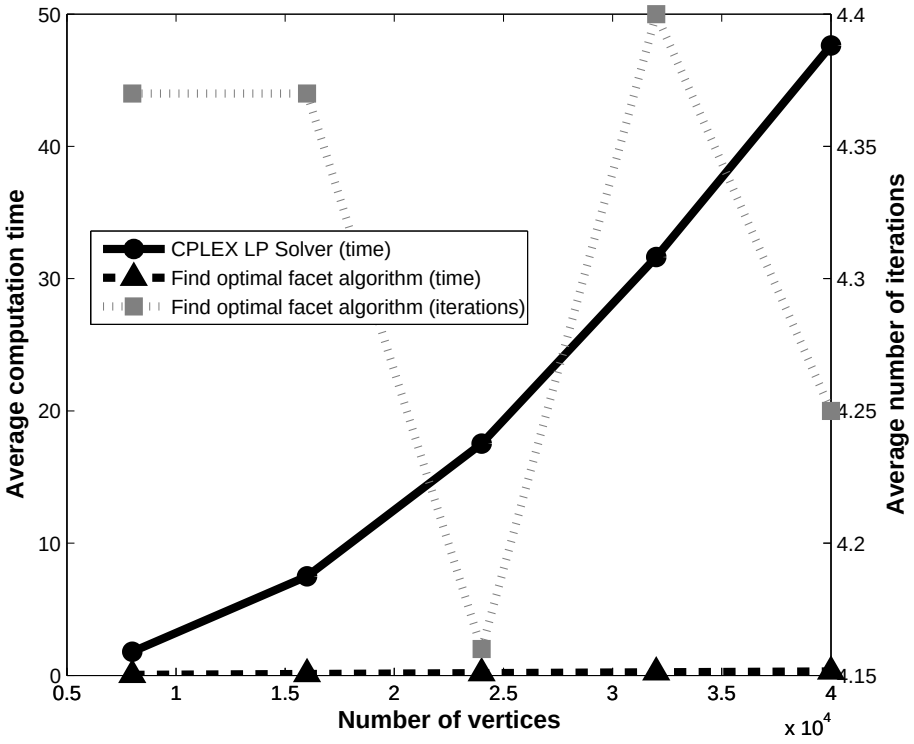


Figure 8.17: The average number of iterations performed in FOFA, the average computation times of FOFA, and the average computation times of the CPLEX LP solver versus number of vertices.

- The average computation times of the CPLEX LP solver and FOFA with interleaver consistency type of constraints is lower than the average computation times of the CPLEX LP solver and FOFA with arbitrary constraints considered in the first test scenario.

8.6 Conclusion and further research

In this chapter, we discussed shortest path problems with 1 or 2 equality side constraints. We proposed a MOLP based solution algorithm to solve the LP relaxation of ECSPP(1). Then we concentrated on a special case of R-ECSPP(1) where the equality constraint was interleaver consistency type of constraint introduced in Chapter 7. We showed that for this special case, NFA finds the non-dominated facet containing the image of an optimal solution in the first iteration. An extension of the MOLP approach to R-ECSPP(2) was also studied. Our numerical experiments showed that NFA finds the optimal objective function value of R-ECSPP(1) up to a factor of more than 100 times faster than the CPLEX LP solver. Similarly, FOFA finds the optimal objective function value of R-ECSPP(2) up to a factor of more than 100 times faster than the CPLEX LP solver.

We have already begun the work of developing a branch and bound type of algorithm to close the duality gap (if it exists) and find an optimal solution to ECSPP(1) and ECSPP(2). An essential further research direction is to generalize the solution approach that was developed for R-ECSPP(2) so that it applies to R-ECSPP(k) problems. Alternatively, approaches which decompose or transform a R-ECSPP(k) to several R-ECSPP(2) or R-ECSPP(1) problems can be investigated. Such a research would facilitate the use of MOLP based approaches introduced in this chapter in LP based decoding methods for LTE turbo codes proposed in Chapter 7.

