2018

Computational Methods for Discrete Conic Optimization Problems

Aykut Bulut
Lehigh University

Computational Methods for Discrete Conic Optimization Problems

by

Aykut Bulut

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Doctor of Philosophy

in

Industrial and Systems Engineering

Lehigh University

January 2018

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Date

_____
Dissertation Advisor

Committee Members:

_____
Dr. Ted Ralphs, Committee Chair

_____
Dr. Pietro Belotti

_____
Dr. Julio C. Góez

_____
Dr. Tamás Terlaky

_____
Dr. Luis Zuluaga

First of all, I would like to thank Dr. Ted Ralphs for accepting me as a student, being a great mentor and a friend. He has my eternal gratitude for his patience with me, his trust in me and giving me the freedom to pursue my own ideas.

I am grateful to my thesis committee members; Dr. Belotti, Dr. Góez, Dr. Terlaky and Dr. Zuluaga for fruitful discussions, editing the earlier drafts of my dissertation and the feedback they have provided. This work would not be possible without their input.

I appreciate all the great people of Lehigh ISE and CORAL Lab. They have provided a wonderful research environment.

Elcin, Onur, Emre, Omid, Amy and Matt have been awesome friends. It would be much harder for me to finish this thesis without their support and friendship.

I would like to thank my parents for their support and encouragement.

And finally, I am grateful to my beloved wife Yagmur for her endless love and being there for me when I needed the most.

Contents

List of Tables

List of Figures

Abstract

This thesis addresses computational aspects of discrete conic optimization. We study two well-known classes of optimization problems closely related to mixed integer linear optimization problems. The case of mixed integer second-order cone optimization problems (MISOCP) is a generalization in which the requirement that solutions be in the non-negative orthant is replaced by a requirement that they be in a second-order cone. Inverse MILP, on the other hand, is the problem of determining the objective function that makes a given solution to a given MILP optimal.

Although these classes seem unrelated on the surface, the proposed solution methodology for both classes involves outer approximation of a conic feasible region by linear inequalities. In both cases, an iterative algorithm in which a separation problem is solved to generate the approximation is employed. From a complexity standpoint, both MISOCP and inverse MILP are NP–hard. As in the case of MILPs, the usual decision version of MISOCP is NP-complete, whereas in contrast to MILP, we provide the first proof that a certain decision version of inverse MILP is rather co-NP-complete.

With respect to MISOCP, we first introduce a basic outer approximation algorithm to solve SOCPs based on a cutting-plane approach. As expected, the performance of our implementation of such an algorithm is shown to lag behind the well-known interior point method. Despite this, such a cutting-plane approach does have promise as a method of producing bounds when embedded within a state-of-the-art branch-and-cut implementation

due to its superior ability to warm-start the bound computation after imposing branching constraints. Our outer-approximation-based branch-and-cut algorithm relaxes both integrality and conic constraints to obtain a linear relaxation. This linear relaxation is strengthened by the addition of valid inequalities obtained by separating infeasible points. Valid inequalities may be obtained by separation from the convex hull of integer solution lying within the relaxed feasible region or by separation from the feasible region described by the (relaxed) conic constraints. Solutions are stored when both integer and conic feasibility is achieved. We review the literature on cutting-plane procedures for MISOCP and mixed integer convex optimization problems.

With respect to inverse MILP, we formulate this problem as a conic problem and derive a cutting-plane algorithm for it. The separation problem in this algorithm is a modified version of the original MILP. We show that there is a close relationship between this algorithm and a similar iterative algorithm for separating infeasible points from the convex hull of solutions to the original MILP that forms part of the basis for the well-known result of Grötschel-Lovász-Schrijver that demonstrates the complexity-wise equivalence of separation and optimization.

In order to test our ideas, we implement a number of software libraries that together constitute `DisCO`, a full-featured solver for MISOCP. The first of the supporting libraries is `OsiConic`, an abstract base class in `C++` for interfacing to SOCP solvers. We provide interfaces using this library for widely used commercial and open source SOCP/nonlinear problem solvers. We also introduce `CglConic`, a library that implements cutting procedures for MISOCP feasible set. We perform extensive computational experiments with DisCO comparing a wide range of variants of our proposed algorithm, as well as other approaches. As DisCO is built on top of a library for distributed parallel tree search algorithms, we also perform experiments showing that our algorithm is effective and scalable when parallelized.

Notation

$\mathbb{R}$	Set of real numbers.
$\mathbb{R}_+$	Set of non-negative real numbers.
$\mathbb{Q}$	Set of rational numbers.
$\mathbb{Z}$	Set of integer numbers.
$\mathbb{L}^n$	n dimensional Lorentz cone, i.e., $\{x \in \mathbb{R}^n \vert x_1 \geq \|x_{2:n}\|\}$.
$\alpha, \beta, \ldots$	Lower Greek letters denote scalars.
$\|\alpha\|$	Absolute value of scalar α.
$a, b, x, \ldots$	Lower case letters denote column vectors except when used for indices, $i, j, k, \ldots$. The distinction will be made obvious within the context.
x_i	i-th element of vector x.
$x_{i:j}$	Sub-vector of x, formed by elements of x from i to j.
$\|x\|$	Euclidean norm of x.
$\|x\|_p$	p-norm of x.
$X, S, \ldots$	Upper case letters denote matrices or index sets.

$\mathcal{A}, \mathcal{B}, \ldots$	Calligraphic upper case letters denote sets.
$\mathcal{Q}(Q, q, \rho)$	Triplet that denotes set $\{x \in \mathbb{R}^n \vert x^\top Q x + 2q^\top x + \rho \leq 0\}$
$f_i(x)$	Small case letters for mathematical function names.

Chapter 1

Introduction

1.1 Motivation

A *mathematical optimization problem* is the problem of finding a point in a given *feasible region* that minimizes the value of a given *objective function*. More precisely, such a problem is to find

$$x^* \in \operatorname{argmin}_{x \in \mathcal{F}} f(x) \tag{OPT}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and

$$\mathcal{F} = \{x \in \mathcal{X} \mid g_i(x) \leq 0 \; \forall i \in M\}$$

is the feasible region, described by *constraint functions* $g_i : \mathbb{R}^n \to \mathbb{R}$ for i in finite set M indexing the set of constraints and a set $\mathcal{X} \subseteq \mathbb{R}^n$, usually indicating the requirement that some variables take on integer values. A point x in $\mathcal{F}$ is called a *solution* and the point x^* is an *optimal solution* and need not be unique. In this thesis, we use the convention that $f(x^*) = -\infty$ means (OPT) is unbounded and $f(x^*) = \infty$ means infeasible.

Mathematical optimization can be thought of as a generalization of the problem of

finding a solution to a system of mathematical inequalities. The goal in mathematical optimization is not only to find a solution that satisfies the given constraints but also find the one that optimizes the given objective function among all the solutions.

Like many algorithms, all computational approaches discussed in this study work within an error tolerance. When determining whether a given solution is either optimal or feasible, we apply the following definitions for a chosen value of ϵ.

Definition 1.1. (ϵ-feasibility) A given point $x \in \mathbb{Q}^n$ is said to be ϵ-feasible for a positive ϵ, if the following holds,

$$\min_{y \in \mathcal{F}} \|x - y\| \leq \epsilon.$$

The set

$$\mathcal{F}_\epsilon := \{y \in \mathbb{R}^n \mid \exists t \in \mathcal{F} \ \|y - t\| \leq \epsilon\}$$

of all ϵ-feasible solutions is call the *epsilon-feasible set*.

Definition 1.2. (ϵ-optimality) A given point $x \in \mathbb{Q}^n$ is said to be ϵ-optimal for a positive ϵ, if the following holds,

$$f(x) \leq \min_{y \in \mathcal{F}_\epsilon} f(y).$$

Mathematical optimization can be thought of as a generalization of the problem of finding a solution to a system of mathematical inequalities. The goal in mathematical optimization is not only to find a solution that satisfies the given constraints but also find the one that optimizes the given objective function among all the solutions. Optimization problems can be classified according to the form of the objective and constraint functions, as well as the form of set $\mathcal{X}$. The most widely studied form of mathematical optimization

problem is the *linear optimization problem* (LP), whose standard form is

$$\min c^\top x \tag{LPa}$$

$$\text{s.t. } Ax = b \tag{LPb}$$

$$x \in \mathbb{R}^n_+, \tag{LPc}$$

where $A \in \mathbb{Q}^{m\times n}$, $c \in \mathbb{Q}^n$ and $b \in \mathbb{Q}^m$. Here, the objective and all constraint functions are linear, while $\mathcal{X} = \mathbb{R}^n_+$ (the variables are all *continuous* and nonnegative). LPs are the most basic form of mathematical optimization problems and can be considered to be a core form to which more complex constraints may be added. Throughout this chapter, the notation above will thus be used to denote similar input data for classes of optimization that generalize (LP). One such class is the *mixed integer linear optimization problem* (MILP), which is similar to an LP except that we have $\mathcal{X} = \mathbb{Z}^r_+ \times \mathbb{R}^{n-r}_+$, i.e., some of the variables (those with index less than or equal to r, called *integer variables*), must take on integer values. Although MILPs are difficult to solve in theory (see Section 1.3), they are a well-studied class with specialized solution methods that are highly effective in practice.

Nonlinear optimization problems (NLPs) are those in which the objective and constraint functions are not assumed to be linear. Within this broad class, there is a further division according to whether the objective function and feasible regions are *convex* or *non-convex*. These two classes have very different properties. Historically, most research about non-convex NLPs has focused on finding *locally optimal* solutions, which are, roughly speaking, those satisfying conditions that ensure they are at least optimal within a local neighborhood. Finding globally optimal solutions to problems for which the feasible region is convex is an efficiently solvable problem in general, while non-convex problems are at least as difficult to solve as MILPs.

Until relatively recently, technologies for solving MILPs and NLPs have been devel-

oped relatively independently due to the fact that solution methods for these two broad classes of problems seem on the surface to have little in common. With solution technologies for these two classes of problems having now matured, there has been a more recent movement towards development of more general solvers for a broader class that encompasses both MILPs and NLPs. This class is a generalization of the NLP in which we have $\mathcal{X} = \mathbb{Z}_+^r \times \mathbb{R}_+^{n-r}$. Such problems are known as *mixed integer nonlinear optimization problems* (MINLPs). It has now become feasible to develop practical solvers for this very general class of problems that was considered highly intractable until only a decade ago. The development of effective solvers for MINLPs requires the integration of methods for nonlinear and discrete optimization problems into a single, coherent whole. Achieving this coherency is not easy, especially when developing a solver targeted at this most general classes of problems. For this reason, it will be some time before such solvers are robust enough to be used in production settings. It is therefore natural to consider more restricted classes on which progress might be made more quickly or on which there are efficiencies precluded by consideration of more general cases.

A topic of investigation in this dissertation is computational aspects of *second-order cone optimization problems* (SOCPs), a class that generalizes the class of LPs and bridges the gap between linear and nonlinear optimization. In particular, SOCPs are convex nonlinear problems in which the non-negativity constraints (LPc) of (LP) are replaced by the requirement that the solutions lie in one or more second-order cones (see Section 1.2.2 for formal definition). Just like LPs, SOCPs can be solved efficiently using the *Interior Point Method* (IPM) (see Section 2.1.2), but are a general enough class that they can be used to model many interesting problems that could not be modeled or compactly approximated as pure linear optimization problems.

The *mixed integer second-order cone optimization problem* (MISOCP) is a generalization of the SOCP where some variables must take on integer values. An MISOCP is

formed by requiring some of the variables of an SOCP to take integer values. MISOCP is a broad and extremely important class of optimization problem in practice. Because many theoretical properties of MILPs, such as subadditive duality, extend naturally to MISOCPs, we expect that this class can eventually be made as tractable as MILPs. This dissertation aims to develop practical techniques for this important class of problems and to demonstrate their effectiveness through computational experiments.

SOCP has a vast number of applications. Convex quadratic optimization problems (QPs) and quadratically constrained quadratic optimization problems (QCQPs) can be reformulated as SOCPs. Ben-Tal and Nemirovski [BN98] give SOCP formulations of robust LP. Ghaoui and Lebret [GL97] use SOCP to find robust solutions to least squares problems with uncertain data. Lobo et al. [Lob+98] formulate problems involving the Euclidean norm—particularly, the minimization of the sum of norms, the minimization of the maximum of norms, and the minimization of the sum of a fixed number of largest norms—as SOCP. Alizadeh and Goldfarb [AG03] give SOCP formulations of structural optimization, logarithmic Tchebychev approximation and finding smallest ball containing a given set of ellipsoids. More practical application areas of SOCP include filter design, antenna array design, truss design and grasping force optimization.

MISOCPs are used to model design of supply chain networks by Atamtürk, Berenguer, and Shen [ABS12], telecommunication networks by Fampa and Maculan [FM04], cardinality constrained portfolio optimization by Bertsimas and Shioda [BS09] and turbine balancing by Drewes [Dre09]. Fampa and Maculan [FM04] also give MISOCP formulation of Steiner tree problem. Aktürk, Atamtürk, and Gürel [AAG09] formulate machine job assignment problem as MISOCP.

The broad topic of this thesis is to investigate existing computational approaches to solve MISOCPs, as well as to propose and test new methods. Overall, we aim to develop a flexible framework within which a large variety of algorithmic approaches can be com-

pared. This study covers both theoretical and computational aspects. Implementational details are given and the proposed methods are implemented within the COIN-OR framework [Lou03], which is an existing and widely used open source optimization framework.

In addition to traditional mathematical (forward) optimization, also consider herein the related problem of inverse optimization. The goal of solving an optimization problem is to determine the member of a given feasible set (the solution) that minimizes the value of a given objective function, whereas the goal of inverse optimization is, given a solution (usually one that is a member of a given feasible set), to determine the missing parameters of the problem for which it is optimal.

1.2 Problem Classes

1.2.1 Conic Optimization

In a conic optimization problem, the feasible region is the intersection of an affine set and cartesian product of a given set of *cone*s. Some definitions related to cones are as follows.

Definition 1.3. (Cone) A subset $\mathcal{K}$ of a finite dimensional real space $\mathbb{R}^n$ is said to be a *cone* if an x in $\mathcal{K}$ imply λx being in $\mathcal{K}$ for a λ greater than 0.

Definition 1.4. (Convex Cone) A cone is said to be a *convex cone* if it is convex.

Definition 1.5. (Proper Cone) A cone is called *proper cone* if $\overline{\mathcal{K}} \cap -\overline{\mathcal{K}} = \{0\}$, where $\overline{\mathcal{K}}$ is closure of $\mathcal{K}$.

Definition 1.6. (Dual and Self-Dual Cone) The *dual cone* of a given cone $\mathcal{K}$ is denoted as $\mathcal{K}^*$ and defined as $\mathcal{K}^* := \{y \in \mathbb{R}^n \mid x^\top y \geq 0 \; \forall x \in \mathcal{K}\}$. $\mathcal{K}$ is *self-dual* if $\mathcal{K} = \mathcal{K}^*$.

Definition 1.7. (Linear Cone) *Linear cone* is a cone of the form $\{x \in \mathbb{R}^n \mid x \geq 0\}$.

Definition 1.8. (Semidefinite Cone) *Semidefinite cone* is a cone of the form $\{x \in \mathbb{R}^{n^2} \mid X \in \mathbb{R}^{n \times n}, x = vec(X), X = X^\top, X \succeq 0\}$, where x is vectorized form of matrix X in

column-major order (same as row-major since X is symmetric) and $vec(.)$ denotes this operation. $X \succeq 0$ denotes matrix X is positive semidefinite.

The canonical conic optimization problem is

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \in \mathbb{K}_{\text{CP}}, \end{aligned} \tag{CP}$$

where $\mathbb{K}_{\text{CP}}$ is cartesian product of cones, and A, c, b are same as in (LP). Conic optimization problems form a broad class and tractability varies widely depending on the cones in $\mathbb{K}_{\text{CP}}$. The most well-known conic optimization problem is (LP), in which $\mathbb{K}_{\text{CP}}$ is linear cone. Other well-studied cones from the optimization literature include second-order cones (SOCs) [NN94; Lob+98], positive semi-definite cones [NN94], exponential cones, power cones and copositive cones [Qui+98]. In this dissertation we study primarily second-order cones.

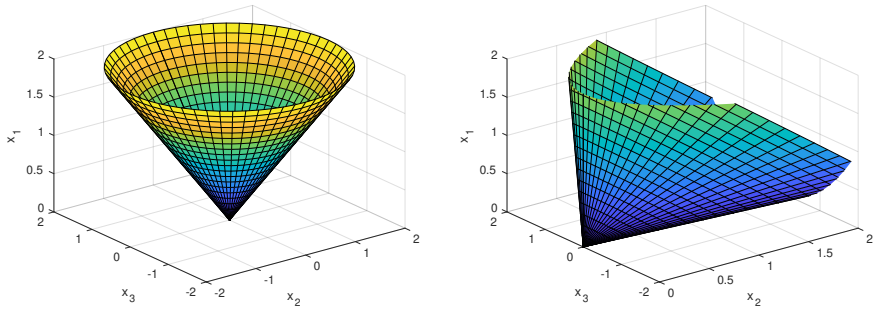
1.2.2 Second-Order Conic Optimization

Recall that the set of feasible solutions of (LP) are precisely the solutions to the system (LPb) of equations that lie in the non-negative orthant. In an SOCP, we replace the requirement that the solution be in the non-negative orthant with a requirement that the solution lies in the cartesian product of *Lorentz cones*.

Definition 1.9. (Lorentz Cone) A *Lorentz cone* is a cone of the form,

$$\mathbb{L}^k := \{x \in \mathbb{R}^k \mid x_1 \geq \|x_{2:k}\|\},$$

where k is the dimension of the cone and $x_{2:k}$ denotes the vector consisting all but the first component of x. When k is 1, the Lorentz cone is $\{x \in \mathbb{R} \mid x \geq 0\} = \mathbb{R}_+$.



(a) $\mathbb{L}^3$ (b) $\mathbb{L}^3_{\text{rot}}$

Figure 1.1: 3-dimensional Lorentz and rotated Lorentz cones ($\mathbb{L}^3$ and $\mathbb{L}^3_{\text{rot}}$)

A closely related cone is the *rotated Lorentz cone*.

Definition 1.10. (Rotated Lorentz Cone) A *rotated Lorentz cone* is a cone of the form

$$\mathbb{L}^k_{\text{rot}} := \{x \in \mathbb{R}^k \mid 2x_1x_2 \geq \|x_{3:k}\|^2, x_1 \geq 0, x_2 \geq 0\},$$

where k, dimension of the cone, is at least 3 and $x_{3:k}$ denotes the vector consisting all but the first two components of x.

Figure 1.1 displays 3-dimensional Lorentz and rotated Lorentz cones. There is an obvious relationship between Lorentz cones and rotated Lorentz cones that can be expressed in terms of the orthogonal transformation

$$T_k = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & I_{k-2} \end{bmatrix},$$

where k is at least 3 and I_{k-2} is identity matrix of rank $k-2$. In particular, we have

$$x \in \mathbb{L}^k \Leftrightarrow T_k x \in \mathbb{L}^k_{\text{rot}}.$$

Note that T_k is a symmetric involutory matrix (i.e., equal to its inverse) and is also orthogonal, which means that it represents an isometry (a linear transformation which preserves Euclidean distance). Hence, $\mathbb{L}^k_{\text{rot}}$ is a rotation of $\mathbb{L}^k$ and the two sets are isometric.

We are now ready to formally define an SOCP as precisely the problem of optimizing a linear objective function over a feasible region that is the intersection of a cartesian product of Lorentz cones and an affine space. In other words, an SOCP is any conic optimization problem of the form

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \in \mathbb{K}, \end{aligned} \tag{SOCP}$$

where $\mathbb{K}$ is cartesian product of k Lorentz cones of dimensions $n_1, \ldots, n_k$, i.e.,

$$\mathbb{K} := \mathbb{L}^{n_1} \times \mathbb{L}^{n_2} \times \cdots \times \mathbb{L}^{n_k}.$$

Throughout the dissertation, we denote the feasible set of (SOCP) by $\mathcal{S}_{\text{SOCP}}$. When the problem is given precisely in this form, it is referred to as *standard form* (or *primal form* for reasons we discuss later), but there are a number of alternative forms in which an SOCP can be specified, depending on exactly how the nonlinear constraints are integrated with the linear constraints. These lead to alternative formulations for the SOCP.