Chapter 9

Conclusion and further research

A major part of this thesis covers mathematical programming based decoding approaches for binary linear codes. In the literature, these approaches are studied by researchers working in the field of information theory where applied mathematics and electrical engineering overlap. To the best of our knowledge, the problem of maximum likelihood decoding (MLD) of binary linear codes was first studied as an optimization problem in [9] by Breitbach et al. Later, articles from Feldman, Wainwrigth and Karger, e.g., [24], [26] aroused interest on linear programming decoding. This interest has grown rapidly over the years. Figure 9.1 illustrates the trend in the number of publications on linear programming decoding from 2002 to 2009.

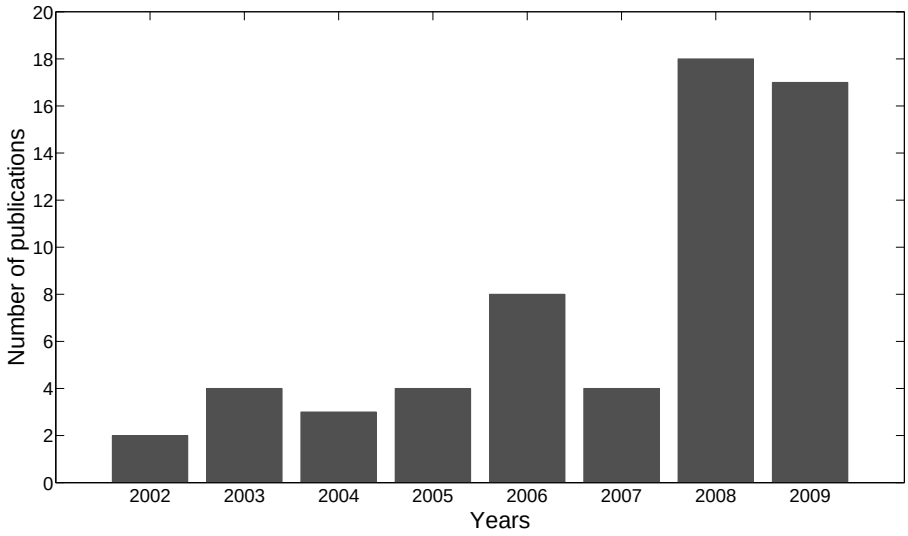


Figure 9.1: Publications on linear programming decoding between years 2002 and 2009 (to the best of authors knowledge).

Many of these publications are categorized and summarized in this thesis. Concepts from linear programming, linear programming duality, polyhedral theory, integer programming, network flows, non-linear programming, and matroid theory have been used to develop decoding algorithms that are efficient in terms of computational complexity and error-correcting performance.

From a mathematical programming point of view, maximum likelihood decoding (MLD) problem of binary linear codes is the problem of minimizing a linear cost function over the convex hull of all codewords, the so-called codeword polytope. Studying polyhedral properties of the codeword polytope is of crucial importance to develop efficient integer programming based solution approaches. There is a one-to-one correspondence between binary linear codes and binary matroids and the same polytope as the codeword polytope is defined as the cycle polytope of a binary matroid in matroid theory. The cycle polytope is the convex hull of the incidence vectors of all cycles of a binary matroid. Polyhedral properties of cycle polytope are studied by Barahona and Grötschel [5] as well as Grötschel and Truemper [35], [36]. These properties directly apply to codeword polytope. We translated some results from binary matroids to binary linear codes. An interesting observation we made is that polyhedral properties of complete binary matroids introduced in [36] directly apply to the codeword polytope of simplex codes. This means a complete and non-redundant description of the codeword polytope of any simplex code can be attained. Matroid decomposition techniques are useful to determine code classes for which MLD can be performed in polynomial time. We showed that binary matroids associated to Hamming codes can not be decomposed into smaller matroids via 2 and 3 sums.

MLD and minimum distance problems are NP-hard integer programming (IP) problems and in general they can not be solved in polynomial time. However for some instances, a reasonable solution approach is to model the problem with a cost function and constraints which are linear in the n dimensional real space and then to solve the formulation with a general purpose IP solver. We applied this approach to MLD and minimum distance problems. We were able to compute frame error rates for MLD and minimum distances for some interesting code classes. A key success factor here is to model the MLD and minimum distance problems such that existing all purpose solvers can deal efficiently with them. To this aim, we numerically compared several alternative IP formulations. For LDPC and BCH codes, starting with few constraints and variables then iteratively adding cuts seems to be favorable. For LTE turbo codes, a network flows based formulation performs best in terms of computation time. We conclude that it is worthwhile to make an analysis of different formulations for different code classes since the selected formulation effects the sizes of solvable instances.

A first step toward the solution of an IP problem is solving its LP relaxation. In linear programming decoding (LPD), the linear cost function of the IP for-

mulation for MLD is minimized on a relaxed polytope. If an LPD algorithm outputs a codeword, then it is the maximum likelihood (ML) codeword. Otherwise an error is output. We proposed a new IP formulation of the MLD problem which has some advantages compared to other formulations proposed in the literature. We developed a novel cutting plane method, called separation algorithm decoding (SAD), to solve the IP problem. Our simulations demonstrate that SAD outperforms existing LP based decoding algorithms in terms of error-correcting performance. SAD is not limited to the LDPC codes, since it can, for instance, be used for decoding algebraic codes like BCH codes. Furthermore, we introduced a new class of valid inequalities and showed that they define facets of the codeword polytope of the $(8, 4, 4)$ Hamming code.

If the code classes at hand provide more structure, e.g., for turbo-like codes the MLD problem turns into a side constrained shortest path problem, combinatorial algorithms based on Lagrangian duality theory, network theory etc. can be used to develop efficient decoding algorithms. We presented a novel IP formulation for MLD of LTE turbo codes. This IP has a tighter LP relaxation than the conventional LP relaxation. As a new decoding method for LTE turbo codes we proposed solving the dual problem of a Lagrangian relaxation, in conjunction with a k-th shortest path algorithm to close the duality gap. Simulation results show that our algorithm gives superior FER performance than that of LPD.

Motivated by the problem of decoding of LTE turbo codes we defined a new class of side constrained shortest path problem (SCSPP) on a directed acyclic graph. Various articles addressing SCSPP where the side constraint is a less than or equal to constraint exist in the literature. Different from these articles, we considered an equality constraint with integral coefficients and zero right hand side. We referred to the new class of SCSPP by equality constraint shortest path problem ECSPP(1). It can be shown that ECSPP(1) is NP-hard. We concentrated on solving the LP relaxation of ECSPP(1). The approach we used is a variant of an approach introduced in [59] to solve the LP relaxation of SCSPP with less than or equal to constraint. The side constraint is interpreted as an objective function and ECSPP(1) is reformulated as a multiple objective linear programming (MOLP) problem. The basic idea is to work in the objective space and solve a sequence of shortest path problems to identify the non-dominated facet subsuming the optimal solution and interpolating. This yields the LP-optimum. We then extended the MOLP approach to solve the LP relaxation of ECSPP with two equality side constraints ECSPP(2).

The content of this thesis has an interdisciplinary nature. To continue the research on mathematical programming decoding, apart from familiarity with results from different fields of mathematical programming, it is important to have a solid knowledge of code classes, channel models, state-of-the-art decoding methods etc. There are numerous new questions originated by our observations. We mention some of them in the following.

Compared to iterative message passing decoding (IMPD), a major drawback of LPD is its complexity. Lately, it has been shown that LPD can be approximated in linear time complexity (see [10]) for LDPC codes. Some research on special purpose solvers employing interior point algorithms have also been made by Vontobel [69], Wadayama [72], and Taghavi et al. [62]. Further research can be focused on lowering the time complexity of improved (in terms of error correcting performance) LPD.

Coding theory is closely related to linear algebra. Mathematical optimization approaches can be combined with algebraic approaches. In particular, BCH codes have some special algebraic structure which might be of use. Let x be a codeword of a BCH code. Shifting the bits of x one position left or right yields again a codeword. Such properties can be used in polyhedral analysis and in solution algorithms.

Forbidden set inequalities are facets of a codeword polytope if the associated binary matroid has the sum of circuits property. If the associated binary matroid does not have the sum of circuits property then the codeword polytope does not have only facets in the form of forbidden set inequalities. We made some observations on facet defining inequalities by employing a collection of routines[1] for analyzing polytopes and polyhedra. It might be worthwhile investigating special structures of facets. Given a code class, an interesting research direction is to analyze facet defining inequalities of the codeword polytope.