Since it is not always obvious whether a given optimization problem actually is an SOCP when it is given in one of the alternative forms, we would like to have a precise characterization of when a set can be equivalently represented in the form given above. [BN01a] build a theory of conic representability of general sets. They construct elementary sets that are conic representable (like the epigraph of a Euclidean norm) and give elementary operations that conserve conic representability (like intersection of two conic

representable sets). From there, they prove that any set that can be written using the given atomic sets and conic-property conserving operations is conic representable.

Herein, we consider the the most common forms in which conic sets can be represented and show how to identify which set are in fact conic. We have the following definition of conic representability.

Definition 1.11. (Conic Representability) A set $\mathcal{S}$ is called *second-order conic representable* if there exist $A \in \mathbb{Q}^{n\times m}$ and $b \in \mathbb{Q}^m$ such that

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid Ax = b, x \in \mathbb{K}\}.$$

In the remainder of this section, we introduce two alternative formulations and discuss (1) when a conic set can be expressed in one of these forms and (2) when a set expressed in one of these forms is a conic set. The first of these alternative forms is the *dual form*.

Definition 1.12. (Conic Dual Form) A set $\mathcal{S}$ is in *conic dual form* if it is defined as

$$\mathcal{S} = \left\{x \in \mathbb{R}^n \mid \|A^i x - b^i\| \leq {d^i}^\top x - \gamma^i, i = 1, \ldots, k\right\}, \tag{1.1}$$

where $A^i \in \mathbb{Q}^{m_i\times n_i}$ are rational matrices of rank m_i, $b^i \in \mathbb{Q}^{m_i}$, $d^i \in \mathbb{Q}^{n_i}$ and γ^i are scalars.

Sets in this form are said to be in conic dual form since they arise in formulating a certain dual of (SOCP). The following theorem and proof show that a set is a conic set if and only if it can be represented in conic dual form.

Theorem 1.2.1. *A set $\mathcal{S} \subseteq \mathbb{R}^n$ is a second-order conic representable set if and only if there exists $A^i \in \mathbb{R}^{m_i\times n_i}$, $b^i \in \mathbb{R}^{m_i}$, $d^i \in \mathbb{R}^{n_i}$ and $\gamma^i \in \mathbb{R}$ such that*

$$\mathcal{S} = \left\{x \in \mathbb{R}^n \mid \|A^i x - b^i\| \leq {d^i}^\top x - \gamma^i, i = 1, \ldots, k\right\}. \tag{1.2}$$

Proof. ($\Rightarrow$): Let $\mathcal{S}$ be a second-order conic representable set. There are two cases—the case in which $\mathcal{S} = \emptyset$ and the case $\mathcal{S} \neq \emptyset$. When $S = \emptyset$, setting A^i, b^i and d^i to be identically 0 and $\gamma^i = 1$ for $i = 1, \ldots, k$ results in the set on the right-hand side of (1.2) being the empty set, which proves the result when $\mathcal{S} = \emptyset$. For the second case, we assume $\mathcal{S} \neq \emptyset$. Since $\mathcal{S}$ is a second-order conic set, it can be represented in standard form. Hence, there exists matrix $\bar{A} \in \mathbb{Q}^{m \times n}$ of rank m and $\bar{b} \in \mathbb{Q}^m$ such that,

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid \bar{A}x = \bar{b}, x \in \bar{\mathbb{K}}\}.$$

where $\bar{\mathbb{K}}$ is a cartesian product of $\bar{k}$ Lorentz cones of dimensions $\bar{n}_1, \ldots, \bar{n}_{\bar{k}}$, as earlier. Let $H \in \mathbb{Q}^{n \times (n-m)}$ be a matrix such that its column vectors form an orthonormal basis for the null space of $\bar{A}$. Let $x^0 \in \mathbb{Q}^n$ be a solution to $\bar{A}x = \bar{b}$ (such x^0 exists since $\mathcal{S}$ is non-empty). Using H and x^0, $\mathcal{S}$ can be represented in dual form as

$$\mathcal{S} = \left\{x \in \mathbb{R}^n \mid x = Hw + x^0, Hw + x^0 \in \mathbb{K}, w \in \mathbb{R}^{n-m}\right\}.$$

Let $H^i \in \mathbb{Q}^{\bar{n}_i \times (\bar{n}_i - \bar{m}_i)}$ denote the rows of matrix H corresponding to Lorentz cone $\mathbb{L}^{\bar{n}_i}$ and H^i_1 be the first row and $H^i_{2:\bar{n}_i}$ be the rest of the rows of matrix H^i. Similarly, define x^{0i} to denote the portion of x^0 that corresponds to the Lorentz cone $\mathbb{L}^{\bar{n}_i}$. Then $\mathcal{S}$ can be written as

$$\begin{aligned}
\mathcal{S} &= \left\{x \in \mathbb{R}^n \mid x = Hw + x^0, \|H^i_{2:\bar{n}_i} w + x^{0i}_{2:\bar{n}_i}\| \leq H^i_1 w + x^{0i}_1, i = 1, \ldots, \bar{k},\ w \in \mathbb{R}^{n-m}\right\}, \\
&= \left\{x \in \mathbb{R}^n \mid \|x - Hw - x^0\| \leq 0, \right. \\
&\qquad \left. \|H^i_{2:\bar{n}_i} w + x^{0i}_{2:\bar{n}_i}\| \leq H^i_1 w + x^{0i}_1, i = 1, \ldots, \bar{k},\ w \in \mathbb{R}^{n-m}\right\}.
\end{aligned}$$

$\mathcal{S}$ is written using conic dual forms in the last expression. This ends the first part of the proof.

($\Leftarrow$): Let arbitrary $A^i \in \mathbb{R}^{m \times n}$, $b^i \in \mathbb{R}^m$, $d^i \in \mathbb{R}^n$ and $\gamma^i \in \mathbb{R}$ for $i \in 1, \ldots, k$ be

given arbitrarily so that $\mathcal{S}$ is a set in conic dual form. $\mathcal{S}$ can be represented using linear constraints and Lorentz cones as

$$\begin{aligned}\mathcal{S} &= \{x \in \mathbb{R}^n \mid \|A^i x - b^i\| \leq d^{i\top} x - \gamma^i, i = 1, \ldots, k\} \\ &= \{x \in \mathbb{R}^n \mid t^i = d^{i\top} x - \gamma^i, s^i = A^i x - b^i, (t^i, s^i) \in \mathbb{L}^{m_i+1}, i = 1, \ldots, k\}.\end{aligned}$$

This proves that any non-empty set in conic dual forms is conic representable.
Having proved both directions, the theorem is proven. □

A third way to represent conic sets is in terms of *quadrics*.

Definition 1.13. (Quadric) A *quadric* is a set of the form

$$\mathcal{Q} = \left\{ x \in \mathbb{R}^n \mid x^\top Q x + 2q^\top x + \rho \leq 0 \right\}, \tag{1.3}$$

where $Q \in \mathbb{Q}^{n\times n}$ is symmetric, $q \in \mathbb{Q}^n$, and $\rho \in \mathbb{Q}$.

The conic representability of quadrics is not quite as straightforward as the conic dual forms. Nevertheless, we describe below an explicit method to decide whether a set given by a quadratic constraint is conic representable.

It is well-know that $\mathcal{Q}$ is conic representable when Q is positive semidefinite. In this section, we show that $\mathcal{Q}$ may be conic representable even when Q has 1 negative eigenvalue. We begin by showing that second-order conic sets can be represented using quadrics.

Theorem 1.2.2. *If a set $\mathcal{S} \subseteq \mathbb{R}^n$ is second-order conic representable, then $\mathcal{S}$ can be represented using quadrics.*

Proof. Let a second-order conic representable set $\mathcal{S}$ be given. Then there exists $\bar{A} \in \mathbb{Q}^{m\times n}$ of rank m and $\bar{b} \in \mathbb{Q}^m$, and $\bar{\mathbb{K}}$, a cartesian product of $\bar{k}$ Lorentz cones of dimensions

$\bar{n}_1, \ldots, \bar{n}_{\bar{k}}$ as earlier, such that

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid \bar{A}x = \bar{b}, x \in \bar{\mathbb{K}}\}.$$

First, we represent x using using a basis for the null space of $\bar{A}$. Let $H^i \in \mathbb{Q}^{\bar{n}_i \times (\bar{n}_i - \bar{m}_i)}$ denote the rows of matrix H corresponding to Lorentz cone $\mathbb{L}^{\bar{n}_i}$ and H^i_1 be the first row and $H^i_{2:\bar{n}_i}$ be the rest of the rows of matrix H^i as before. Similarly, define x^{0i} to denote the portion of x^0 that corresponds to the Lorentz cone $\mathbb{L}^{\bar{n}_i}$ as before. We can write $\mathcal{S}$ as

$$\begin{aligned}\mathcal{S} =& \{x \in \mathbb{R}^n \mid \bar{A}x = \bar{b}, x \in \bar{\mathbb{K}}\} \\ =& \{x \in \mathbb{R}^n \mid x = Hw + x^0, x \in \bar{\mathbb{K}}, w \in \mathbb{R}^{n-m}\}.\end{aligned}$$

Let matrix J^i be an identity matrix of size $\bar{n}_i \times \bar{n}_i$ except with $J^i_{11} := -1$. We can re-write $\mathcal{S}$ using J^i as

$$\begin{aligned}\mathcal{S} =& \{x \in \mathbb{R}^n \mid (x - Hw - x^0)^\top (x - Hw - x^0) \leq 0, \\ & w^\top H^{i\top} J^i H^i w + 2(H^{i\top} J^i x^{0i})^\top w + x^{0i\top} J^i x^{0i} \leq 0, i = 1, \ldots, \bar{k}, w \in \mathbb{R}^{n-m}, \\ & (H^i w + x^{0i})_1 \geq 0, i = 1, \ldots, \bar{k}\}.\end{aligned}$$

When written in this form, the sets in the representation are in the form of quadrics and $\mathcal{S}$ can therefore be represented using quadrics. □

It is easy to see that not every quadric is convex, so it is clear that not all quadrics are conic representable. Theorem 1.2.2 proves conic representable sets can be represented with quadrics. The next theorem answers the question of under what conditions a quadric is a conic set.

Theorem 1.2.3. *Let quadric $\mathcal{Q}$ be given with a matrix Q with exactly 1 negative and $n-1$*

positive eigenvalues. $\mathcal{Q}$ is union of two symmetric sets in conic dual form if there exists $\overline{x} \in \mathbb{R}^n$ such that $Q\overline{x} = q$ and $\rho \geq q^\top \overline{x}$.

Proof. First, observe that Q can be decomposed as $M^\top JM$ for some matrix M where J is identity except $J_{11} = -1$. Using $\overline{x}$, quadric $\mathcal{Q}$ can be written as

$$\mathcal{Q} = \left\{ x \in \mathbb{R}^n \mid (x + \overline{x})^\top Q(x + \overline{x}) - q^\top \overline{x} + \rho \leq 0, \right\}$$

Using the mentioned decomposition of Q, quadric $\mathcal{Q}$ can be represented as

$$\begin{aligned}
\mathcal{Q} &= \left\{ x \in \mathbb{R}^n \mid (x + \overline{x})^\top M^\top JM(x + \overline{x}) - q^\top \overline{x} + \rho \leq 0 \right\}, \\
&= \left\{ x \in \mathbb{R}^n \mid (x + \overline{x})^\top (M_{2:n}^\top M_{2:n} - M_1^\top M_1)(x + \overline{x}) - q^\top \overline{x} + \rho \leq 0 \right\}, \\
&= \left\{ x \in \mathbb{R}^n \mid (x + \overline{x})^\top M_{2:n}^\top M_{2:n}(x + \overline{x}) \leq (M_1(x + \overline{x}))^2 - (\rho - q^\top \overline{x}) \right\}.
\end{aligned}$$

At this point we can write the right hand side as a multiplication of two terms using nonnegativity of $\rho - q^\top \overline{x}$ and represent the quadric $\mathcal{Q}$ as a union of $\mathcal{Q}^+$ and $\mathcal{Q}^-$ that are defined as follows

$$\begin{aligned}
\mathcal{Q}^+ = \Bigg\{ x \in \mathbb{R}^n \mid \Bigg(& \frac{1}{\sqrt{2}} \left(M_1(x + \overline{x}) - \sqrt{\rho - q^\top \overline{x}} \right), \\
& \frac{1}{\sqrt{2}} \left(M_1(x + \overline{x}) + \sqrt{\rho - q^\top \overline{x}} \right), \\
& M_{2:n}(x + \overline{x}) \Bigg) \in \mathbb{L}_{\text{rot}}^{n+1} \Bigg\}, \\
\mathcal{Q}^- = \Bigg\{ x \in \mathbb{R}^n \mid \Bigg(& -\frac{1}{\sqrt{2}} \left(M_1(x + \overline{x}) - \sqrt{\rho - q^\top \overline{x}} \right), \\
& -\frac{1}{\sqrt{2}} \left(M_1(x + \overline{x}) + \sqrt{\rho - q^\top \overline{x}} \right), \\
& M_{2:n}(x + \overline{x}) \Bigg) \in \mathbb{L}_{\text{rot}}^{n+1} \Bigg\}.
\end{aligned}$$

□

Both $\mathcal{Q}^+$ and $\mathcal{Q}^-$ are in conic dual form. Belotti et al. [Bel+13] report that $\mathcal{Q}$ is a cone (geometrical shape) when $\rho = q^\top \overline{x}$ and is hyperboloid of two sheets when $\rho > q^\top \overline{x}$. $\mathcal{Q}^+$ and $\mathcal{Q}^-$ each correspond to one sheet. We showed that each sheet is second-order conic representable. In an optimization problem, a quadric can be reduced to either $\mathcal{Q}^+$ or $\mathcal{Q}^-$ if it can be shown that its relaxation without the quadric intersects with only one of the $\mathcal{Q}^+$ or $\mathcal{Q}^-$.

1.2.3 Mixed Integer Linear Optimization

Optimization problems, some of whose variables are constrained to have integer values, are generally referred to as *discrete optimization* problems. The most well-studied class of discrete optimization problems is the MILPs. An MILP can be given in the following form,

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \in \mathbb{Z}_+^r \times \mathbb{R}_+^{n-r}. \end{aligned} \tag{MILP}$$

Note that this problem implicitly includes a conic constraint in the form of a requirement for the variables to be non-negative. Without this conic constraint, finding a feasible solution to MILP is to solve a system of (linear) Diophantine equations. For integer matrices, linear Diophantine equations can be solved by computing the Smith and Hermite normal form of the matrix A. Kannan and Bachem [KB79] give an algorithm that computes the Smith and Hermite normal form of an integer matrix in time bounded by a polynomial of the encoding of the input matrix. The result holds in our case even though the matrix A is not integral. The coefficients of the linear system are rational in our case. However, it is easy to see that it can be made integral by scaling each individual equation by the minimum common multiple of its denominators. The encoding length of the resulting system will be reasonable. In essence, we can conclude that the only difference between problem

of finding a feasible solution for MILP and linear Diophantine equations, is the presence of the nonnegativity constraints in the former. The latter can be solved in polynomial time and the presence of nonnegativity constraints makes the former NP-hard.

MILPs constitute a very well-studied, widely known problem class with which a vast amount of literature is associated. For an introduction to basic theory, see, e.g., Gomory [Gom60], Nemhauser and Wolsey [NW88], Cornuéjols [Cor08], and Linderoth and Ralphs [LR05].

1.2.4 Mixed Integer Second-Order Cone Optimization

As with the generalization from linear optimization to conic optimization, problem (MILP) can be generalized by replacing the requirement that solutions be in the non-negative orthant with a more general requirement the solution lie in a given cone. We focus here on the specific case in which the cone is the cartesian product of Lorentz cones, introduced earlier as the MISOCP. The MISOCP is then

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \in \mathbb{K} \\ & x_i \in \mathbb{Z} \qquad i \in I, \end{aligned} \tag{MISOCP}$$

where I is the index set of integer variables. We use C as the index set of continuous variables and $\mathcal{S}_{\text{MISOCP}}$ as the feasible set of (MISOCP). A_I is used to denote the matrix composed of columns of A that corresponds to indices in set I. A_C is defined similarly. MISOCP can be thought of as a generalization of MILP. The *continuous relaxation* of an MISOCP is (SOCP), introduced earlier. MISOCP is used to model many practical applications. Chapter 3 discusses the application areas and solution methods for MISOCP.

1.2.5 Inverse Optimization

Another problem studied in this thesis is the so-called *inverse MILP problem*. When solving an optimization problem, the goal is to determine the member of a given feasible set (the *solution*) that minimizes the value of a given objective function. In inverse optimization, the goal is to determine the values of uncertain problem parameters (objective value, constraints, etc.) such that a given point is optimal with those values as inputs.

We provide a conic formulation for a particular class of inverse MILP in which the goal is to determine the objective coefficient vectors. Moreover, we provide evidence that places this problem into the tightest complexity class possible without resolving the well-known question of whether $\mathsf{P} = \mathsf{NP}$ (see Section 1.3 for discussion). For a feasible set $\mathcal{S}$, the inverse optimization problem can be given as

$$\begin{aligned} \min \quad & \|c - d\| \\ \text{s.t.} \quad & d^\top x^0 \leq d^\top x \qquad \forall x \in \mathcal{P}, \end{aligned} \tag{Inverse}$$

where $\|.\|$ is a given norm. In this formulation d, is the variable vector of the problem, c and x^0 are parameters. A feasible solution for the problem is a vector which minimizes x^0 over $\mathcal{S}$. The optimal solution is the feasible d that is closest to estimate c.

Note that the problem is always feasible since $d \leftarrow 0$ is feasible for the problem. This gives an upper bound $\|c\|$ on the objective value. Moreover it is easy to see that objective value is bounded from below by 0.

Ahuja and Orlin [AO01] study the case where $\mathcal{S}$ is a polyhedral and l_1 or l_∞ norm is used as a distance measure. They prove that the inverse problem can be solved in polynomial time when the forward problem can be solved in polynomial time. To obtain this result they rely on the seminal work on separation–optimization of Grötschel, Lovász, and Schrijver [GLS93].

Grötschel, Lovász, and Schrijver [GLS93] build a detailed theory of the relationship of

the separation and optimization problems associated with given convex sets. They show that the separation problem associated with a given convex set is equivalent (in the sense they define) to optimization over the same set and that efficient algorithms for each can be implemented, given an oracle for the other. Using this equivalence, they show that some of the combinatorial problems for which a compact formulation is unknown (but solving the separation problem over their feasible set is easy) can be solved in polynomial time.

Chapter 5 contains a study of some theoretical properties of the inverse MILP, including its computational complexity. A result similar to Ahuja and Orlin [AO01] can be obtained for the case of MILP by applying the results from Grötschel, Lovász, and Schrijver [GLS93] cited above. We show that for both the l_1 and l_∞ norms, given an oracle for optimizing over $\mathcal{P}$, the feasible region of an MILP, the associated inverse problem, can be solved in polynomial time. This immediately results in a classification of the decision version of the inverse optimization as being in the class Δ_2^{P} (see Section 1.3 for discussion). Chapter 5 goes beyond this result and proves that the decision version of inverse MILP is coNP–complete for any given norm. This is the tightest complexity class that can be achieved for inverse MILP without resolving the well-known question of whether P equals NP. Discussion of these complexity classes is given next.