The mathematical models and solution approaches we used to solve the MLD problem for various classes of binary linear codes have aroused many fundamental research questions in integer programming. One such research question which was discussed in Chapter 8 is the problem of finding efficient combinatorial algorithms to solve the LP relaxation of ECSPP(1) and ECSPP(2). Some other research areas motivated by MLD of binary linear codes are:

- Finding suitable IP formulations for the problem of minimizing a real valued linear objective function over a constraint set defined in $\mathbb{F}_q$ where q is a positive integer.
- Multiple objective optimization on a polytope which is completely described by forbidden set type inequalities.
- Given a problem $\min\{c^T x : Ax = b, x \geq 0, x \in \mathbb{Z}^n\}$ where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \{0,1\}^{m \times n}$. Identify special cases based on the distribution of ones and zeros in A such that this problem can be solved easily.

[1] PORTA, http://www.iwr.uni-heidelberg.de/groups/comopt/software/PORTA/

List of abbreviations

LPD: adaptive linear programming decoding
BCH: Bose-Chaudhuri-Hocquenghem
BCQPD: box constrained quadratic programming decoding
BIAWGNC: binary input additive white Gaussian noise channel
BLPD: bare linear programming decoding
BSC: binary symmetric channel
CGA1: cut generation algorithm 1
CGA2: cut generation algorithm 2
CHA: construct $\hat{H}$ algorithm
ECSPP: equality constrained shortest path problem
EFGD: exhaustive facet guessing decoding
FDA: fractional distance algorithm
FFG: Forney style factor graph
FIPA: find initial points algorithm
FOMA: find optimal multiplier algorithm
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
MD: minimum Hamming distance
ML: maximum likelihood
MLD: maximum likelihood decoding
MOP: multiple objective optimization
MOLP: multiple objective linear programming

MSD: min-sum decoding
NFA: non-dominated facet algorithm
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
R-ECSPP: relaxed equality constrained shortest path problem
RA: repeat accumulate
RBGD: randomized bit guessing decoding
RFGD: randomized facet guessing decoding
RIPD: relaxed integer programming decoding
RPC: redundant parity-check
SAD: separation algorithm decoding
SCSPP: side constrained shortest path problem
SD: subgradient decoding
SDLPD: softened dual linear programming decoding
SNR: signal to noise ratio
SPC: single parity-check
SPD: sum-product decoding
SPLP: shortest path linear program
WSSP: weighted sum scalarization problem

List of Tables

List of Figures

Bibliography

[1] *ILOG,CPLEX 11 Users Manual, 2007*.

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

[3] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46:325–343, 2000.

[4] F. Barahona. The max-cut problem in graphs not contractible to K_5. *Operations Research Letters*, 2:107–111, 1983.

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

[6] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, pages 954–972, 1978.

[7] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel. Designing good permutations for turbo codes: towards a single model. In *Proceedings of IEEE International Conference on Communications*, volume 1, pages 341–345, 2004.

[8] D. Bertsimas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, NJ, USA, 1989.

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

[10] D. Burshtein. Linear complexity approximate LP decoding of LDPC codes: Generalizations and improvements. In *Proceedings of 5th International Symposium on Turbo Codes and Related Topics, Lausanne*, 2008.

[11] M. Chertkov. Reducing the error floor. In *Proceedings of Information Theory Workshop, 2007*, pages 230–235, 2007.

[12] 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 46th Annual Allerton Conference on Communication, Control and Computing, Monticello, Illinois, September*, 2006.

[13] M. Chertkov and V. Y. Chernyak. Loop calculus in statistical physics and information science. *Physical Review E*, 73:65–102, 2006. arXiv.org:cond-mat/0601487.

[14] M. Chertkov and V. Y. Chernyak. Loop series for discrete statistical models on graphs. *Journal of Statistical Mechanics*, page P06009, 2006.

[15] M. Chertkov and M. Stepanov. Pseudo-codeword landscape. In *Proceedings of IEEE International Symposium on Information Theory*, pages 1546–1550.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[17] S. Crozier, P. Guinand, and A. Hunt. Estimating the minimum distance of turbo-codes using double and triple impulse methods. *IEEE Communications Letters*, 9:631–633, 2005.

[18] P. Dawkins. *Calculus III, Online Notes*. http://tutorial.math.lamar.edu/Classes/CalcIII/EqnsOfPlanes.aspx.