1.3 Computability and Computational Complexity

1.3.1 Overview

The modern theory of computation can be considered to have its origin with the statement of Hilbert's 10^{th} problem, which was to find an algorithm to decide whether a given polynomial equation with integer coefficients (Diophantine equation) has an integer solution. After many years of consideration, it was finally shown that no such algorithm exists. The result was the combined work of Yuri **M**atiyasevich, Julia **R**obinson, Martin

Davis and Hilary **P**utnam and is known as the MRDP theorem. Note that the problem of solving a system of polynomial Diophantine equations is a special case of general MINLP, so this shows that general MINLP is also undecidable, i.e., there is no algorithm (a Turing machine or similar, as described next) that will terminate for all finite problem instances in finite time.

Another problem formulated by Hilbert was the "Entscheidungsproblem" (German for "decision problem"). The problem is to find an algorithm that decides whether a given statement (in first order logic) is provable from the given axioms and can be considered one historical reason that complexity theory is defined in terms of classes of decision problems. Alonzo Church and Alan Turing independently prove that there is no solution for this problem. In their proofs, Alonzo Church introduced lambda calculus and Turing proposed a conceptual model akin to a modern computer in order to mathematically formalize the computation. The model of the computational device introduced by Turing has since become known as a *Turing machine* and many alternatives were subsequently developed. One alternative was the *Unlimited Register Machines* (URM) proposed by Shepherdson and Sturgis [SS63]. It is still widely used in the theory of computability since it is easier to work with than the Turing machine model. For details on the URM and its use in the theory of computability, see Cutland [Cut80].

Despite the many alternative models of computation proposed, most have turned out in the end to be equivalent. For example, Turing [Tur37] proved that a function is λ–computable if and only if it is Turing computable. To date, all models of computation have turned out to result in the same set of computable functions and to obey an analogue of the well-known Church–Turing thesis, which states that a function is computable if its values can be found by a mechanical process. Bernstein and Vazirani [BV97] generalized the Church–Turing thesis with efficiency considerations. The Extended Church–Turing thesis states that all *reasonable* models of computation result in the same set of *polynomially*

computable functions. Reasonable models can be interpreted as those modeling machines bounded by the laws of physics and no physical model of computation (including quantum models of computation or possession of quantum computers) that defies the Extended Church-Turing thesis has yet been perceived.

1.3.2 Computational Complexity

The studies we've mentioned up to this point concern computability of mathematical functions. These efforts can be considered under the umbrella of the *theory of computability*. Computational complexity theory, on the other hand, concerns the measurement of how difficult a given problem is to solve in terms of the resources that are required. The two resources most typically considered are time and space (memory, storage). From computational complexity theory perspective, time has been considered a more valuable resource than space. PSPACE is the class of all decision problems that can be solved using an amount of space that is polynomial as a function of the size of the input. We know that PSPACE contains all problems in all classes of the so-called *polynomial hierarchy* (see formal description below). On the other hand, it is not known whether P, the class of problems that can be solved in *time* polynomial in the size of the input, contains all problems in the polynomial hierarchy, but it is considered extremely unlikely. This is the aforementioned problem of deciding whether $\mathsf{P} = \mathsf{NP}$.

For the historical reasons mentioned earlier, the theory of computational complexity is generally focused on decision problems, i.e., the problem of proving a given statement is TRUE or FALSE. Such a problem can be seen as the evaluation of a function that take as input a string of characters from a given *alphabet* and gives TRUE or FALSE as output. A *language* is a set of strings that return the value TRUE with respect to a particular question. A language can thus be thought of as a "problem," while a particular input string can be thought of as an "instance" of that problem. Given string s as an input

instance and a language L, a procedure that outputs TRUE whenever $s \in L$ and FALSE otherwise is said to *recognize* language L. In its original conception, a Turing machine is the implementation of such a procedure using Turing's original computation model. A Turing machine can thus be thought of as an "algorithm" to solve a particular class of problems.

The (time) complexity of an algorithm is determined by the *running time function* that maps the size of the input (the length of the input string) to the worst-case time (number of steps) required by the associated Turing machine across all possible inputs of the given size. The (time) complexity of a problem (language), on the other hand, is the running time function of the best known algorithm. Note that the for almost all problems, this "best" running time is an upper bound, as it has been impossible to determine whether the best known algorithm is the best possible.

Using the time complexity, problems can be placed into the aforementioned *polynomial hierarchy*, which serves to divide problems into classes according to the efficiency of their best known algorithms. The classes can be equivalently defined in multiple ways, but the most straightforward is to define them recursively using the concept of an *oracle*. The class P is the set of problems whose running time function is a polynomial function of the input size. NP is the set of problems that can be solved in polynomial time on a *non-deterministic Turing machine*, which can be roughly described as a Turing machine with an infinite number of processors capable of exploring any number of execution paths of the algorithm simultaneously. It is not difficult to see that a problem is in NP if and only if for a given instance in the language, there exists a polynomial-sized string that can be used to verify in polynomial time that the instance is in the language.

Papadimitriou and Yannakakis [PY82] define the class D^{P} to be the class of languages that are the intersection of two languages, the first of which is in NP and the second of which is in coNP. D^{P} is a broader class that includes NP and coNP.

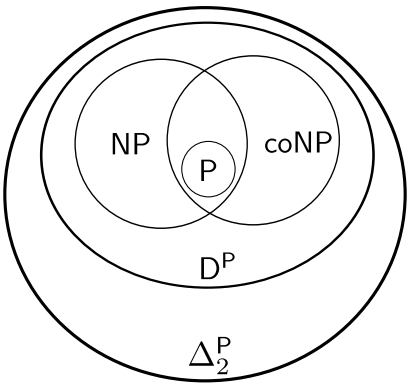


Figure 1.2: Complexity classes Δ_2^{P}, D^{P}, NP, coNP and P, assuming $\mathsf{P} \neq \mathsf{NP}$

The next class in the hierarchy is Δ_2^{P}, which is the set of problems that can be solved in polynomial time in a Turing machine given an oracle capable of solving any problem in NP in a single time step. It is a broader class that includes D^{P}. Every problem in NP, coNP or D^{P} is also in Δ_2^{P}. Figure 1.2 illustrates class Δ_2^{P} relative to D^{P}, NP, coNP and P, assuming $\mathsf{P} \neq \mathsf{NP}$.

Σ_2^{P} is the set of problems that can be solved in polynomial time on a non-deterministic Turing machine given the same NP oracle. Similarly Σ_k^{P} is the set of problems that can be solved in polynomial time on a non-deterministic Turing machine given a $\Sigma_{k-1}^{\mathsf{P}}$ oracle. Note that Σ_k^{P} contains $\Sigma_{k-1}^{\mathsf{P}}$, Σ_2^{P} contains Δ_2^{P}. If $\mathsf{P} = \mathsf{NP}$ then the whole polynomial hierarchy is same as P, which is called the collapse of the polynomial hierarchy to its first level [Pap94]. A problem in a specific class is called *complete* for the class if all other problems in the class can be reduced to it.

1.3.3 Complexity of Optimization Problems

Historically, the complexity of optimization problems was studied by converting them to an equivalent decision problem. Optimization problems are converted to decision problems by asking whether a feasible solution exists that leads to an objective value better than a given threshold.

The invention of the Ellipsoid Method placed the decision version of LP (dLP) into the class P of decision problems solvable in polynomial time. It is easy to see decision version of MILP (dMILP) is in the class NP. Reduction from SAT proves that dMILP is complete for class NP. Existence of IPM places decision version of SOCP (dSOCP) and semidefinite optimization problems (dSDP) into class P. Similarly, the decision version of MISOCP (dMISOCP) is NP–complete.

Another decision problem is asking whether the optimal solution of a given optimization problem has objective function value γ for a given γ. It is easy to see that this problem is polynomially solvable for optimization problems that are polynomially solvable, e.g., LP, SOCP, SDP, etc. Papadimitriou and Yannakakis [PY82] show that this problem for MILP (see Section 5.3.1 for formal definition) is D^P–complete. This means, exact optimal value problem is not in NP or coNP unless P is NP.

A closely related problem is to decide whether the optimal solution is unique. Papadimitriou [Pap84] shows that deciding this for traveling salesman problem is Δ_2^{P}–complete. This indicates that this problem is not in class NP or coNP unless P is NP.

Separation decision problem is deciding whether a given hyperplane separates a given point from a given set. It is easy to see that this problem is in coNP for MILP case, i.e., when separating from MILP feasible sets. Separation problem for MILPs (sMILP) is in the same level of polynomial hierarchy as dMILP. In Chapter 5 we show that this also holds for the inverse MILP. Decision version of inverse MILP is in the same level of polynomial hierarchy with dMILP and sMILP.

1.4 Basic Algorithms

1.4.1 Branch-and-Bound Algorithm

To date, almost all practical algorithms that have been suggested in the literature for solving global optimization problems are based on the well-known *branch-and-bound* algorithm first introduced by Land and Doig [LD60]. The branch-and-bound algorithm recursively partitions the feasible region of the original optimization problem to obtain smaller *subproblems*, which are solved recursively. The partitioning procedure also allows for the computation of a bound on the value of an optimal solution to the original problem (the *optimal value*) by solving a *relaxation* of each of the subproblems. In particular solving a relaxation of each subproblems yields a lower bound on the optimal value of the subproblem and a bound on the optimal value of the original problem is given by the minimum of the lower bounds for the subproblems. An upper bound for the original problem, on the other hand, is given by the value of the best feasible solution.

More formally, the subproblems are created by imposing so-called *valid disjunctions* on the original problem.

Definition 1.14. (Valid Disjunction) A valid disjunction for (OPT) is a disjoint collection of finitely many sets $\mathcal{F}_i \subseteq \mathbb{R}^n,\ i = 1, \ldots, k$ such that

$$\mathcal{F} \subseteq \bigcup_{i=1,\ldots,k} \mathcal{F}_i.$$

Such a disjunction is *violated* by a given $\hat{x} \in \mathbb{R}^n$ if

$$\hat{x} \notin \bigcup_{i=1,\ldots,k} \mathcal{F}_i.$$

Every disjunction can be associated with a *disjunctive constraint* which require that solutions be contained in one of the sets $\mathcal{F}_i$. When a disjunctive constraint is added to the

optimization problem (OPT), the resulting optimization problem is known as a *disjunctive program* [Bal79; Bal85]. The most straightforward way of solving such a disjunctive program is to associate a subproblem with each term $\mathcal{F}_i$ of the disjunction that requires solutions to that subproblems to be in set $\mathcal{F}_i$. If the set $\mathcal{F}_i$ is a polyhedron (the set of solution to a system of linear inequalities), then the requirement that solution be in $\mathcal{F}_i$ does not change the form of the problem with respect to the classification in Section 1.2.

Algorithm 1 specifies a generic version of the branch-and-bound algorithm. The following algorithm does not assume a specific problem and describes the algorithm in its most generic form. The details of how the algorithm is implemented in the particular case of MISOCP will be given in Chapter 3.

Algorithm 1 A Generic Branch-and-Bound Algorithm

1: $\mathcal{Q} \leftarrow \{0\}$.
2: $LB \leftarrow \infty$.
3: $UB \leftarrow -\infty$.
4: **while** $LB < UB$ **do**
5: $\quad i \leftarrow \text{argmax}_{j \in \mathcal{Q}} p_j$.
6: $\quad$ **Bound** subproblem i to obtain $LB(i)$ and $UB(i)$.
7: $\quad$ Set $UB \leftarrow \min\{UB(i), UB\}$.
8: $\quad$ **if** $LB(i) < UB$ **then**
9: $\quad\quad$ **Branch** to create children subproblems $i_1, \ldots, i_k$ of subproblem i with priorities $p_{i_1}, \ldots, p_{i_k}$; initial lower bounds $LB(i_1), \ldots, LB(i_k)$; and initial upper bounds $UB(i_1), \ldots UB(i_k)$, by partitioning the feasible region of subproblem i.
10: $\quad\quad \mathcal{Q} \leftarrow \{i_1, \ldots, i_k\}$.
11: $\quad\quad$ Set $LB \leftarrow \min_{i \in \mathcal{Q}} LB(i)$.
12: $\quad$ **end if**
13: **end while**

Set $\mathcal{Q}$ contains the indices corresponding to the set of subproblems that are currently waiting to be bounded, with index 0 corresponding to the original problem. LB and UB denote lower and upper bounds, respectively, on the optimal value of the original problem. These are updated throughout the algorithm until they become equal (the termination criterion). In the bounding step (line 6), we compute upper and lower bounds

for subproblem i, denoted by $LB(i)$ and $UB(i)$, respectively (see Section 1.4.1).

The details of exactly how the various components are implemented and the entire process is managed are crucial to the overall effectiveness of the approach. Although the procedure seems straightforward when presented in this form, it is well-known that a naive implementation of this procedure will not work well in practice for most applications.

The branch-and-bound algorithm is inherently recursive in nature, since the feasible region of a given subproblem is initially obtained by a partitioning of the *parent* subproblem and may itself be again partitioned at a later step to obtain *children*. Thus, the evolution of the algorithm can be viewed as exploring a so-called *search tree* that visualizes the recursion process. Ralphs, Ladanyi, and Saltzman [RLS04] and Xu et al. [Xu+05] suggest the branch-and-bound algorithm can be considered to be a specialized *tree search algorithm*. In employing this metaphor, each subproblem generated is associated with a *node* in the search tree. A typical tree search algorithm processes nodes one by one. In a branch-and-bound algorithm, processing a node amounts to determining bounds on the optimal value of the associated subproblem. The process of obtaining new subproblems from existing ones by partitioning (typically by imposing a valid disjunction) is called *branching* (line 9).

The tree itself is described through parent–child relationships, beginning with the root node. Except for the root node, each subproblem has a *parent* and zero or more *children*. Subproblems with no children are called *terminal* or *leaf* nodes. The *depth* of a node in the tree is the length of the path to the root node in terms of the parent-child relationships. From this point of view, the lower bound LB is the minimum of the lower bounds of all leaf nodes of the current search tree. Upper bound UB is updated when a new solution to the problem is found, as we describe below.

In the worst case, the branch-and-bound algorithm consists roughly of a complete enumeration of all possible subproblems. The key to avoiding this is to generate enough

information during the search in order to prove that the feasible regions of certain nodes (and all their potential descendants) cannot contain improved solutions. Such nodes are not processed in order to save computation. This elimination of unpromising nodes is called *fathoming* and takes place on line 8 (in the case where $LB(i) \geq UB$). Moreover, the algorithm controls the order in which the nodes are processed based on priorities that are modified dynamically as the search progresses. The decision of which node to process next is answered based on this prioritization. This kind of prioritization and decisions are called *search strategy*.

Even though we stated that branch and bound performs a complete enumeration in the worst case, this seldom occurs in practice when the algorithm is implemented in a sophisticated way. The success of the branch and bound compared to the complete enumeration is due to fathoming and search strategy. These two factors significantly reduce the number of nodes processed and save computation time. In the following sections, we briefly describe in more detail how each of the components of the algorithm just described are implemented in practice.

Bounding

As mentioned earlier, in the most straightforward variant of branch and bound, bounding is accomplished by solving a relaxation of the given subproblem. The optimal value of this relaxation is always a lower bound on the optimal value of the subproblem (if the relaxation is infeasible, then so is the the subproblem). Furthermore, if the solution to the relaxation is feasible for the original problem, then the solution is optimal for the subproblem and the optimal value is also an upper bound in this case.

For efficiency reasons, it is important that the relaxation solved to obtain the bound be more tractable than the subproblem itself. Thus, the relaxation is often obtained by approximating the feasible region by a convex set. In the case of an MILP (and MISOCP),

such relaxation can most easily be obtained by relaxing integrality constraints.

Branching

Branching is the process of creating new subproblems from an existing one. As described earlier, this is done by imposing a *valid disjunction*. Such a disjunction partitions the feasible region of the parent subproblem. Ideally, the disjunction is chosen such that the solution to the relaxation of the current subproblem is not is not contained in the feasible regions of those of any of the created subproblems. Such valid disjunctions are called *violated valid disjunctions*. The procedure by which the disjunctions are chosen (the problem of choosing a disjunction is itself an optimization problem) is called *branching strategy*.

In the case of MILP, when the relaxation is obtained by relaxing integrality constraints, at least one integer variable must have fractional value in the solution to the relaxation. For example, if the solution to the relaxation is $\hat{x}$, then we have $\lfloor \hat{x}_i \rfloor < \hat{x}_i < \lceil \hat{x}_i \rceil$ for some $i \in I$. The solution therefore violates the valid disjunction

$$\mathcal{F}_1 = \{x \in \mathbb{R}^n : \hat{x}_i \leq \lfloor \hat{x}_i \rfloor\} \;\; OR \;\; \mathcal{F}_2 = \{x \in \mathbb{R}^n : \hat{x}_i \geq \lceil \hat{x}_i \rceil\}$$

In this case, the branching strategy is known as *variable branching*.

There are many tradeoffs and concerns involved in the branching process. To begin with the process of determining the disjunction should itself be tractable. Bounding of newly created subproblems should also be tractable, which means that the sets associated with terms of the disjunction should themselves typically be convex. Moreover, a good branch strategy should allow the employment of warm starting in solving the new subproblems created, using the information available from bounding process of the parent subproblem.

Branching strategy should decide the disjunction to be used among possible alterna-

tives. The goal is to choose disjunctions that will lead the optimal solution fastest. How to achieve this goal with the information on hand is not clear. Typically strategies use bounding information of the parent and new subproblems at this point. One idea is to use disjunctions that will lead to new subproblems such that lower bound is improved most. Computing lower bound improvement for all disjunction candidates might be computationally expensive. Approximate methods are preferred over computing bound improvements exactly. These phenomena are well studied in MILP case. *Strong branching* introduced by Applegate et al. [App+95] for MILP case considers possible variable disjunctions and computes bound improvement for all. *Pseudocost branching* first used by Bénichou et al. [Bén+71], estimates, instead of computing exactly as in strong branching case, the bound improvement for all possible disjunctions to save computation. *Reliability branching* given by Achterberg [Ach07] combines pseudocost and strong branching. It initializes the pseudocosts of variables using strong branching and updates the pseudocost estimates that are unreliable. Achterberg [Ach07] reports that reliability branching is the best performer in terms of CPU time for the MILP case.

Search Strategy

In Algorithm 1, at the beginning of each iteration of the while loop, a subproblem should be selected for processing. In another words, which node should be searched for a solution next, should be decided. The way this decision is made is called *search strategy*.

Most widely known tree search strategies are *breadth-first* and *depth-first* search. Breadth-first strategy searches all the children of the current node first, where depth-first searches the first found children and then moves to its children. Note that branch-and-bound tree is dynamically created during the execution. In a tree hanged from the root, breadth-first search expands the tree horizontally, where depth-first search creates deeper nodes and expands the tree vertically. Breath-first search creates more subproblems com-

pared to depth-first search. In a branch-and-bound algorithm this means exponentially many (in terms of the depth of the tree) candidate subproblems is created and this will increase memory requirements. On the contrary, in depth first, the number of subproblems increases linearly in terms of the depth of the tree. Hence memory requirements of depth-first search is exponentially less than the breadth first.

Note that the deeper a node is in the tree, the more constraints will have been accumulated from the imposition of various disjunctions at ancestor nodes. When these constraint tend to encourage feasibility (such as when they constraint a certain variables value to a single integer), it becomes more likely that feasible solutions to the original problem will be obtained by solving the relaxations to subproblems at nodes deeper in the tree. On the other hand, the quality of such solutions is not guaranteed and highly depends on the instance and in what part of the search tree they are obtained. As new solutions improve the upper bound of the problem, it can be ascertained that depth-first search aims to improve the upper bound. In breadth-first search, on the other hand, the depth of the tree increases more slowly. This indicates that finding solutions is less likely compared to depth-first search.

Another search strategy is *best-first* search which uses the objective values to decide the next node to search. Note that both breadth-first and depth-first search strategies employ parent–child relations. The best first search strategy, on the other hand, uses the *quality* of the corresponding subproblems to decide which node to search next, it selects the node with the best quality. One of the most common quality measure is the objective value estimate. When a new subproblem is created its objective value estimate is set to parents optimal objective value or its estimate. Achterberg [Ach07] proves that the best-first strategy minimizes the size of the tree when branching is fixed and the right node is picked from the ones with the same quality.

Hybrid-search strategy carries depth-first search until the node quality of the both

siblings is worse than the best available by a certain threshold [Xu07]. Hybrid-search strategy stops diving at this point and selects the node with the best quality next. Hybrid search is a hybrid of depth-first and best-first search strategies.

Heuristics

Heuristics are auxiliary methods used as alternatives to solving the subproblem relaxation for the purpose of finding solutions to a given optimization problem. Unlike the branch-and-bound method itself, heuristics typically use less resources but do not guarantee success. They can be considered an attempt to find a solution using minimal effort by exploiting the information on hand.

Heuristics can be executed in tandem with the branch-and-bound algorithm. If they use primal solution information obtained when solving the relaxations of the subproblems are called *primal heuristics*. Such methods play a crucial role in all available state-of-the-art optimization solvers, both open source and commercial. The literature is abundant with heuristic methods, e.g., Bertacco, Fischetti, and Lodi [BFL07], Danna, Rothberg, and Pape [DRP05], and Lodi, Allemand, and Liebling [LAL99].

1.4.2 Cutting-Plane Algorithm

The classical cutting-plane method is an iterative approach to solving optimization problems with a convex feasible region. The basic approach is to approximate the feasible region from the outside with a polyhedron. This defines a relaxation of the problem that is first solved to obtain a tentative solution. If the solution is infeasible, a hyperplane separating the solution from the feasible region is generated (this is the *separation problem* and an inequality corresponding to this separating hyperplane is then added to the description of the approximating polyhedron. Under certain conditions, such a cutting-plane method can be guaranteed to converge and to solve the original optimization problem. To state

the method more formally, we need some definitions.

Definition 1.15. (Valid Inequality) With respect to a convex set $\mathcal{C} \subseteq \mathbb{R}^n$, a *valid inequality* is a pair (a, α) such that $a \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$ and

$$\mathcal{C} \subseteq \{x \in \mathbb{R}^n \mid a^\top x \leq \alpha\}.$$

An inequality valid for $\mathcal{C}$ is *violated* by $\hat{x} \in \mathbb{R}^n$ if $\hat{x} \notin \{x \in \mathbb{R}^n \mid a^\top x \leq \alpha\}$.

In the literature, a violated valid inequality is also known as a *cutting plane* or simply a *cut*, which is the origin of the name. The *separation problem* is that of generating a violated valid inequality whenever the solution to the current approximation is not feasible. We define it formally as follows.

Definition 1.16. (Separation Problem) With respect to a given convex set $\mathcal{C} \subseteq \mathbb{R}^n$ and a given $\hat{x} \in \mathbb{R}^n$, the *separation problem* is that of determining whether $\hat{x} \in \mathcal{C}$ and if not, determining an inequality (a, α) valid for $\mathcal{C}$ but violated by $\hat{x}$.

With these two definitions, we can state a generic version of the cutting plane algorithm algorithm for solving

$$\min_{x \in \mathcal{C}} c^\top x,$$

where $\mathcal{C}$ is a given convex set and $c \in \mathbb{Q}^n$, as follows. Algorithm 2 relaxes the original problem, finds an optimal solution for the relaxed problem, solves the separation problem, and refines the relaxation by adding the cuts generated. The algorithm stops when a feasible solution is found (in practice, an ϵ-feasible solution are acceptable).

It should already be clear from this brief discussion that the separation problem with respect to a given convex set is closely related to that of optimize a linear function over the same set. In fact, Grötschel, Lovász, and Schrijver [GLS93] showed that optimization over a convex set can be solved in polynomial time given an oracle for the separation problem

Algorithm 2 A Generic Cutting-Plane Algorithm

Let $\mathcal{P}^0$ be a polyhedron such that $\mathcal{P}^0 \supseteq \mathcal{C}$.
$\hat{x}^0 \leftarrow \min_{x \in \mathcal{P}^0} c^\top x$
$i \leftarrow 0$
while $\hat{x}^i \notin \mathcal{C}$ **do**
    Solve the separation problem with respect to $\hat{x}^i$ and $\mathcal{C}$.
    if $\hat{x}^i \in \mathcal{C}$ **then**
        $\hat{x}^0$ is optimal
    else
        Let (a^i, α^i) be an inequality valid for $\mathcal{C}$, but violated by $\hat{x}^i$.
        Let $\mathcal{P}^{i+1} \leftarrow \mathcal{P}^i \cap \{x \in \mathbb{R}^n : a^{i\top} x \leq \alpha^i\}$
    end if
end while

(and vice versa).

Since solving (MILP) is equivalent to optimizing over the convex hull of feasible solution, which is a convex set, the cutting plane algorithm can be used, in principle, for solving MILPs. In practice, however, the cutting-plane method is used in a modified form with a branch-and-bound algorithm as a method for obtaining improved bounds on the optimal values of the subproblems that arise. There are many methods for generating valid inequalities for the convex hull of feasible solutions to an MILP in the literature. Some of these are Gomory [Gom58], mixed integer rounding [NW90], clique [Sav94], flow cover [PRW85; RW86], knapsack [Bal75; HJP75; Pad75], odd hole [Fis94; CG96], and probing cuts [Sav94]. These cuts might not be effective when used in a pure cutting-plane approach but they are extremely effective when used within a branch-and-bound approach.

The question of how to generate inequalities valid for the feasible region of MISOCP is a question under active research. Conic mixed integer rounding, conic Gomory and disjunctive conic cuts are among the ones proposed. These methods are reviewed in Section 3.2.

1.4.3 Branch-and-Cut Algorithm

In the case of MILPs, it has been shown through experimentation that cutting-plane methods on their own are not effective. In practice, cutting-plane methods can be combined with the branch-and-bound algorithm to produce an algorithmic approach that has proven to be very effective overall in practice [Bal+96; Bix+00; Cor07]. When cutting-plane methods are used within branch-and-bound framework, the resulting algorithm is called *branch and cut*.

In branch-and-cut algorithms, a truncated version of the cutting-plane method can be used to dynamically improve the relaxation of the subproblem solved at nodes in the branch-and-bound tree [Mar+02; BCC96; FM05a]. In this approach, valid inequalities are generated iteratively, as in the standard cutting-plane method, but the method is typically terminated prior to full convergence once the rate of progress in improving the bound slows. Once this occurs, the branching operation is invoked to create new subproblems.

The question of how to determine precisely when the cutting-plane phase should be terminated and the branching method invoked is not particularly well-studied in the literature, although some answer to this question is required as part of the implementation of this approach. Generally speaking, cutting planes are added more aggressively when generating bounds at the root node, since the improvement of the formulation at this early stage of the algorithm pays bigger dividends in general than the improvement of the formulation for subproblems associated with nodes deeper in the tree. We propose a branch-and-cut algorithm to solve MISOCP in Chapter 3 and discuss details of the implementational issues in this context.

1.4.4 Global Optimization Algorithms

Global optimization is a broad field that involves the development of algorithms for finding the so-called "global optimal solution" of mathematical optimization problems of the

form (OPT). The word "global" is meant to be in contrast with the "locally optimal" solutions which are the result of a different class of algorithms for nonlinear optimization that attempt to find solutions satisfying necessary conditions for optimality but which are not necessarily "global optimal."

The fact that global optimization addresses a very broad class of problems without many assumptions makes global optimization problems difficult to solve in general. Only the very minimal assumptions necessary to ensure the convergence of given algorithms are typically made. Some of the most common assumptions are continuity, Lipschitz continuity, and differentiability of the functions used to describe the problem.

A variety of strategies are used to obtain global solutions to this broad class of problems. Pintér [Pin13] lists grid search, random search, tunneling, relaxation strategies, and branch-and-bound methods as solution methods to global optimization problems. Grid search looks for local solutions on uniformly distributed points that form a grid in the relevant space. Random search samples the grid points randomly rather then selecting them uniformly.

The global optimization methods most relevant to this study are the relaxation-based strategies, such as the branch-and-bound and cutting-plane methods described earlier. In relaxation strategies, constraints of the original problem are relaxed to obtain problems that are easier to solve. By solving a series these relaxations and improving the relaxation successively, a globally optimal solution is eventually produced.

As described earlier, in branch-and-bound method, some of the original constraints are relaxed to obtain an easier subproblem (root node). Disjunctions are imposed in the root subproblem that partition the feasible region of the original problem, ensuring that each feasible solution to the original problem is feasible to one of the resulting subproblems, as well as ensuring that the solution to the relaxation of the parent subproblem is not feasible to the relaxations of either of the resulting subproblems (assuming the disjunction has two

terms). In Section 3.1.4, we discuss both commercial and open source global optimization solvers that can be used to solve MISOCPs.

1.4.5 Related Methodologies

Outer Approximation

Outer approximation is a term used to describe any number of strategies for either solution of or computation of bounds for a given optimization problem [DG86; FL94; Ben98]. In an outer-approximation method, the idea is to construct a set enclosing the original feasible region of an optimization problem that is easier to optimize over than the original set, e.g., it is convex. This approximating set defines a relaxation of the original one. If the optimal value of the original problem and that of relaxation over the outer-approximating set coincides, then the optimal solution is found. This general approach is closely related to the cutting-plane method, which generates a sequence of polyhedral approximations. Various methods in the literature are referred to as outer-approximation methods.

Polyhedral Approximation. Polyhedral approximation is the case when the outer-approximation set is polyhedral, such as in a standard cutting-plane method. The set being approximated may be either convex or non-convex. One such method for SOCP is given by Ben-Tal and Nemirovski [BN01b]. Another polyhedral outer approximation is given by Duran and Grossmann [DG86].

Convexification. Convexification is approximation of non-convex sets by their convex hulls. When the objective function is a linear function and the feasible set is closed, optimization over the convex hull is equivalent to optimization over the original feasible region. Therefore, convexification can be used, in principle, to convert an optimization from being over a non-convex region to being over a convex one. Naturally, this is not a way around the fundamental complexity of a given problem, since the convex hull of a

given set is not always easy to describe explicitly and hence may not be easy to optimize over. The cutting-plane method in MILP case can be considered as a convexification method. The motivation is to optimize over the convex hull, which is equivalent when the objective function is linear as long as the feasible set is closed.

1.5 Contribution

The primary motivation of this thesis is to propose and investigate various methods for solving MISOCPs. These methods include a number of variants of the branch-and-cut algorithm based on different relaxations, different methods of generating valid inequalities, and different approaches to algorithmic control. We conduct an extensive set of experiments to explore the effect of various algorithmic options, such as branching strategies, cut generation strategies, and search strategies. We introduce new software tools to carry out these experiments in a controlled environment. A particular highlight is the investigation of a recent disjunctive cut procedure for MISOCP from literature. We also test the scalibility of the tools introduced for parallel computers.

A second contribution is to formulate the inverse MILP problem as a conic optimization problem. We provide a cutting-plane algorithm for it and show that that decision version of inverse MILP is coNP–complete. Moreover, we prove that whether a given objective value is optimal for the inverse problem is D^{P}–complete. This is also true for the traditional MILP problem and one can hence argue that these problems are in the same complexity class.

Contributions of this thesis can be listed as follows.

- We introduce an outer-approximation algorithm for solving the (continuous) second-order cone optimization problems.
- We develop a unified algorithmic framework for solving MISOCP by a variety of

enumeration-based algorithms.

  - The core methodology used in the algorithms is branch and bound.
  - The relaxation employed can be either an SOCP or an LP relaxation.
  - The solver for solving the relaxation can be either an interior point method or the simplex algorithm.
  - The relaxation is strengthened by addition of dynamically generated inequalities customized for the chosen relaxation.
  - Branching is performed on variable disjunctions.

- We develop a unified algorithmic framework for separation and dynamic generation of violated valid inequalities.
  - When the relaxation is linear, valid inequalities take into account violations of relaxed conic structure.
  - When the relaxation is conic, disjunctive conic cuts based on relaxed integrality can be generated.
  - Inequalities from known classes for MILP relaxation can also be utilized in LP subproblems.
- We develop an extensive open source software framework that implements the ideas described above. Moreover this framework can be used for comparative testing and development of new algorithmic ideas. This framework includes the following.
  - COLA, an outer-approximation based solver for second-order cone optimization problems.
  - OsiConic, a generic solver application programming interface (API) for conic solvers, that extends the existing Open Solver Interface.

  - OsiConic implementations for Mosek, Ipopt, CPLEX, and COLA.
  - CGL-CONIC, a library of valid inequality generators for MISOCP.
  - DisCO, a unified, distributed memory parallel, extensible solver for MISOCP, implementing branch and cut.

- We perform extensive computational experiments to evaluate the effectiveness of various computational approaches. These experiments include the following.
  - Experiments to determine the optimal balance between branching and cutting for the LP subproblem based branch-and-cut algorithm.
  - Comparison of branching strategies for both LP and SOCP based branch and cut algorithm.
  - Investigation of the effectiveness of cuts from the MILP literature when used within an LP-based branch-and-cut algorithm.
  - Investigation of the effectiveness of disjunctive conic cuts for both LP- and SOCP-based branch-and-cut algorithms.
  - Evaluation of the scalibility of the introduced algorithms using parallel computers.
  - Comparison of the computational performance of SOCP and LP subproblem based branch-and-cut algorithms.

- We define three decision versions of the inverse MILP and determine their computational complexity. These problems are complete for the determined classes. They are the tightest complexity classes one can achieve without resolving P–NP problem.

1.6 Outline

This thesis continues as follows. In Chapter 2 we introduce a simple procedure for separating a given point from a Lorentz cone. Based on this, we present an outer-approximation method for solution of an SOCP. The outer-approximation method relaxes all conic constraints and iteratively solves the resulting LP relaxations in the fashion of the previously described cutting-plane method. Solutions to the relaxations are checked for feasibility with respect to the conic constraints. If a solution is ϵ-feasible, then the procedure terminates. If not, the LP relaxation is improved by adding violated valid inequalities. The algorithm continues in this fashion until a ϵ-feasible solution is found.

Chapter 3 introduces an algorithmic framework for solution of MISOCPs by a branch-and-cut approach. The overall approach is similar to the branch-and-cut algorithms in the MILP case except that in addition to the relaxation of integrality constraints, conic constraints may also be relaxed. In the case where conic constraints are relaxed, the relaxation is an LP and thus easy to solve. A solution to the original MISOCP is found whenever the solution to a given relaxation satisfies *both integrality and conic constraints*. Valid inequalities are added to remove solutions that are conic infeasible from the relaxation feasible region, while solutions violating integrality conditions are removed either by addition of valid inequalities or by branching. MILP cut procedures (Gomory, mixed integer rounding, etc.) can also be used to improve the bounds of the subproblems.

Chapter 3 surveys different valid inequalities for MISOCP feasible set from literature. Valid inequalities are meant to improve description of MISOCP feasible set by cutting off integer-infeasible solutions of relaxation subproblems. We refer valid inequalities that cuts integer-infeasible points as cut. Using cuts during branch-and-bound search is meant to reduce size of the tree and speed up achieving optimal solution and termination of algorithm. Chapter 3 investigates five of such cutting procedures from implementational perspective.

Chapter 4 introduces the various software libraries developed to solve MISOCP. One fundamental library is OsiConic, a generic interface for conic solvers. Another project is COLA (**c**onic **o**ptimization with **l**inear **a**pproximations) solver that uses an outer-approximation method to solve SOCP. Chapter 4 also presents the implementation of a conic solver interface for CPLEX, Ipopt, and Mosek. Another fundamental library presented is CglConic, which is a library of procedures for generating valid inequalities for MISOCP. Finally, we introduce the solver DisCO that combines all the mentioned libraries for the purpose of solving MISOCP. DisCO is a framework for implementing branch-and-cut algorithms and combines all of the previously mentioned projects. The algorithms in DisCO can be executed both on serial and on distributed-memory parallel computing platforms.

Chapter 6 presents the results of computational experiments conducted. Experiments were conducted on a standard set of benchmark instances for MISOCP. These instances are random instances defined by Góez [Góe13], **c**onic **b**enchmark **lib**rary 2014 (CBLIB) instances [Fri16] and MISOCP formulations of Euclidean Steiner tree problem. COLA is tested on the continuous relaxations of these problems. DisCO is tested with various algorithmic approaches and using various solvers for solving the relaxations, including COLA, CPLEX, Ipopt and Mosek. DisCO is tested both in serial and distributed-memory parallel mode.

Chapter 5 introduces inverse MILP. We formulate inverse MILP as a general (not necessarily second-order) conic optimization problem and give an algorithm that solves inverse MILP. Chapter 5 defines various decision problems related to inverse MILP. We derive the complexity of associated decision problems. Chapter 7 summarizes the work completed and explains the research directions we propose to investigate in the future.

Chapter 2

Second-Order Cone Optimization Problems

Second-Order Cone Optimization Problems (SOCPs) are an important special class of nonlinear optimization problem that generalize the well-known and efficiently solvable linear optimization problem. In SOCP, a linear objective function is optimized over the intersection of an affine set with the cartesian product of Lorentz cones. Although SOCPs are nonlinear (quadratic) convex optimization problems, the feasible region of an SOCP is convex and the problem itself can therefore be solved efficiently by interior point algorithms, among others.

In this Chapter, we first briefly discus duality theory and the interior point method for SOCP. We then detail a procedure for dynamically constructing a polyhedral approximation of a conic set and develop a cutting-plane algorithm based on this approximation procedure. The software implementation of the method is described in Chapter 4 and computational experiments with the method are described in Chapter 6.

2.1 Background

2.1.1 Duality Theory

In this section, we explore the duality theory for SOCP. The material presented here is based on Andersen, Roos, and Terlaky [ART03] and interested readers are referred to the paper for details. We assume an SOCP in the standard form (SOCP) given earlier, which is repeated here for completeness.

$$\begin{array}{lll} \min & c^\top x & \\ \text{s.t.} & Ax = b & \\ & x \in \mathbb{K}, & \end{array} \qquad \text{(SOCP)}$$

where $\mathbb{K}$ is the cartesian product of Lorentz cones defined in (SOCP). We refer to this as the *primal* problem in order to distinguish it from the *dual* problem, which is the following.

$$\begin{array}{lll} \max & b^\top y & \\ \text{s.t.} & A^\top y + s = c & \\ & s \in \mathbb{K}. & \end{array} \qquad \text{(D-SOCP)}$$

The conic constraints are the same in the primal and dual problems, since Lorentz cones are self-dual.

Theorem 2.1.1 (Weak Duality)**.** *Let x be a feasible solution to* (SOCP)*, and (y, s) be a feasible solution to* (D-SOCP)*, then*

$$c^\top x \geq b^\top y.$$

An SOCP is said to be *strictly feasible*, if there is a feasible solution to either (SOCP) or (D-SOCP) that satisfies the conic constraint strictly, i.e., $x \in int(\mathbb{K})$.

Theorem 2.1.2 (Strong Duality)**.** *If* (SOCP) *is strictly feasible and its optimal value is bounded or if* (D-SOCP) *is strictly feasible and its optimal value is finite, then* x *and* (y, s) *are optimal solutions if and only if*

$$c^\top x - b^\top y = x^\top s = 0,$$

x *is feasible for* (SOCP) *and* (y, s) *is feasible for* (D-SOCP)*.*

Theorem 2.1.3 (Primal Infeasibility)**.** *If there exists* (y, s) *with* $s \in \mathbb{K}$ *and such that* (y, s) *satisfies*

$$A^\top y + s = 0$$
$$b^\top y > 0,$$

then (SOCP) *is infeasible.*

Theorem 2.1.4 (Dual Infeasibility)**.** *If there exists* $x \in \mathbb{K}$ *satisfying*

$$Ax = 0$$
$$c^\top x < 0,$$

then (D-SOCP) *is infeasible.*

See Ben-Tal and Nemirovski [BN01a] for the proofs of the duality theorems.

2.1.2 Existing Algorithms

There are two main existing algorithms for solving (SOCP). The first one is based on a generalization of the well-known interior point algorithm (IPM) for solving linear optimization problems [NS96; AA95]. The second one is based on lifting and approximating the feasible region with a polyhedron [BN01b].