[19] A. G. Dimakis, A. A. Gohari, and M. J. Wainwright. Guessing facets: Polytope structure and improved LP decoding. *IEEE Transactions on Information Theory*, 55:4134–4154, 2009.

[20] 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*, 2007.

[21] J. Edmonds and L. Johnson. Matching, Euler tours and the Chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[22] M. Ehrgott. *Multicriteria Optimization*. Springer, Berlin, 2000.

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

[24] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. In *Proceedings of 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 251 – 260, 2002.

[25] J. Feldman and D. R. Karger. Decoding turbo-like codes via linear programming. *Journal of Computer and System Sciences*, 68:733 – 752, 2004.

[26] J. Feldman, D. R. Karger, and M. J. Wainwright. Linear programming-based decoding of turbo-like codes and its relation to iterative approaches. In *Proceedings of 40th Allerton Conference on Communications, Control, and Computing*, 2002.

[27] J. Feldman, M. J. Wainwright, and D. R. Karger. Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51:954–972, 2005.

[28] G. D. Forney Jr. Codes on graphs: Normal realizations. *IEEE Transactions on Information Theory*, 47:529–548, 2001.

[29] R. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8:21–28, 1962.

[30] R. Garello, P. Pierleoni, and S. Benedetto. Computing the free distance of turbo codes and serially concatenated codes with interleavers: algorithms and applications. *IEEE Journal on Selected Areas in Communications*, 19:800–812, 2001.

[31] M. Garey and D. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[32] I. Goldenberg and D. Burshtein. Error bounds for repeat-accumulate codes decoded via linear programming. In *Proceedings of 6th International Symposium on Turbo Codes and Related Topics, Brest*, 2010.

[33] 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*, pages 99–120, 2004.

[34] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, Berlin, Heidelberg, 1988.

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

[36] M. Grötschel and K. Truemper. Master polytopes for cycles in binary matroids. *Linear Algebra and its Applications*, 114/115:523–540, 1989.

[37] 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*, 51:265–280, 2005.

[38] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[39] S. Heupel. Cycle polytopes and their application in coding theory. Master's thesis, University of Kaiserslautern, 2009.

[40] IEEE 802.11n. Wireless LAN Medium Access Control and Physical Layer specifications: Enhancements for Higher Throughput. IEEE P802.11n/D3.0, September 2007.

[41] IEEE 802.16e. Air Interface for Fixed and Mobile Broadband Wireless Access Systems.

[42] V. Jimenez and A. Marzal. Computing the k shortest paths. A new algorithm and an experimental comparison. In *Proceedings of 3rd Workshop on Algorithm Engineering (WAE99), LNCS 1668*, pages 15–29, 1999.

[43] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[44] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[45] N. Kashyap. A decomposition theory for binary linear codes. *IEEE Transactions on Information Theory*, 54:3035–3058, 2008.

[46] S. O. Krumke. *Integer Programming, Polyhedra and Algorithms*. 2006.

[47] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[48] Lovász L. and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991.

[49] T. Larsson and Z. Liu. A Lagrangian relaxation scheme for structured linear programs with application to multicommodity network flows. *Optimization*, 40(3):247–284, 1997.

[50] S. Lin and D. J. Costello Jr. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[51] D. J. C. MacKay. Encyclopedia of sparse graph codes. [Online; Stand 30. October 2009].

[52] M. Minoux and C. Ribeiro. A transformation of hard (equality constrained) knapsack problems into constrained shortest path problems. *Operations Research Letters*, 3:211–214, 1984.

[53] M. Miwa, T. Wadayama, and I. Takumi. A cutting plane method based on redundant rows for improving fractional distance. *Journal on Selected Areas in Communications*, 27:1005–1012, 2009.

[54] T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, Hoboken, NJ, 2005.

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

[56] A. Nimbalker, Y. Blankenship, B. Classon, and T. K. Blankenship. Arp and qpp interleavers for LTE turbo coding. In *Proceedings of IEEE Wireless Communications and Networking Conference WCNC 2008*, pages 1032–1037, 2008.

[57] J. G. Oxley. *Matroid Theory*. Oxford University Press, New York, 1992.