Interior Point Method

The most efficient method known to date to solve a single SOCP from scratch is the *interior point method* (IPM). The IPM is a well-studied method for solving (LP), (SOCP) and other convex optimization problems. In this section, we discuss IPMs for solving (SOCP). Interested readers are referred to Andersen, Roos, and Terlaky [ART03] for details of the computational implementation of IPMs for (SOCP).

We briefly present a primal-dual path-following IPM. This method is called a primal-dual method because the Newton method is applied to both the primal and dual problems. The primal-dual IPM presented works on the homogeneous model described by Kuhn, Tucker, and Dantzig [KTD56], this model is known as the Goldman and Tucker homogeneous model and is given by

$$\begin{aligned} Ax - b\tau &= 0, \\ A^\top y + s - c\tau &= 0, \\ -c^\top x + b^\top y - \kappa &= 0, \\ (x, \tau) &\in \mathbb{K} \times R_+, \\ (s, \kappa) &\in \mathbb{K} \times R_+. \end{aligned} \tag{2.1}$$

The homogeneous model itself is always feasible. A solution provides either a certificate of infeasibility or a solution to primal and dual problems. Lemma 2.1.5 gives this result.

Lemma 2.1.5. *Let* $(x^*, \tau^*, y^*, s^*, \kappa^*)$ *be a solution to* (2.1)*. Then*

1. $(x^*)^\top s^* + \tau^* \kappa^* = 0$.
2. *If* $\tau^* > 0$*, then* $\frac{(x^*, y^*, \tau^*)}{\tau^*}$ *is a primal-dual optimal solution.*
3. *If* $\kappa^* > 0$*, then at least one of the strict inequalities* $b^\top y^* > 0$ *and* $c^\top x^* < 0$ *holds. If the former inequality holds, then* (SOCP) *is infeasible. If the latter inequality holds,*

then (D-SOCP) *is infeasible.*

As such, the goal is to solve (2.1). In principle, one can try to do this directly using the Newton method. Termination of the Newton method depends on assumptions that may not be satisfied, however, and thus is not guaranteed. The IPM instead solves a modified versions of (2.1) successively in a fashion that guarantees convergence to an optimal solution in a finite number of iterations. The modified system solved in each iteration is as follows.

$$\begin{aligned} Ax - b\tau &= \gamma(Ax^0 - b\tau^0), \\ A^\top y + s - c\tau &= \gamma(A^\top y^0 + s^0 - c\tau^0), \\ -c^\top x + b^\top y - \kappa &= \gamma(-c^\top x^0 + b^\top y^0 - \kappa^0), \\ XSe &= \gamma u^0 e, \\ \tau\kappa &= \gamma u^0, \end{aligned} \tag{2.2}$$

where

$$\begin{aligned} u^0 &:= \frac{(x^0)^\top s^0 + \tau^0\kappa^0}{k+1}, \\ X &:= diag(X^1, \ldots, X^k), \\ S &:= diag(S^1, \ldots, S^k), \\ X^i &:= \begin{bmatrix} (T^i x^i)_1 & (T^i x^i)_{2:n}^\top \\ (T^i x^i)_{2:n} & (T^i x^i)_1 I^\top \end{bmatrix}, \\ S^i &:= \begin{bmatrix} (T^i s^i)_1 & (T^i s^i)_{2:n}^\top \\ (T^i s^i)_{2:n} & (T^i s^i)_1 I^\top \end{bmatrix}. \end{aligned}$$

If the i^{th} cone is a Lorentz cone, then T^i is an identity matrix. If it is a rotated Lorentz

cone, then T^i is defined as

$$T^i := \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \dots & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}.$$

$(x^0, \tau^0, y^0, s^0, \kappa^0)$ is a given starting point for which x^0 and s^0 are strictly feasible. The vector e is the vector of 1's with appropriate dimension. Note that the right-hand side of the complementarity constraint is modified to $\gamma u^0 e$. The set of solutions of this system with respect to varying γ is a smooth curve and called the *central path*. The central path starts with $(x^0, \tau^0, y^0, s^0, \kappa^0)$ $(\gamma = 1)$ and ends at a solution of (2.1) $(\gamma = 0)$.

The IPM follows the central path approximately in a way that guarantees convergence to an optimal solution. Roughly speaking, (2.2) is solved approximately by Newton's method in each iteration for decreasing γ values. The approximate solution is guaranteed to be in a certain neighborhood of the central path. Practical concerns for implementing the algorithm include scaling and step-size selection. Step-size selection plays an important role in the practical performance of the method.

Ben-Tal–Nemirovski Outer-Approximation Method

An alternative to the IPM is the outer-approximation method of Ben-Tal and Nemirovski [BN01b]. At a high level, their approximation method works as follows. First, they decompose all the cones in the problem to obtain a problem in which all cones are at most 3-dimensional. Then they approximate the 3-dimensional cones with supporting hyperplanes with respect to a given ϵ. At the end of the this process, they obtain an LP with larger number of variables and constraints then the original SOCP. Any feasible solution to

this LP is ϵ-feasible for the original SOCP. The number of variables and constraints are polynomials in the original number of variables, constraints and ϵ.

Rather than presenting the full details, we illustrate the cone decomposition process briefly with a 5-dimensional cone. Let such a Lorentz cone be given as $\|(y_1^0, y_2^0, y_3^0, y_4^0)\| \leq t$. The following is the decomposition of this cone into 3-dimensional cones.

$$\begin{aligned}
\|(y_1^0, y_2^0)\| &\leq y_1^1 \\
\|(y_3^0, y_4^0)\| &\leq y_2^1 \\
\|(y_1^1, y_2^1)\| &\leq t.
\end{aligned}$$

Note the increase in the number of variables. In this decomposition, the superscript represents the level of the variable. The first level variables are at level 0. Parents of y_i^l are y_{2i-1}^{l-1} and y_{2i}^{l-1}. At the top level we have variable t. In this decomposition, at level 2, we have variable t, at level 1, y_1^1 and y_2^1, and the original variables are at level 0.

After decomposing the larger cones to 3-dimensional cones, the 3-dimensional cones are approximated with linear constraints. The following is the linear approximation of a 3-dimensional Lorentz cone.

$$\begin{aligned}
\xi^0 &\geq |x_2| \\
\eta^0 &\geq |x_3| \\
\xi^j &= \cos\left(\frac{\pi}{2^{j+1}}\right)\xi^{j-1} + \sin\left(\frac{\pi}{2^{j+1}}\right)\eta^{j-1} \qquad j = 1, \ldots, \nu \\
\eta^j &\geq \left|-\sin\left(\frac{\pi}{2^{j+1}}\right)\xi^{j-1} + \cos\left(\frac{\pi}{2^{j+1}}\right)\eta^{j-1}\right| \quad j = 1, \ldots, \nu \\
\xi^\nu &\leq x_1 \\
\eta^\nu &\leq \tan\left(\frac{\pi}{2^{\nu+1}}\right)\xi^\nu,
\end{aligned}$$

where ν is an integer parameter of construction that controls how fine the approximation is. More variables (ξ^j, η^j) are added in this step. The number of linear constraints also

increase. The increase in the number of variables depends on how fine the approximation is.

2.2 A Cutting-Plane Algorithm

In Section 2.1.2, we reviewed the two main existing algorithms for (SOCP). In general, the interior point algorithm has better performance [Gli00], but the Ben-Tal and Nemirovski scheme has the distinct advantage that it reformulates the problem as a linear program. This means that in the context of a branch-and-cut algorithm, the latter can take advantage of the excellent warm-starting properties of linear programs to accelerate the branch-and-bound process.

The obvious disadvantage of the Ben-Tal and Nemirovski scheme is that it must approximate the feasible region using a potentially very large number of inequalities in a lifted space and the approximation is calculated a priori in order to ensure a given level of accuracy. Increasing the level of accuracy increases the number of required inequalities. To obtain an ϵ–accurate polyhedral approximation for an SOCP in dual conic form with k cones, the number of additional variables and constraints required is $O(k \ln \frac{2}{\epsilon})$. Solving a linear program this large to a high accuracy may be problematic and may defeat the advantage of the warm-starting.

On the other hand, it has long been known in the case of MILPs that there is a way out of such dilemmas and that is to dynamically generate only the inequalities that are required to produce the optimal solution, i.e., those inequalities that are binding at the optimal extreme point. Thus, it is natural to ask whether a dynamic outer-approximation scheme similar in spirit to Ben-Tal and Nemirovski approximation, but in which inequalities are generated dynamically rather than statically might be competitive. To test this, we developed and implemented such an algorithm.

2.2.1 The Separation Problem

In this section, we introduce a method to solve the separation problem, introduced in Definition 1.16, with respect to a given point $\overline{x} \in \mathbb{R}^n$ and the Lorentz cone $\mathbb{L}^n$. Theorem 2.2.1 presents the theoretical basis for the method. It is easy to check whether a given point is in a given Lorentz cone, so the result focuses on the second step of producing a separating hyperplane in the case that the given point is not in the Lorentz cone.

Theorem 2.2.1. *For any $\overline{x} \notin \mathbb{L}^n$, we have that*

$$(a^{\overline{x}})^\top x \geq 0 \; \forall x \in \mathbb{L}^n \tag{2.3}$$

and

$$(a^{\overline{x}})^\top \overline{x} < 0, \tag{2.4}$$

where $a^{\overline{x}} \in \mathbb{R}^n$ is defined by

$$a_1^{\overline{x}} = \|\overline{x}_{2:n}\|$$
$$a_{2:n}^{\overline{x}} = \overline{x}_{2:n}.$$

Proof. Let $\overline{x} \in \mathbb{R}^n \setminus \mathbb{L}^n$ be given. We first show that the inequality defined in (2.3) is valid for $\mathbb{L}^n$. For the sake of contradiction, assume the inequality does not hold for some $\hat{x} \in \mathbb{L}^n$, i.e., $0 > \|\overline{x}_{2:n}\|\hat{x}_1 - \overline{x}_{2:n}^\top \hat{x}_{2:n}$. Since $\hat{x} \in \mathbb{L}^n$, we have that $\hat{x}_1 \geq \|\hat{x}_{2:n}\|$. Then we have that

$$0 > \|\overline{x}_{2:n}\|\hat{x}_1 - \overline{x}_{2:n}^\top \hat{x}_{2:n} > \|\overline{x}_{2:n}\|\|\hat{x}_{2:n}\| - \overline{x}_{2:n}^\top \hat{x}_{2:n} \geq 0.$$

The first inequality follows from our initial assumption, the second from $\hat{x} \in \mathbb{L}^n$ and the third one is the Cauchy–Schwarz inequality. Hence, we have reached a contradiction. There is no member of $\mathbb{L}^n$ that violates the given inequality.

Next, we prove that $\overline{x}$ violates the given inequality, i.e.

$$\|\overline{x}_{2:n}\|\overline{x}_1 - \overline{x}_{2:n}^\top \overline{x}_{2:n} < 0.$$

For this, we use the fact that $\overline{x} \notin \mathbb{L}^n$, which means that $\overline{x}_1 < \|\overline{x}_{2:n}\|$, so we have that

$$\|\overline{x}_{2:n}\|\overline{x}_1 - \overline{x}_{2:n}^\top \overline{x}_{2:n} = \|\overline{x}_{2:n}\|\overline{x}_1 - \|\overline{x}_{2:n}\|^2 < \|\overline{x}_{2:n}\|^2 - \|\overline{x}_{2:n}\|^2 = 0,$$

and this completes the proof. □

Note that same as cutting points, Theorem 2.2.1 can be used for restricting directions. For a direction $\overline{d} \notin \mathbb{L}^n$, $(a^{\overline{d}})^\top \overline{d} < 0$ and $(a^{\overline{d}})^\top x \geq 0 \; \forall x \in \mathbb{L}^n$.

Theorem 2.2.2. *The hyperplane*

$$\{x \in \mathbb{R}^n \mid (a^{\overline{x}})^\top x = 0\} \tag{2.5}$$

associated with the valid inequality described in Theorem 2.2.1 supports $\mathbb{L}^n$ *along an extreme ray* $\mathcal{E} = \{\gamma a^{\overline{x}} \mid \gamma \in \mathbb{R}_+\}$.

Proof. It is easy to see that $\mathcal{E}$ is an extreme ray of $\mathbb{L}^n$. It is also in the given hyperplane. □

Theorem 2.2.3. *Among all hyperplanes separating a given* $\overline{x} \notin \mathbb{L}^n$ *from* $\mathbb{L}^n$*, the hyperplane* (2.5) *associated with the valid inequality given in Theorem 2.2.1 is maximally distant from* $\overline{x}$ *in terms of the* l_2 *norm.*

Proof. To find the valid inequality whose associated hyperplane is maximally distant from $\overline{x}$, one needs to project $\overline{x}$ onto $\mathbb{L}^n$ and generate the supporting hyperplane passing through this projected point.

$\overline{x}$ can be projected on $\mathbb{L}^n$ by finding the closest point of $\mathbb{L}^n$ to it. This point can be

found by solving the following problem,

$$\begin{aligned} \min \quad & z_1 \\ \text{s.t.} \quad & z_{2:n} - x = -\overline{x} \\ & z \in \mathbb{L}^{n+1} \\ & x \in \mathbb{L}^n. \end{aligned} \tag{Projection}$$

Both the projection problem and its dual are strictly feasible and strong duality holds. Any point satisfying the following optimality conditions is optimal.

$$\begin{aligned} (z_1, x - \overline{x}) &\in \mathbb{L}^{n+1} && \text{(Primal Feasibility)} \\ x &\in \mathbb{L}^n && \text{(Primal Feasibility)} \\ (1, -u) &\in \mathbb{L}^{n+1} && \text{(Dual Feasibility)} \\ u &\in \mathbb{L}^n && \text{(Dual Feasibility)} \\ z_1 - u^\top (x - \overline{x}) &= 0 && \text{(Complimentary Slackness)} \\ u^\top x &= 0 && \text{(Complimentary Slackness)} \\ z_1 &= -\overline{x}^\top u. && \text{(Strong Duality)} \end{aligned}$$

Assuming $\overline{x} \notin \mathbb{L}^n \bigcup -\mathbb{L}^n$ (otherwise, the solution is trivially 0), the solution is

$$x^* = \frac{\overline{x}_1 + \|\overline{x}_{2:n}\|}{2\|\overline{x}_{2:n}\|} \begin{bmatrix} \|\overline{x}_{2:n}\| \\ \overline{x}_{2:n} \end{bmatrix}.$$

The vector defining the valid inequality is then computed as the gradient (normal to the tangent hyperplane) at x^*. The gradient is $(x_1, -x_2, \ldots, -x_n)$. Then we have that

$$\frac{\overline{x}_1 + \|\overline{x}_{2:n}\|}{2\|\overline{x}_{2:n}\|} \left(\|\overline{x}_{2:n}\| x_1 - \overline{x}_{2:n}^\top x_{2:n} \right) \geq 0 \ \ \forall x \in \mathbb{L}^n.$$

This inequality is the one given in Theorem 2.2.1 except that it is scaled by $\frac{\overline{x}_1+\|\overline{x}_{2:n}\|}{2\|\overline{x}_{2:n}\|}$. The projected point x^* is on the extreme ray of Theorem 2.2.2 that constitutes the support of the hyperplane formed by the set of points that satisfy the inequality at equality. □

Theorem 2.2.4. *Given $\overline{x} \notin \mathbb{L}^n_{rot}$, we have that*

$$(a^{\overline{x}}_{\text{rot}})^\top x \geq 0 \; \forall x \in \mathbb{L}^n_{rot},$$

where $a^{\overline{x}}_{\text{rot}} \in \mathbb{R}^n$ is defined by

$$\begin{aligned}
(a^{\overline{x}}_{\text{rot}})_1 &= \sqrt{(-\overline{x}_1+\overline{x}_2)^2 + 2\overline{x}^\top_{3:n}\overline{x}_{3:n}} + (-\overline{x}_1+\overline{x}_2),\\
(a^{\overline{x}}_{\text{rot}})_2 &= \sqrt{(-\overline{x}_1+\overline{x}_2)^2 + 2\overline{x}^\top_{3:n}\overline{x}_{3:n}} - (-\overline{x}_1+\overline{x}_2),\\
(a^{\overline{x}}_{\text{rot}})_{3:n} &= -2\overline{x}_{3:n}.
\end{aligned}$$

and that

$$(a^{\overline{x}}_{\text{rot}})^\top \overline{x} < 0.$$

Proof. For the sake of contradiction assume $\exists \hat{x} \in \mathbb{L}^n_{rot}$ such that

$$(a^{\overline{x}}_{\text{rot}})^\top \hat{x} < 0.$$

Then a contradiction can be obtained by the following chain of equality and inequalities. The first inequality is due to our assumption. The second and third equalities are obtained by substituting values for $a^{\overline{x}}_{\text{rot}}$ and performing some algebraic manipulations. The last

inequality is the Cauchy–Schwarz inequality.

$$
\begin{aligned}
0 &> (a^{\overline{x}}_{\text{rot}})^\top x \\
&= (\hat{x}_1 + \hat{x}_2)\sqrt{(-\overline{x}_1 + \overline{x}_2)^2 + 2\overline{x}_{3:n}^\top \overline{x}_{3:n}} + (\hat{x}_1 - \hat{x}_2)(-\overline{x}_1 + \overline{x}_2) - 2\overline{x}_{3:n}^\top \hat{x}_{3:n} \\
&= (\hat{x}_1 + \hat{x}_2)\left\| \begin{array}{c} \overline{x}_1 - \overline{x}_2 \\ \sqrt{2}\overline{x}_{3:n} \end{array} \right\| - \left[\begin{array}{c} \overline{x}_1 - \overline{x}_2 \\ \sqrt{2}\overline{x}_{3:n} \end{array} \right]^\top \left[\begin{array}{c} \hat{x}_1 - \hat{x}_2 \\ \sqrt{2}\hat{x}_{3:n} \end{array} \right] \\
&\geq \left\| \begin{array}{c} \hat{x}_1 - \hat{x}_2 \\ \sqrt{2}\hat{x}_{3:n} \end{array} \right\| \left\| \begin{array}{c} \overline{x}_1 - \overline{x}_2 \\ \sqrt{2}\overline{x}_{3:n} \end{array} \right\| - \left[\begin{array}{c} \overline{x}_1 - \overline{x}_2 \\ \sqrt{2}\overline{x}_{3:n} \end{array} \right]^\top \left[\begin{array}{c} \hat{x}_1 - \hat{x}_2 \\ \sqrt{2}\hat{x}_{3:n} \end{array} \right] \\
&\geq 0.
\end{aligned}
$$

The step before the Cauchy–Schwarz inequality is crucial and holds due to the fact that $\hat{x} \in \mathbb{L}^n_{rot}$, so that

$$
\begin{aligned}
2\hat{x}_1\hat{x}_2 &\geq \hat{x}_{3:n}^\top \hat{x}_{3:n} \\
4\hat{x}_1\hat{x}_2 &\geq 2\hat{x}_{3:n}^\top \hat{x}_{3:n} \\
(\hat{x}_1 + \hat{x}_2)^2 - (\hat{x}_1 - \hat{x}_2)^2 &\geq 2\hat{x}_{3:n}^\top \hat{x}_{3:n} \\
(\hat{x}_1 + \hat{x}_2)^2 &\geq (\hat{x}_1 - \hat{x}_2)^2 + 2\hat{x}_{3:n}^\top \hat{x}_{3:n} \\
\hat{x}_1 + \hat{x}_2 &\geq \left\| \begin{array}{c} \hat{x}_1 - \hat{x}_2 \\ \sqrt{2}\hat{x}_{3:n} \end{array} \right\|.
\end{aligned}
$$

□

2.2.2 Overall Algorithm

Algorithm 3 presents a cutting-plane algorithm for (SOCP). It uses the separation procedure described in the previous section for dynamically generating the valid inequalities associated with each of the cones comprising the cartesian product $\mathbb{K}$. In the algorithm, we

define a parametric family of polyhedra that contain the feasible region $\mathcal{S}_{\text{SOCP}}$ of (SOCP) as follows.