[58] M. Punekar, F. Kienle, N. Wehn, Tanatmis A., Ruzika S., and Hamacher H. W. Calculating the minimum distance of linear block codes via integer programming. In *Proceedings of 6th International Symposium on Turbo Codes and Related Topics, Brest*, 2010.

[59] S. Ruzika. *On Multiple Objective Combinatorial Optimization*. PhD thesis, University of Kaiserslautern, 2007.

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

[61] P. D. Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory Series B*, 28:305–359, 1980.

[62] M. H. Taghavi, A. Shokrollahi, and P.H. Siegel. Efficient implementation of linear programming decoding. In *Proceedings of 46th Annual Allerton Conference on Communication, Control and Computing, Monticello, Illinois, September*, 2008.

[63] M. H. Taghavi and P. H. Siegel. Adaptive methods for linear programming decoding. *IEEE Transactions on Information Theory*, 54:5396–5410, 2008.

[64] A. Tanatmis, S. Ruzika, H. W. Hamacher, M. Punekar, F. Kienle, and N. Wehn. A separation algorithm for improved LP-decoding of linear block codes. *IEEE Transactions on Information Theory*, 56:3277 – 3289, 2010.

[65] R. M. Tanner, D. Sridhara, and T. Fuja. A class of group-structured LDPC codes. In *Proceedings of International Conference on Information Systems Technology and its Applications*, 2001.

[66] Third Generation Partnership Project. *3GPP TS 36.212 V8.5.0; 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8)*, 2008.

[67] W. T. Tutte. Connectivity in matroids. *Canadian Journal of Mathematics*, 18:1301–1324, 1966.

[68] A. Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43:1757–1766, 1997.

[69] P. O. Vontobel. Interior-point algorithms for linear-programming decoding. In *Proceedings of Information Theory and its Applications Workshop, UC San Diego, La Jolla, CA, USA*, 2008.

[70] P. O. Vontobel and R. Kötter. Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes. *accepted for IEEE Transactions on Information Theory*, 2007.

[71] P. O. Vontobel and R. Kötter. On low-complexity linear-programming decoding of LDPC codes. *European Transactions on Telecommunications*, 18:509–517, 2007.

[72] 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*, pages 389 – 393.

[73] D. J. A. Welsh. *Matroid Theory*. L. M. S. Monographs. Academic Press, London - New York, 1976.

[74] N. Wiberg. *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, 1996.

[75] Y. Xiao, K. Thulasiraman, G. Xue, and Jüttner A. The constrained shortest path problem: Algorithmic approaches and an algebraic study with generalization. *AKCE International Journal on Graphs and Combinatorics*, 2:63–86, 2005.

[76] K. Yang, J. Feldman, and X. Wang. Nonlinear programming approaches to decoding low-density parity-check codes. *IEEE Journal on Selected Areas in Communications*, 24:1603–1613, 2006.

[77] K. Yang, X. Wang, and J. Feldman. A new linear programming approach to decoding linear block codes. *IEEE Transactions on Information Theory*, 54:1061–1072, 2008.

[78] M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43:441–466, 1991.

[79] A. Yufit, A. Lifshitz, and Y. Be'ery. Efficient linear programming decoding of HDPC codes. *Submitted to IEEE Transactions on Communications*, 2009.

[80] M. Ziegelmann. *Constrained Shortest Paths and Related Problems*. PhD thesis, Max-Planck Institute für Informatik, 2001.

Curriculum Vitae

06/2002	B.Sc., Bilkent University, Industrial Engineering, Ankara/Türkei
05/2003 – 06/2006	Wissenschaftliche Hilfskraft bei KEIPER GmbH, Kaiserslautern
04/2006	Diplom in Mathematik, verliehen von der Technischen Universität Kaiserslautern
seit 10/2006	Doktorand in der Arbeitsgruppe Optimierung des Fachbereichs Mathematik der Technischen Universität Kaiserslautern

06/2002	B.Sc., Bilkent University, Industrial Engineering, Ankara/Turkey
05/2003 – 06/2006	Student assistant in KEIPER GmbH, Kaiserslautern
04/2006	Diploma in mathematics from the University of Kaiserslautern
since 10/2006	Ph.D. student in optimization research group, department of mathematics, University of Kaiserslautern