Definition 2.1. ($\mathcal{P}(\mathcal{E})$) For a given set $\mathcal{E} \subseteq \mathbb{Q}^n$, $\mathcal{P}(\mathcal{E})$ is the polyhedron defined by the inequalities of Theorem 2.2.1 corresponding to the members of set $\mathcal{E}$ for each of the Lorentz cones comprising the cartesian product $\mathbb{K}$, plus the linear constraints from (SOCP). The formal description of this polyhedron is

$$\begin{aligned} & Ax = b \\ & x_1^i \geq 0 \qquad\qquad i = 1, \ldots, k \\ & (a^{\overline{x}^i})^\top (x^i) \geq 0 \quad \overline{x} \in \mathcal{E},\ i \in \mathcal{K}(\overline{x}). \end{aligned} \tag{$\mathcal{P}(\mathcal{E})$}$$

Here, $1, \ldots, k$ are the indices of the Lorentz cones comprising $\mathbb{K}$. For a given $x \in \mathbb{R}^n$, $\mathcal{K}(x)$ are the indices of cones to which x does not belong and $a^{\overline{x}^i}$ is the left-hand side vector of the inequality from Theorem 2.2.1.

Note that initially, the constraint $x^i \in \mathbb{L}^{n_i}$ is replaced by the relaxed constraint $x_1^i \geq 0$, which a valid since $\{x^i \in \mathbb{R}^{n_i} \mid x_1^i \geq 0\} \supset \mathbb{L}^{n_i}$. Algorithm 3 is the proposed cutting plane algorithm for solving (SOCP). In this algorithm, the set $\mathcal{E}^j$ is the set of (infeasible) points generated so far in the algorithm and $\mathcal{P}(\mathcal{E}^j)$ is the current polyhedral approximation.

Algorithm 3 solves (SOCP) by solving a sequence of problems whose feasible regions are successively tighter polyhedral approximations of the original feasible region. In iteration j of the algorithm, we optimize over the approximation $\mathcal{P}(\mathcal{E}^j)$, which is either infeasible, unbounded, or has an optimal solution. When it is infeasible, we conclude that (SOCP) is infeasible (lines 12–13). When it is unbounded (lines 5–10), we determine whether the direction of unboundedness $\overline{d}$ is ϵ-feasible for $\mathbb{K}$. This can be done by checking whether $\overline{d}$ is ϵ-feasible for each cone in $\mathbb{K}$. If it is, then (SOCP) is unbounded. If not, a valid inequality violated by the direction of unboundedness is generated according

Algorithm 3 Cutting-plane Algorithm for (SOCP)

$\mathcal{E}^0 \leftarrow \emptyset$.
$j \leftarrow 0$.
while TRUE **do**
  $z^j = \min_{x \in \mathcal{P}(\mathcal{E}^j)} c^\top x$.
  if $z^j = -\infty$ **then**
    Let $\alpha \overline{d} \in \mathcal{P}(\mathcal{E}^j) \forall \alpha \in \mathbb{R}_+$ and $c^\top \overline{d} < 0$.
    if $\overline{d} \in \mathbb{K}_\epsilon$ **then**
      (SOCP) is unbounded, STOP.
    else
      $\mathcal{E}^{j+1} \leftarrow \mathcal{E}^j \bigcup \{\overline{d}\}$.
    end if
  else if $z^0 = \infty$ **then**
    (SOCP) is infeasible, STOP.
  else
    Let $x^j \in \operatorname{argmin}_{x \in \mathcal{P}(\mathcal{E}^j)} c^\top x$
    if $x^j \notin \mathbb{K}_\epsilon$ **then**
      $\mathcal{E}^{j+1} \leftarrow \mathcal{E}^j \bigcup \{x^j\}$.
    else
      x^j is ϵ-optimal for (SOCP), STOP.
    end if
  end if
  $j \leftarrow j + 1$
end while

to Theorem 2.2.1. If $\mathcal{P}(\mathcal{E}^j)$ has an optimal solution (lines 14–19), then feasibility of the solution x^i with respect to conic constraints is checked. If x^i is ϵ-feasible for $\mathbb{K}$, then it is also ϵ-optimal for (SOCP). If not, x^i is added to $\mathcal{E}^i$, which means that the corresponding valid inequalities are generated and added to the polyhedral approximation.

After augmenting the set of points generated so far, the relaxation is re-solved with the improved polyhedral approximation $\mathcal{P}(\mathcal{E}^j)$. These steps are repeated until one of the three stopping criteria is achieved. In the presented algorithm, not all of the supporting hyperplanes are generated a priori. Instead, approximating supports are generated and added to LP dynamically as needed. This can be considered as a method of approximating the conic constraints only in the region needed to solve the problem, as is usual with cutting-plane algorithms.

Note that when the simplex method is used to solve the LPs, Algorithm 3 can exploit its warm start capability. In each iteration, more constraints are added to the approximating LP. The number of constraints added are at most equal to the number of cones comprising $\mathcal{K}$. The dual simplex method is known for its ability to quickly re-solve a given instance after constraints are added to the problem.

Note that some of the cuts generated might become redundant in the subsequent iterations. In practical implementations, redundant cuts can be removed from the $(\mathcal{P}(\mathcal{E}))$ formulation to reduce the size of $(\mathcal{P}(\mathcal{E}))$. Algorithm 3 is implemented in COLA solver that is introduced in Section 4.2.

2.3 Comparison to Ben-Tal and Nemirovski Method

As discussed earlier, Ben-Tal and Nemirovski [BN01b] also proposed an outer approximation algorithm for (SOCP). The approach presented is Section 2.2 differs from their procedure in important respects. The main differences are (1) the problem is not lifted to higher dimensions to reduce the cone dimensions, (2) not all of the approximation

hyperplanes are added apiori. Ben-Tal and Nemirovski represent the whole problem as an LP. Once the approximation is created, its solution will be an approximate solution for (SOCP). The cutting-plane algorithm introduced in this section does not approximate all cones fully. It initially replaces all conic constraints with constraints $x_j \geq 0$, where x_j is the leading variable. If the resulting relaxation yields a solution that is feasible for some conic constraints, those cones are not approximated since they are already feasible. Moreover, for the cones that the relaxation problem does not satisfy, the introduced algorithm approximates them only with a single hyperplane corresponding to a valid inequality violated by solution to the relaxation.

In summary, the cutting-plane algorithm presented approximates the cones dynamically as needed, whereas the Ben-Tal and Nemirovski approach approximates all cones fully a priori. In our approximation scheme, we add only linear constraints only. The Ben-Tal and Nemirovski procedure both lifts the problem by adding additional variables and adds constraints. We believe with the current LP technology (warm start capability after addition of cuts), our approach is promising in terms of efficiency.

Glineur [Gli00] extends the Ben-Tal and Nemirovski approximation procedure by improving the outer-approximation method so that it needs fewer variables and constraints to approximate the feasible region of (SOCP). It also provides a computational study and reports that solving the original SOCP with IPM is more efficient than solving the linear approximations, even at low accuracies.

2.4 Comparison to IPM

IPM is a well studied method to solve (SOCP) with theoretical guarantees. The modern implementations of IPM in commercial solvers have been successful in solving (SOCP) efficiently. One disadvantage of the IPM is its relatively weaker warm starting capability (compared to simplex) and this capability is heavily relied upon in solving discrete opti-

mization problems in a branch-and-bound framework. Warm starting IPM is an active research topic [SAY13; ÇPT17]. Skajaa, Andersen, and Ye [SAY13] give a warm starting method that reports reductions in solution time in the range of 30–75 % depending on the problem class and magnitude of the problem perturbation. Where after bound update on a variable, simplex method is expected to obtain optimality in a few iterations and simplex iterations are cheap.

There have been computational studies that test the performance of outer-approximation methods for (SOCP). One such study is Glineur [Gli00]. It suggests a variant of Ben-Tal and Nemirovski outer approximation that leads to relatively smaller LP approximations and reports that outer approximation is slow compared to IPM. Another computational study is Vielma, Ahmed, and Nemhauser [VAN08]. This study is on discrete problems and introduces a branch-and-bound algorithm based on Ben-Tal and Nemirovski approximation. It reports that branch and bound based on Ben-Tal and Nemirovski approximation is favorable compared to an IPM based branch and bound.

In the cutting-plane algorithm proposed in this section, the feasible region of an SOCP is approximated by a polyhedron. The relaxations are improved dynamically rather than by constructing a single large polyhedral approximation a priori. One advantage of this method is the cones are approximated well only in the relevant regions. A second advantage is that the problems can be warm started when new approximating cuts are added. Computational performance of this method will be compared to IPM in Chapter 6.

2.5 Conclusion

In this chapter, we introduced a closed form linear cut to separate a given point from Lorentz and rotated Lorentz cones. We show that the valid inequality is the strongest possible when measured in terms of depth and that it supports the cone along an extreme ray.

2.5. CONCLUSION

We proposed a cutting-plane algorithm that uses the valid inequalities described to solve (SOCP). We listed similarities and differences of the defined procedure to the existing outer approximation methods.

In Chapter 3, we build a branch-and-bound algorithm on top of the separation procedure introduced. In Chapter 4, we introduce software libraries that implement the separation procedure and cutting-plane algorithm introduced in this chapter. In Chapter 6, we measure the performance of the algorithms introduced and compare them to the IPM implementations of various commercial and open source solvers. We also test the effectiveness of the procedures defined in a branch-and-bound algorithm at solving MISOCP.

Chapter 3

Mixed Integer Second-Order Cone Optimization Problems

This chapter addresses computational methods for solution of MISOCP. MISOCP is a special case of the more general class of MINLPs and a generalization of the well-studied case of MILP. As such, MISOCP is an NP–hard problem and no provably efficient algorithm is yet known for it. Although the same is true for the case of MILP, progress in recent decades has led to the successful implementation of algorithms that have proven to work well in practice on many real-world instances. Our goal in this study is to exploit the hard-won knowledge that has been gained regarding how to solve MILPs in order to develop improved solution techniques for MISOCP. For a detailed introduction and definitions of related problem classes like SOCP and LP, see Section 1.2.2.

As we have motivated earlier, there exist many important applications of MISOCP in the literature. MISOCP is used to model design of supply chain networks [ABS12], telecommunication networks [FM04], cardinality constrained portfolio optimization [BS09], turbine balancing and minimum Steiner Tree problem [FM04].

In the remainder of the chapter, we review existing methodology and introduce our

proposed computational framework. In Section 3.1, we give an overview of the existing approaches to solve MISOCP. This section includes a discussion of a branch-and-bound algorithm based on Ben-Tal and Nemirovski outer-approximation method given by Vielma, Ahmed, and Nemhauser [VAN08]. In Section 3.2, we review the known classes of valid inequalities. In Section 3.3, we introduce a flexible branch-and-cut framework for MISOCP designed to allow fine-tuning of various aspects of the algorithm for different problem classes. Advantages of the proposed algorithmic framework as compared to existing approaches are also discussed.

3.1 Existing Algorithms

3.1.1 SOCP-based Branch and Bound

Branch and bound with SOCP relaxations is the most natural generalization of the branch-and-bound algorithm for MISOCP solvers. In this approach, integrality constraints are relaxed to obtain SOCP relaxations. The branch-and-bound algorithm, as described earlier, is then utilized to systematically search for solutions to the original problem that satisfy the relaxed integrality constraints. We refer this algorithm in what follows as `bb-socp`. In this algorithm, the SOCP solver is employed as a black box and any available solver can be used. In this study, we conduct experiments in which the SOCP subproblem relaxations are solved both with IPM and with an outer-approximation approach similar to the one described in Algorithm 3 in Chapter 2 with the additional allowance for valid inequalities arising from violated valid disjunctions derived from integrality restrictions. To the best of our knowledge, the commercial solver Mosek, which is a standard in the field, uses such an SOCP-based branch-and-bound approach and solves the relaxations with IPM [MOS15].

3.1.2 Branch and Bound with Linear Relaxation

In this section, we briefly describe a branch-and-bound method to solve MISOCP based on the Ben-Tal and Nemirovski outer-approximation approach. Vielma, Ahmed, and Nemhauser [VAN08] show that even though Ben-Tal and Nemirovski type of approximations are not very efficient for purely continuous problems, they are efficient at solving MISOCP. They show this by introducing a branch-and-bound algorithm based on a lifted polyhedral relaxation given by Ben-Tal and Nemirovski [BN01b] and Glineur [Gli00].

Vielma, Ahmed, and Nemhauser [VAN08] describe their algorithm in the context of the more general class of nonlinear problems known as a *mixed integer nonlinear convex optimization problem* (MICXP), stated as

$$\begin{aligned} z_{\text{MICXP}} := \quad & \min \quad c^\top x + d^\top y \\ & \text{s.t.} \quad (x, y) \in \mathcal{C} \\ & \qquad x \in \mathbb{Z}^n. \end{aligned} \tag{MICXP}$$

where $\mathcal{C}$ is a convex set. This class clearly includes (MISOCP) as a special case. We summarize their algorithm here.

First, we consider the continuous relaxation

$$\begin{aligned} z_{\text{CXP}} := \quad & \min \quad c^\top x + d^\top y \\ & \text{s.t.} \quad (x, y) \in \mathcal{C} \\ & \qquad x \in \mathbb{R}^n. \end{aligned} \tag{CXP}$$

of (MICXP). Now, let $\mathcal{P}$ denote a lifted polyhedral relaxation of $\mathcal{C}$ obtained using a procedure similar to that of Ben-Tal and Nemirovski [BN01b]. The relationship between $\mathcal{C}$ and $\mathcal{P}$ is

$$\mathcal{C} \subset \{(x, y) \in \mathbb{R}^{n+p} \mid \exists v \in \mathbb{R}^q \text{s.t.} (x, y, v) \in \mathcal{P}\}.$$

We denote by (CXLP) the approximation of (CXP) in the lifted space, as follows.

$$\begin{aligned} z_{\text{CXLP}} := \quad & \min \quad && c^\top x + d^\top y \\ & \text{s.t.} \quad && (x, y, v) \in \mathcal{P}. \end{aligned} \tag{CXLP}$$

For a given $\hat{x} \in \mathbb{Z}^n$, we define

$$\begin{aligned} z_{\text{CXP}(\hat{x})} := \quad & \min \quad && c^\top \hat{x} + d^\top y \\ & \text{s.t.} \quad && (\hat{x}, y) \in \mathcal{C}. \end{aligned} \tag{CXP($\hat{x}$)}$$

For any $(l^k, u^k) \in \mathbb{Z}^n$, CXLP$(l^k, u^k)$ denotes CXLP with additional constraints $l^k \leq x \leq u^k$, CXP(l^k, u^k) denotes CXP with additional constraints $l^k \leq x \leq u^k$. Branch-and-bound node k is denoted by (l^k, u^k, LB^k), where (l^k, u^k) are bounds for variable x and LB^k is a lower bound on $z_{\text{CXP}(l^k, u^k)}$.

Algorithm 4 gives the lifted branch-and-bound algorithm of Vielma, Ahmed, and Nemhauser [VAN08]. Algorithm 4 does not assume that a solution for (CXLP) is also a solution of (CXP). This is due to their relatively large choice of approximation parameter value. Due to this, (CXP) is approximated loosely rather than high accuracy. (CXLP) is considered as a loose approximation of (CXP). There are advantages and disadvantages to this. Disadvantage is that (CXLP) approximates loosely and a solution of it can not be considered a solution of (CXP). The advantage is size of (CXLP) is smaller since the size increases with the accuracy demand. Due to the fact that a solution to (CXLP) may not be a solution to (CXP), extra steps are needed in the branch-and-bound algorithm.

Algorithm 4 tries to achieve integer feasibility first by doing branch and bound on (CXLP) problems until an integer solution is found, i.e. $(\hat{x}^k, \hat{y}^k)$. When an integer solution is found then we worry about its feasibility to other constraints. Integer (CXLP) solution is loosely feasible to (CXP). Once this integer solution is found, we solve (CXP($\hat{x}$)). If (CXP($\hat{x}$)) is

Algorithm 4 Lifted branch-and-bound algorithm [VAN08]

$LB^0 \leftarrow -\infty, UB \leftarrow +\infty$
$l_i^0 \leftarrow -\infty, u_i^0 \leftarrow +\infty$ for all $i \in \{1, \ldots, n\}$.
$\mathcal{H} \leftarrow \{(l^0, u^0, LB^0)\}$.
while $\mathcal{H} \neq \emptyset$ **do**
    Select and remove a node $(l^k, u^k, LB^k) \in \mathcal{H}$.
    Solve CXLP(l^k, u^k).
    if CXLP(l^k, u^k) is feasible and $z_{CXLP(l^k,u^k)} < UB$ **then**
        Let $(\hat{x}^k, \hat{y}^k)$ be the optimal solution to CXLP(l^k, u^k).
        if $\hat{x}^k \in \mathbb{Z}^n$ **then**
            Solve CXP$(\hat{x}^k)$.
            if CXP$(\hat{x}^k)$ is feasible and $z_{CXP(\hat{x}^k)} < UB$ **then**
                $UB \leftarrow z_{CXP(\hat{x}^k)}$.
            end if
            if $l^k \neq u^k$ and $z_{CXLP(l^k,u^k)} < UB$ **then**
                Solve CXLP(l^k, u^k).
                if CXLP(l^k, u^k) is feasible and $z_{CXLP(l^k,u^k)} < UB$ **then**
                    Let $(\tilde{x}^k, \tilde{y}^k)$ be the optimal solution to CXLP(l^k, u^k).
                    if $\tilde{x}^k \in \mathbb{Z}^n$ **then**
                        $UB \leftarrow z_{CXLP(l^k,u^k)}$.
                    else
                        Pick i_0 in $\{i \in \{1, \ldots, n\} : \tilde{x}_i^k \notin \mathbb{Z}\}$.
                        $l_i \leftarrow l_i^k, u_i \leftarrow u_i^k$ for all $i \in \{1, \ldots, n\} \setminus \{i_0\}$.
                        $u_{i0} \leftarrow \lfloor \tilde{x}_{i0}^k \rfloor, l_{i0} \leftarrow \lfloor \tilde{x}_{i0}^k \rfloor + 1$.
                        $\mathcal{H} \leftarrow \mathcal{H} \cup \{(l^k, u, z_{CXP(l^k,u^k)}), (l, u^k, z_{CXP(l^k,u^k)})\}$.
                    end if
                end if
            end if
        else
            Pick i_0 in $\{i \in \{1, \ldots, n\} : \hat{x}_i^k \notin \mathbb{Z}\}$.
            $l_i \leftarrow l_i^k, u_i \leftarrow u_i^k$ for all $i \in \{1, \ldots, n\} \setminus \{i_0\}$.
            $u_{i0} \leftarrow \lfloor \hat{x}_{i0}^k \rfloor, l_{i0} \leftarrow \lfloor \hat{x}_{i0}^k \rfloor + 1$.
            $\mathcal{H} \leftarrow \mathcal{H} \cup \{(l^k, u, z_{CXLP(l^k,u^k)}), (l, u^k, z_{CXLP(l^k,u^k)})\}$.
        end if
    end if
    Remove every node $(l^k, u^k, LB^k) \in \mathcal{H}$ such that $LB^k > UB$.
end while

feasible, then we check whether lower bound can be updated. Note that at this point even though lower bound is updated we are not done with this node and an even better solution (x variable values different than $\hat{x}$ when $l^k \neq u^k$) might be feasible for this node. If this is the case, then we have to solve the (CXP), which is the most expensive continuous problem in the algorithm. If solution $(\tilde{x}^k, \tilde{y}^k)$ is integer feasible, then we check whether lower bound can be updated, if not branch.

Note that in this algorithm expensive steps occur when an integer solution to (CXLP) is found. When this occurs, we need to solve both (CXP($\hat{x}$)) and CXP(l^k, u^k) assuming node is not fathomed due to inferior bound.

3.1.3 Other Outer-Approximation Schemes

As noted earlier, (MISOCP) can be considered a special case of (MICXP). In this section, we discuss outer-approximation schemes for (MICXP) that are also applicable for (MISOCP).

There are various approaches for solving (MICXP) in the literature. Duran and Grossmann [DG86] give an outer-approximation algorithm for a special case of (MICXP) in which the integer variables are in the linear constraints only. This algorithm requires the solution of both convex NLP and MILP subproblem relaxations. Outer-approximation cuts are generated at integer feasible solutions. Quesada and Grossmann [QG92] propose an algorithm to solve (MICXP) for which the NLP relaxations are easy to solve. Abhishek, Leyffer, and Linderoth [ALL10] present a solver based on the algorithm given in Quesada and Grossmann [QG92]. Bonami et al. [Bon+08] propose a hybrid branch-and-bound algorithm that uses OA together with the solution of convex NLP subproblem relaxations. The Bonmin software is an (MICXP) solver that implements this algorithm, using IPM to solve the convex NLPs that arise in the branch and bound.

Fletcher and Leyffer [FL94] solve (MICXP) by generalizing the outer approximation

method of Duran and Grossmann [DG86]. They relieve the restriction of integer variables being in linear constraints only.

BONMIN (Basic Open-source Nonlinear Mixed INteger) is a solver for (MICXP). BONMIN implements 3 algorithms, outer approximation (OA), NLP based branch and bound and a hybrid algorithm that uses OA together with NLP solves. BONMIN uses CBC to solve MILP approximation, CLP to solve LP approximations and CGL to generate cuts for the MILP approximation. It uses Ipopt to solve CXPs. BONMIN warm starts Ipopt by using primal and dual solutions of the parent node. Bonami et al. [Bon+08] show that BONMIN hybrid algorithm performs best compared to pure OA and NLP based branch and bound.

FilMINT is another solver for (MICXP) that implements a hybrid algorithm similar to outer approximation given by Abhishek, Leyffer, and Linderoth [ALL10]. FilMINT solves an MILP and enforces feasibility of nonlinear constraints through cuts. For each integer solution found during branch and bound, it fixes the integer values found and solves NLP. In outer-approximation algorithm, different MILPs are solved, where in FilMINT, only one MILP formulation exists and this formulation is improved by the outer-approximation cuts generated. FilMINT uses filterSQP solver by Fletcher, Leyffer, and Toint [FLT02] to solve convex NLP, and MINTO (Mixed INTeger Optimizer) solver by Nemhauser, Savelsbergh, and Sigismondi [NSS94] as MILP solver. filterSQP implements an active set algorithm with warm starting capabilities that can exploit warm started primal and dual iterates. In FilMINT solver, filterSQP is warm started with the LP solution.

3.1.4 Global Optimization Approaches

Global optimization algorithms/solvers can be used to solve (MISOCP). A brief introduction to global optimization approaches is given in Section 1.4.4.

There are a number of existing solvers, both open source and commercial, for solving

global optimization problems. These are ANTIGONE [MF14], BARON [TS05; Sah17], COCOS, COUENNE [Bel+09], GlobSol [Kea03], ICOS [Leb09], LGO [Pin98], LINGO, OQNLP, Premium solver, MINOS [MS83] and SCIP [VG16]. See Neumaier et al. [Neu+05] for an earlier study on comparison of global optimization solvers.

COUENNE [Bel+09] is an exact global solver for MINLP with not necessarily convex objective and constraints. It implements spatial branch and bound method. It carries reformulation and linearization of the non-convex functions involved. COUENNE implements two different bound tightening methods, optimality based and feasibility based.

3.2 Valid Inequalities

This section surveys literature regarding different general inequalities valid for the feasible set of (MISOCP). These valid inequalities are expressed for different versions of MISOCP formulations in different publications.

Like MILPs, MISOCPs are solved using the branch-and-bound method. Valid inequalities improve MISOCP feasible set by cutting off integer-infeasible solutions of relaxation problems. We refer valid inequalities that cuts integer-infeasible points as cuts. Cuts are used during branch-and-bound search to reduce size of the tree and speed up achieving optimal solution.

In this section we investigate five different procedures for generating valid inequalities violated by a given infeasible point. These are

- conic mixed-integer rounding (MIR) cuts given by Atamtürk and Narayanan [AN10] for general mixed integer case,
- conic Gomory cuts given by Çezik and Iyengar [ÇI05] for mixed 0–1 problems,
- lift-and-project cuts defined by Stubbs and Mehrotra [SM99] for mixed 0–1 convex problems,

- **d**isjunctive **c**onic **c**uts (DCC) and **d**isjunctive **cy**lindirical **c**uts (DCyC) defined by Belotti et al. [Bel+13] for general mixed integer case,
- two-term disjunctions defined by Kılınç-Karzan and Yıldız [KY14] for general mixed integer case.

Note that these cuts all apply to some subclass of MISOCP, though some can themselves be applied in more general cases. The MIR cuts of Atamtürk and Narayanan [AN10] are valid in the general MISOCP case, as are the cuts of Belotti et al. [Bel+13] and Kılınç-Karzan and Yıldız [KY14].

The Gomory cuts described by Çezik and Iyengar [ÇI05] are for general conic optimization problems with both binary and continuous variables. The inequalities of Stubbs and Mehrotra [SM99] are for general convex optimization problems, again with both binary and continuous variables.

3.2.1 Conic MIR Cuts

Conic MIR cuts, introduced by Atamtürk and Narayanan [AN10], are a generalization of MIR cuts [NW90] for MILPs to the MISOCP case. Atamtürk and Narayanan [AN10] describe the procedure for MISOCP formulation with constraints in conic dual form. This section summarizes the cut procedure and discusses how it can be modified to be used for the MISOCP formulation given in (MISOCP). From a computational standpoint, we only give the steps in the procedure for generating the cut here. Interested readers are referred to the related publication for details regarding intuition, motivation, and proofs.

Atamtürk and Narayanan [AN10] describe MIR cuts for a MISOCP given in the fol-

lowing form,

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & \|A^j x - b^j\| \leq d^{j\top} x - \gamma^j,\ j = 1, 2, \ldots, k \\ & x_i \in \mathbb{Z}_+,\ i \in I, \\ & x_i \in \mathbb{R}_+,\ i \in C, \end{aligned} \tag{3.1}$$

where $c \in \mathbb{Q}^n$ and $\gamma^j \in \mathbb{Q}, b^j \in \mathbb{Q}^k, d^j \in \mathbb{Q}^n, A^j \in \mathbb{Q}^{m_j \times n}$ for $j = 1, \ldots, k$. Number of variables is n, and I is the index set of integer variables. m_j is the number of rows in the j^{th} conic constraint. Note that conic constraints of formulation (3.1) are as in Definition 1.12, and has nonnegativity of variables in addition.

The generated inequalities are with respect to a relaxation that includes only one of the of the k conic constraints in (3.1) above. Let us then consider the j^{th} conic constraint for a fixed j of the above form, e.g.,

$$\|A^j x - b^j\| \leq d^{j\top} x - \gamma^j. \tag{3.2}$$

By introducing $(t_1, t_{2,m_j+1}) \in \mathbb{R} \times \mathbb{R}^{m_j}$, (3.2) can be rewritten as follows,

$$\begin{aligned} & t_1 \leq d^{j\top} x - \gamma^j, \\ & t_{l+1} \geq |A^j_l x - b^j_l| \qquad l = 1, \ldots, m_j, \\ & t_1 \geq \|t_{2,m_j+1}\|, \end{aligned} \tag{3.3}$$

where A^j_l is l^{th} row of A^j. We further relax the problem by considering only the l^{th} row in (3.3) for a fixed l. Let β represent b^j_l and vector $a \in \mathbb{Q}^n$ represent $A^{j\top}_l$. We define $\mathcal{S}^{l,j}_{\text{MIR}}$ as the set given by the l^{th} row of (3.3) and it can be represented as

$$\mathcal{S}^{l,j}_{\text{MIR}} := \{(x, t) \in \mathbb{R}^n_+ \times \mathbb{R} \mid t \geq |a^\top x - \beta|, x_i \in \mathbb{Z}\ \forall i \in I\}.$$

Note that $\mathcal{S}^{l,j}_{\text{MIR}}$ is second-order cone with dimension 2, i.e., $(t, a^\top x - \beta) \in \mathbb{L}^2$. At this point, we generate inequalities valid for set $\mathcal{S}^{l,j}_{\text{MIR}}$—these are the MIR cuts. To do so, we first define a piece-wise–linear function $\varphi_f : \mathbb{R} \to \mathbb{R}$ by,

$$\varphi_f(a) := \left\{ \begin{array}{ll} (1-2f)n - (a-n) & \text{if } n \leq a < n+f \\ (1-2f)n + (a-n) - 2f & \text{if } n+f \leq a < n+1 \end{array} \right\}.$$

Then, for any $\alpha \neq 0$, we have that

$$\sum_{i \in I} \varphi_{f_\alpha}\left(\frac{a_i}{\alpha}\right) x_i - \varphi_{f_\alpha}\left(\frac{\beta}{\alpha}\right) \leq \frac{(t + \sum_{i \in C} x_i)}{|\alpha|} \quad \forall x \in \mathcal{S}^{l,j}_{\text{MIR}}, \tag{MIR}$$

where $f_\alpha = \frac{\beta}{\alpha} - \lfloor \frac{\beta}{\alpha} \rfloor$. Moreover, if α is chosen such that $\alpha = a_r$ and $\frac{\beta}{a_r} > 0$ for some $r \in \{1, \ldots, n\}$ and $a_i \leq \beta$ for all $i \in \{1, \ldots, n\} \setminus \{r\}$, then the inequality is facet-defining for $conv(\mathcal{S}^{l,j}_{\text{MIR}})$. Note that the cuts generated are linear in the lifted space and nonlinear in the original formulation space.

We next describe a procedure that generates MIR cuts for the MISOCP formulation given in (MISOCP) but with nonnegativity constraints. It is important to note first that the MIR procedure fails to cut any region from continuous relaxation when applied naively. To see this, observe that second order conic constraints in Lorentz cone format for a fixed cone j is in the form

$$x^j \in \mathbb{L}^{n_j}$$

If we introduce variables $(t_1, t_{2:n_j}) \in \mathbb{R}^{n_j}$ and write the constraint as before, we obtain

$$\begin{aligned} & t_1 \leq x^j_1 \\ & t_l \geq |x^j_i|, \qquad i = 2, \ldots, n_j \\ & t_1 \geq \|t_{2:n_j}\|. \end{aligned} \tag{3.4}$$

Now we generate the inequality (MIR) using a single constraint from (3.4), also as before. Note that $\beta = 0$ in our case. This indicates f_α is 0 for any value of α. Then, it is trivial to observe that $\varphi_{f_\alpha} : \mathbb{R} \to \mathbb{R}$ is the identity function, i.e., $\varphi_{f_\alpha}(a) := a$. Thus, for i^{th} row of (3.4), the generated inequality is

$$\begin{aligned} \frac{x_i^j}{\alpha} &\leq \frac{t_i}{|\alpha|} \qquad \text{if} \quad i \in I \\ 0 &\leq \frac{t_i + x_i^j}{|\alpha|} \qquad \text{if} \quad i \in C \end{aligned} \tag{3.5}$$

The inequality obtained is indeed valid for the feasible set of (MISOCP). Even though the inequality is indeed valid, it cannot be useful, since it is also valid for the continuous relaxation and thus redundant. This is because of the fact that we start with a different formulation than the one given in Atamtürk and Narayanan [AN10].

To see how such inequalities should be generated, note that j^{th} Lorentz cone in (MISOCP) can be embedded in a lifted space by the reformulation

$$\begin{aligned} t &\geq |x_i^j| \\ x_1^j &\geq \|(x_2^j, \ldots, t, \ldots, x_{n_j}^j)\|. \end{aligned}$$

The formulation (MISOCP) contains linear constraints. Let us assume that a single linear constraint in (MISOCP) implies $x_i^j = a^\top x - \beta$. We then generate a valid inequality with respect to the following conic set,

$$\{(x,t) \in \mathbb{R}_+^{n+1} \mid t \geq \left| a^\top x - \beta \right|, x_i \in \mathbb{Z}\ \forall i \in I\}.$$

Once the cut is generated, in order to add it, we need to add variable t to the original

formulation by modifying the associated conic constraint to be

$$x_1^j \geq \|(x_2^j, \ldots, t, \ldots, x_n^j)\|.$$

Note that with this method we lift the model with just one more variable and add only one inequality. The problem size will increase with one more column and one more linear row. Note also that since a conic MIR inequality is a generalization of the MIR inequality in the MILP case every MIR inequality in the MILP case is also a conic MIR inequality.

3.2.2 Conic Gomory Cuts

Çezik and Iyengar [ÇI05] extend the well-known Gomory cuts for MILPs to linear, second-order and semidefinite conic problems using the following equivalence

$$\{x \in \mathbb{R}^n \mid Ax - b \in \mathcal{K}\} = \left\{x \in \mathbb{R}^n \mid \left(A^\top u\right)^\top x \geq u^\top b \; \forall u \in \mathcal{K}^*\right\},$$

where $\mathcal{K}$ is either a linear, Lorentz, or semidefinite cone. For the cones we consider, cone $\mathcal{K}$ is self-dual, so that $\mathcal{K}^* = \mathcal{K}$. Çezik and Iyengar [ÇI05] cover Gomory cuts for problems with binary variables only. Drewes [Dre09] extends the procedure to the mixed-integer case.

We summarize the procedure for generating Gomory cuts for the problem given in the form (MISOCP) with only 1 conic constraint and additional nonnegativity constraints on x. This feasible set and its continuous relaxation can be given as

$$\mathcal{S}_{\text{G-MISOCP}} := \left\{x \in \mathbb{R}^n_+ \mid Ax = b, x \in \mathbb{L}^n, x_i \in \mathbb{Z}_+ \; \forall i \in I\right\},$$
$$\mathcal{S}_{\text{G-SOCP}} := \left\{x \in \mathbb{R}^n_+ \mid Ax = b, x \in \mathbb{L}^n\right\}.$$

$\mathcal{S}_{\text{G}-\text{SOCP}}$ can be written as

$$\mathcal{S}_{\text{G}-\text{SOCP}} = \left\{ x \in \mathbb{R}^n_+ \mid \left(y^\top A_I + u_I^\top\right) x_I + \left(y^\top A_C + u_C^\top\right) x_C \geq y^\top b \; \forall y \in \mathbb{R}^m, \right.$$
$$\left. (u_I^\top, u_C^\top)^\top \in \mathbb{L}^n \right\}.$$

Then the projection of $\mathcal{S}_{\text{G}-\text{SOCP}}$ onto the integer variables is

$$\mathcal{S}^I_{\text{G}-\text{SOCP}} := \left\{ x_I \in \mathbb{R}^{n_I}_+ \mid \exists x_C \in \mathbb{R}^{n_C}_+ \mid \left(y^\top A_I + u_I^\top\right) x_I + \left(y^\top A_C + u_C^\top\right) x_C \geq y^\top b \; \forall y \in \mathbb{R}^m, \right.$$
$$\left. (u_I^\top, u_C^\top)^\top \in \mathbb{L}^n \right\},$$

where y and u are dual variables. Finally, $\mathcal{S}^I_{\text{G}-\text{SOCP}}$ can be rewritten as

$$\mathcal{S}^I_{\text{G}-\text{SOCP}} = \left\{ x_I \in \mathbb{R}^{n_I}_+ \mid \exists x_C \in \mathbb{R}^{n_C}_+ \mid \left(y^\top A_I + u_I^\top\right) x_I + \left(y^\top A_C + u_C^\top\right) x_C \geq y^\top b \; \forall y \in \mathbb{R}^m, \right.$$
$$\left. (u_I^\top, u_C^\top)^\top \in \mathbb{L}^n, \left(y^\top A_C + u_C^\top\right) \leq 0 \right\}.$$

Then we have that

$$\lceil A_I^\top y + u_I \rceil^\top x_I \geq \lceil y^\top b \rceil \;\; \forall x_I \in \mathcal{S}^I_{\text{G}-\text{SOCP}} \cap \mathbb{Z}^{n_I}_+. \qquad \text{(Gomory)}$$

Let $\overline{x}$ be a solution to optimization over $\mathcal{S}_{\text{G}-\text{SOCP}}$ and $\overline{x} \notin \mathcal{S}_{\text{G}-\text{MISOCP}}$ and let

$$\lceil A_I^\top y + u_I \rceil^\top \overline{x}_I = y^\top b$$

hold. Assume $y^\top b$ is not an integer. Then $\overline{x}$ is cut off by Gomory cut (Gomory) generated. Drewes [Dre09] summarizes this result in the following theorem.

Theorem 3.2.1. *([Dre09]) Assume the continuous relaxation and its dual have feasible interior points. Let $\overline{x} \notin \mathbb{Z}^n$ be a solution to the continuous relaxation and $(\overline{s}, \overline{y})$ the*

corresponding dual solution, then

$$\lceil A_I^\top (\overline{y} - \Delta_y) + \overline{s}_I \rceil^\top x_I \geq \lceil (\overline{y} - \Delta_y)^\top b \rceil,$$

where Δ_y *solves*

$$\begin{pmatrix} -A_C \\ A_I \end{pmatrix} \Delta_y = \begin{pmatrix} c_C \\ 0 \end{pmatrix},$$

is a valid inequality. Furthermore, if $(\overline{y} - \Delta_y)^\top b \notin \mathbb{Z}$*, this inequality separates* $\overline{x}$ *from the integer feasible set* $\mathcal{S}_{\text{G-MISOCP}}$*.*

3.2.3 Lift-and-Project Cuts for Mixed 0–1 Convex Sets

As was done with MIR inequalities, Stubbs and Mehrotra [SM99] generalized the traditional framework of the lift-and-project inequalities that are employed in the solution of MILPs to general convex optimization problems. These were later implemented and applied to the case of 0–1 SOCP by Drewes [Dre09]. In this section, we summarize how to generate these cuts. Interested readers are referred to Drewes [Dre09] for the details. The cuts are given for a special case of (MISOCP) where there is only one conic constraint and all integer variables are binary. This feasible set is denoted as $\mathcal{S}_{\text{SM-MISOCP}}$ and its continuous relaxation as $\mathcal{S}_{\text{SM-SOCP}}$. These sets can be given as

$$\mathcal{S}_{\text{SM-MISOCP}} := \{x \in \mathbb{R}^n \mid Ax = b, x \in \mathbb{L}^n, x_i \in \{0,1\}\ \forall i \in I\},$$
$$\mathcal{S}_{\text{SM-SOCP}} := \{x \in \mathbb{R}^n \mid Ax = b, x \in \mathbb{L}^n\}.$$

We impose a disjunction on $\mathcal{S}_{\text{SM}-\text{SOCP}}$ using a specific binary variable indexed by j and define the following two sets,

$$\mathcal{S}^{j,0}_{\text{SM}-\text{SOCP}} := \{x \in \mathcal{S}_{\text{SM}-\text{SOCP}} \mid x_j = 0\} \text{ and}$$
$$\mathcal{S}^{j,1}_{\text{SM}-\text{SOCP}} := \{x \in \mathcal{S}_{\text{SM}-\text{SOCP}} \mid x_j = 1\}.$$

We define set $\mathcal{S}^{j}_{\text{SM}-\text{SOCP}}$ as the union of $\mathcal{S}^{j,0}_{\text{SM}-\text{SOCP}}$ and $\mathcal{S}^{j,1}_{\text{SM}-\text{SOCP}}$, which can be written as

$$\mathcal{S}^{j}_{\text{SM}-\text{SOCP}} := \{x \in \mathcal{S}_{\text{SM}-\text{SOCP}} \mid x_j \in \{0,1\}\}.$$

We define set $\mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ to represent conv $\left(\mathcal{S}^{j}_{\text{SM}-\text{SOCP}}\right)$. Using auxiliary variables u and λ we can write $\mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ as

$$\mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}}) := \left\{ \begin{array}{ll} & \lambda^0 u^0 + \lambda^1 u^1 = x, \\ & \lambda^0 + \lambda^1 = 1, \lambda^0, \lambda^1 \geq 0 \\ & Au^0 = b \\ & Au^1 = b \\ (x, u^0, u^1, \lambda^0, \lambda^1) \in \mathbb{R}^{3n+2} \mid & u^0 \in \mathbb{L}^n \\ & u^1 \in \mathbb{L}^n \\ & (u^0)_i \in [0,1] (i \in I, i \neq j) \\ & (u^1)_i \in [0,1] (i \in I, i \neq j) \\ & (u^0)_j = 0, (u^1)_j = 1 \end{array} \right\}.$$

Note that u^0 variables corresponds to feasible points in set $\mathcal{S}^{j,0}_{\text{SM}-\text{SOCP}}$ and u^1 corresponds to $\mathcal{S}^{j,0}_{\text{SM}-\text{SOCP}}$. λ^0 and λ^1 are used to take their convex combination. $\mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ can be rewritten so that it is in SOCP form given in introduction. For this we introduce

$v^0 := \lambda^0 u^0$ and $v^1 := \lambda^1 u^1$ and define

$$\tilde{\mathcal{M}}_j(\mathcal{S}_{\text{SM}-\text{SOCP}}) := \left\{ (x, v^0, v^1, \lambda^0, \lambda^1) \in \mathbb{R}^{3n+2} \mid \begin{array}{l} v^0 + v^1 = x, \\ \lambda^0 + \lambda^1 = 1, \lambda^0, \lambda^1 \geq 0 \\ Av^0 - \lambda^0 b = 0 \\ Av^1 - \lambda^1 b = 0 \\ v^0 \succeq_{\mathbb{K}} 0 \\ v^1 \succeq_{\mathbb{K}} 0 \\ (v^0)_i \in [0, \lambda^0] (i \in I, i \neq j) \\ (v^1)_i \in [0, \lambda^1] (i \in I, i \neq j) \\ (v^0)_j = 0, (v^1)_j = 1 \end{array} \right\}.$$

Note that $\tilde{\mathcal{M}}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ is a second-order conic representable set. We define $\mathcal{P}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ as the projection of $\mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ to x–space,

$$\mathcal{P}_j(\mathcal{S}_{\text{SM}-\text{SOCP}}) := \{x \in \mathbb{R}^n \mid (x, u^0, u^1, \lambda^0, \lambda^1) \in \mathcal{M}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})\}.$$

Note that $\mathcal{P}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})$ can also be written as,

$$\mathcal{P}_j(\mathcal{S}_{\text{SM}-\text{SOCP}}) = \{x \in \mathbb{R}^n \mid (x, v^0, v^1, \lambda^0, \lambda^1) \in \tilde{\mathcal{M}}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})\}.$$

Given a continuous relaxation solution $\overline{x}$ where $\overline{x}_j \in (0, 1)$, we can generate a cut by solving the following problem,

$$\min_{x \in \mathcal{P}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})} \|x - \overline{x}\|.$$

This problem is equivalent to the following problem,

$$\min_{w \in \tilde{\mathcal{M}}_j(\mathcal{S}_{\text{SM}-\text{SOCP}})} \|x - \overline{x}\|. \tag{3.6}$$

where $w = (x, v^0, v^1, \lambda^0, \lambda^1)$. Note that the problem given in (3.6) is an SOCP. Let $\hat{x}$ be the optimal solution to (3.6), then the following constraint cuts $\overline{x}$,

$$(\hat{x} - \overline{x})^\top x \geq \hat{x}^\top (\hat{x} - \overline{x}).$$

3.2.4 DCC and DCyC

Belotti et al. [Bel+13] generate valid inequalities for convex hulls of disjunctions on (SOCP). They consider disjunctions generated by both parallel and non-parallel hyperplanes. Here, we discuss their results for a special case of (MISOCP) with only one cone. Feasible set and its continuous relaxation can be given as

$$\mathcal{S}_{\text{DCC}-\text{MISOCP}} := \{x \in \mathbb{R}^n \mid Ax = b, x \in \mathbb{L}^n, x_i \in \mathbb{Z}\ \forall i \in I\},$$
$$\mathcal{S}_{\text{DCC}-\text{SOCP}} := \{x \in \mathbb{R}^n \mid Ax = b, x \in \mathbb{L}^n\}.$$

Using the technique we introduced in Section 1.2, $\mathcal{S}_{\text{DCC}-\text{SOCP}}$ can be reformulated as

$$\mathcal{S}_{\text{DCC}-\text{SOCP}} = \{x \in \mathbb{R}^n \mid x = x^0 + Hw, w \in \mathcal{Q}, x_1 \geq 0\},$$
$$\mathcal{Q} = \{w \in \mathbb{R}^{n-m} \mid (x^0 + Hw)^\top J (x^0 + Hw) \leq 0\}.$$

where J, H, and x^0 are as defined in Section 1.2. Note that quadric $\mathcal{Q}$ does not impose nonnegativity on x_1 and hence $x^0 + Hw \in \mathbb{L}^n \cup -\mathbb{L}^n$. Each $w \in \mathbb{R}^{n-m}$ such that $(x^0 + Hw)_1 \geq 0$ represents a linear combination of null space basis (given by columns of H), hence it corresponds to an $x \in \mathcal{S}_{\text{DCC}-\text{SOCP}}$.

For a given $\overline{x} \in \mathcal{S}_{\text{DCC}-\text{SOCP}}$ there exists a unique $\overline{w} \in \mathcal{Q}$ such that $H\overline{w} = \overline{x} - x^0$. Since

H is orthonormal, we have that $H^\top H = I$ and thus

$$H\overline{w} = \overline{x} - x^0 \Leftrightarrow H^\top H\overline{w} = H^\top\left(\overline{x} - x^0\right) \Leftrightarrow \overline{w} = H^\top\left(\overline{x} - x^0\right).$$

Note that not every $w \in \mathcal{Q}$ corresponds to an $x \in \mathcal{S}_{\text{DCC}-\text{SOCP}}$, since we might have $x_1 = (x^0 + Hw)_1 < 0$. For any $\overline{w}$ such that $(x^0 + H\overline{w})_1 \geq 0$, however, we have

$$\overline{x} = x^0 + H\overline{w} \in \mathcal{S}_{\text{DCC}-\text{SOCP}}. \tag{3.7}$$

This analysis suggests that it might be possible to solve the problem of separating a given $\hat{x} \in \mathbb{R}^n$ from $\mathcal{S}_{\text{DCC}-\text{SOCP}}$ by instead separating the corresponding $\hat{w}$ from $\mathcal{Q}$.

Let $\mathcal{U}$ and $\mathcal{V}$ be two half spaces defined as

$$\mathcal{U} := \{x \in \mathbb{R}^n \mid u^\top x \geq \varphi\},$$
$$\mathcal{V} := \{x \in \mathbb{R}^n \mid v^\top x \leq \omega\}.$$

The set $\text{conv}\left(\mathcal{S}_{\text{DCC}-\text{SOCP}} \cap (\mathcal{U} \cup \mathcal{V})\right)$ is a convex set that cannot contain any extremal element of $\mathcal{S}_{\text{DCC}-\text{SOCP}}$ that is not contained in $\mathcal{U} \cup \mathcal{V}$, i.e., an inequality valid for $\text{conv}\left(\mathcal{S}_{\text{DCC}-\text{SOCP}} \cap (\mathcal{U} \cup \mathcal{V})\right)$ should be violated by any extremal solution of $\mathcal{S}_{\text{DCC}-\text{SOCP}}$.

We now show how to represent $\mathcal{U}$ and $\mathcal{V}$ in w-space. As before, with each $w \in \mathbb{R}^n$, we can associate a unique $x = x^0 + Hw$. Then we have that

$$\mathcal{U} = \{x \in \mathbb{R}^n \mid \exists w \in \mathbb{R}^{n-m}, x = x^0 + Hw, u^\top Hw \geq \varphi - u^\top x^0\}$$
$$\mathcal{V} = \{x \in \mathbb{R}^n \mid \exists w \in \mathbb{R}^{n-m}, x = x^0 + Hw, v^\top Hw \leq \omega - v^\top x^0\}$$

Defining $a^\top := u^\top H$, $\alpha := \varphi - u^\top x^0$, $d^\top := v^\top H$ and $\beta := \omega - v^\top x^0$, we can reformulate

$\mathcal{U}$ and $\mathcal{V}$ in w–space as

$$\mathcal{A} := \left\{ w \in \mathbb{R}^{n-m} \mid a^\top w \geq \alpha \right\},$$
$$\mathcal{B} := \left\{ w \in \mathbb{R}^{n-m} \mid d^\top w \leq \beta \right\}.$$

In other words, sets $\mathcal{A}$ and $\mathcal{B}$ are half-spaces consisting of all $w \in \mathbb{R}^n$ that map to points in $\mathcal{U}$ and $\mathcal{V}$, respectively. Using $\mathcal{A}$ and $\mathcal{B}$, we can further reformulate $\text{conv}\left(\mathcal{S}_{\text{DCC}-\text{SOCP}} \cap (\mathcal{U} \cup \mathcal{V})\right)$ as

$$\mathcal{S}_{\text{DCC}-\text{SOCP}} \cap (\mathcal{U} \cup \mathcal{V}) = \{x \in \mathbb{R}^n : x = x^0 + Hw, w \in \mathcal{Q}, w \in \mathcal{A} \cup \mathcal{B}, x_1 \geq 0\}.$$

We obtain a relaxation of this set by removing the constraint $x_1 \geq 0$ to get

$$\mathcal{S}^{\text{relax}}_{\text{DCC}-\text{SOCP}} \cap (U \cup V) = \{x \in \mathbb{R}^n : x = x^0 + Hw, w \in \mathcal{Q}, w \in \mathcal{A} \cup \mathcal{B}\}.$$

If we let quadric $\mathcal{Q}$ be given by the triplet (Q, q, ρ), then Belotti et al. [Bel+13] show that the family of quadrics that yields

$$Q(\tau) = Q + \tau \frac{ad^\top + da^\top}{2},$$
$$q(\tau) = q - \tau \frac{\beta a + \alpha d}{2},$$
$$\rho(\tau) = \rho + \tau \alpha \beta,$$

where $\tau \in \mathbb{R}$. We denote this parametric quadric family as $\mathcal{Q}(\tau)$.

The shape of $\mathcal{Q}(\tau)$ depends on the position of $q(\tau)^\top Q(\tau)^{-1} q(\tau) - \rho(\tau)$ with respect to 0 in the real line. Belotti et al. [Bel+13] prove a closed form formula for this term as follows,

$$q(\tau)^\top Q(\tau)^{-1} q(\tau) - \rho(\tau) = \frac{f(\tau)}{g(\tau)}, \tag{3.8}$$

where $f(\tau)$ and $g(\tau)$ are given as

$$\begin{aligned} f(\tau) := \tau^2 &\Big[-\|u_a\|^2(\beta + u_d^\top u_q)^2 - \|u_d\|^2(\alpha + u_a^\top u_q)^2 \\ &+ \left(\|u_a\|^2\|u_d\|^2 - (u_a^\top u_d)^2\right)(\|u_q\|^2 - \rho) + 2u_a^\top u_d(u_a^\top u_q + \alpha)(u_d^\top u_q + \beta)\Big] \\ &+ 4\tau\left[-u_a^\top u_d(\|u_q\|^2 - \rho) + (\alpha + u_a^\top u_q)(\beta + u_d^\top u_q)\right] - 4\left[\|u_q\|^2 - \rho\right], \\ g(\tau) := \tau^2 &\left(\|u_a\|^2\|u_d\|^2 - \left(u_a^\top u_d\right)^2\right) - 4\tau u_a^\top u_d - 4. \end{aligned}$$

For a given vector a, u_a is defined as

$$u_a := Q^{\frac{-1}{2}} a.$$

Belotti et al. [Bel+13] show that for a given τ the shape of $\mathcal{Q}(\tau)$ depends on the τ's position with respect to roots of f and g on the real line. Let $\overline{\tau}_1$ and $\overline{\tau}_2$ denote the roots of f and $\hat{\tau}_1$ and $\hat{\tau}_2$ denote the roots of g. Following theorem from Belotti et al. [Bel+13] gives the possible shapes of $\mathcal{Q}(\tau)$.

Theorem 3.2.2. *([Bel+13]) Quadric $\mathcal{Q}(\tau)$ can have the following shapes for $\tau \in \mathbb{R}$:*

- *If $\overline{\tau}_1$ and $\overline{\tau}_2$ are distinct and different than $\hat{\tau}_1$ and $\hat{\tau}_2$, then $\mathcal{Q}(\hat{\tau}_1)$ and $\mathcal{Q}(\hat{\tau}_2)$ are paraboloids, and $\mathcal{Q}(\overline{\tau}_1)$ and $\mathcal{Q}(\overline{\tau}_2)$ are cones.*
- *If $\overline{\tau}_1$ and $\overline{\tau}_2$ are distinct and one of them coincides with either $\mathcal{Q}(\hat{\tau}_1)$ or $\mathcal{Q}(\hat{\tau}_2)$, then either $\mathcal{Q}(\overline{\tau}_1)$ is a cylinder and $\mathcal{Q}(\overline{\tau}_2)$ is a cone, or that $\mathcal{Q}(\overline{\tau}_1)$ is a cone and $\mathcal{Q}(\overline{\tau}_2)$ is a cylinder.*
- *If $\overline{\tau}_1$ and $\overline{\tau}_2$ are distinct and one of them is same as $\hat{\tau}_1$ and the other is same as $\hat{\tau}_2$, then both $\mathcal{Q}(\hat{\tau}_1)$ and $\mathcal{Q}(\hat{\tau}_2)$ are cylinders.*
- *$\overline{\tau}_1 = \overline{\tau}_2$ but are distinct from $\hat{\tau}_1$ and $\hat{\tau}_2$, then $\mathcal{Q}(\overline{\tau}_1)$ is a cone and $\mathcal{Q}(\hat{\tau}_1)$ and $\mathcal{Q}(\hat{\tau}_2)$ are paraboloids.*

- *$\overline{\tau}_1$ and $\overline{\tau}_2$ coincide and equal to either $\hat{\tau}_1$ or $\hat{\tau}_2$. Then, either $\mathcal{Q}(\hat{\tau}_1)$ is a line and $\mathcal{Q}(\hat{\tau}_2)$ is a paraboloid, or $\mathcal{Q}(\hat{\tau}_1)$ is a paraboloid and $\mathcal{Q}(\hat{\tau}_2)$ is a line.*

Theorem 3.2.2 shows that there is always a τ value such that $\mathcal{Q}(\tau)$ is second-order conic representable. We choose τ accordingly to obtain a second-order conic representable cut.

Note that this is a conic cut in w space and the original formulation is in x space. This cut should be represented in x space to be able to use it in the original formulation. This is achieved using the equality $x = x^0 + Hw$ and inserting $H^\top(x - x^0)$ for w to obtain the cone in x space. After this transformation a cone is in the x space in quadric form is obtained. Belotti et al. [Bel+13] use singular value decomposition to represent it in Lorentz cone format with additional linear constraints. We refer reader to Belotti et al. [Bel+13] for details.

3.2.5 Two-Term Disjunctions on Lorentz Cone

This section summarizes the disjunctive cuts given by Kılınç-Karzan and Yıldız [KY14]. They generate representations of convex hull of a general disjunction on a Lorentz cone. Kılınç-Karzan and Yıldız [KY14] argue that considering Lorentz cone is enough and can be generalized to disjunctions on more general cones in dual form using affine transformation given by Andersen and Jensen [AJ13].

Kılınç-Karzan and Yıldız [KY14] come up with representations, not necessarily conic, of disjunctions on $\mathbb{L}^n$. The disjunction is defined by the following two disjunctive sets,

$$\mathcal{C}_1 := \{x \in \mathbb{L}^n : c_1^\top x \geq c_{1,0}\},$$
$$\mathcal{C}_2 := \{x \in \mathbb{L}^n : c_2^\top x \geq c_{2,0}\},$$

where disjunctive inequalities are scaled such that $c_{1,0}, c_{2,0} \in \{0, \pm 1\}$. Kılınç-Karzan and

Yıldız [KY14] make the following two assumptions related to the disjunctions,

Assumption 3.2.3. $\mathcal{C}_1 \not\subseteq \mathcal{C}_2$ *and* $\mathcal{C}_2 \not\subseteq \mathcal{C}_1$.

Assumption 3.2.4. $\mathcal{C}_1 \cap int\,\mathbb{L}^n \neq \emptyset$ *and* $\mathcal{C}_2 \cap int\,\mathbb{L}^n \neq \emptyset$.

From duality theory, a linear inequality $\mu^\top x \geq \mu_0$ is valid for $\overline{\text{conv}}(\mathcal{C}_1 \cup \mathcal{C}_2)$ if and only if there exist $\alpha_1, \alpha_2, \beta_1, \beta_2$ such that $(\mu, \mu_0, \alpha_1, \alpha_2, \beta_1, \beta_2)$ satisfies the following,

$$\mu = \alpha_1 + \beta_1 c_1$$

$$\mu = \alpha_2 + \beta_2 c_2$$

$$\beta_1 c_{1,0} \geq \mu_0$$

$$\beta_2 c_{2,0} \geq \mu_0$$

$$\alpha_1, \alpha_2 \in \mathbb{L}^n$$

$$\beta_1, \beta_2 \in \mathbb{R}_+.$$

Some of the inequalities specified by the given system are dominated by the others. They further improve the system of inequalities to include undominated ones only, we will skip that step here. Finally, they give the following inequality that implies all non-dominated valid inequalities (for $\beta_1 = \beta$, $\beta_2 = 1$) specified by the given system, for any $\beta > 0$ such that $\beta c_{1,0} \geq c_{2,0}$ and $\beta c_1 - c_2 \notin \mathbb{L}^n$. We have that

$$2c_{2,0} - (\beta c_1 + c_2)^\top x \leq \sqrt{((\beta c_1 - c_2)^\top x)^2 + N_1(\beta)\left(x_1^2 - \|x_{2:n}\|^2\right)}\ \forall x \in \overline{\text{conv}}\,(\mathcal{C}_1 \cup \mathcal{C}_2) \tag{3.9}$$

where

$$N_1(\beta) := \|\beta c_{1,2:n} - c_{2,2:n}\|_2^2 - (\beta c_{1,1} - c_{2,1})^2.$$

The set of points that are members of $\mathbb{L}^n$ and that satisfy the inequality given in (3.9)

can be written as

$$\left\{x \in \mathbb{L}^n \mid 2c_{2,0} - (\beta c_1 + c_2)^\top x \leq \sqrt{((\beta c_1 - c_2)^\top x)^2 + N_1(\beta)\left(x_1^2 - \|x_{2:n}\|^2\right)}\right\}, \quad (3.10)$$

and it is convex. Inequality given in (3.9) can be cast in conic dual form as

$$N_1(\beta)x + 2(c_2^\top x - c_{2,0}) \begin{pmatrix} \beta c_{1,2:n} - c_{2,2:n} \\ -\beta c_{1,1} + c_{2,1} \end{pmatrix} \in \mathbb{L}^n \quad (3.11)$$

for a β that satisfies the following,

$$\beta > 0, \quad (3.12)$$

$$\beta c_{1,0} \geq c_{2,0} \quad (3.13)$$

$$\beta c_1 - c_2 \notin \pm\mathbb{L}^n \quad (3.14)$$

$$-2c_{2,0} + (\beta c_1 + c_2)^\top x \leq \sqrt{((\beta c_1 - c_2)^\top x)^2 + N_1(\beta)\left(x_1^2 - \|x_{2:n}\|^2\right)} \quad \forall x \in \overline{conv}\left(\mathcal{C}_1 \cup \mathcal{C}_2\right). \quad (3.15)$$

Note that (3.9) is conic representable if such β exists. (3.11) is valid for $\overline{\text{conv}}\left(\mathcal{C}_1 \cup \mathcal{C}_2\right)$ but that does not mean it gives convex hull exactly. The valid inequality in conic dual form given in (3.11) is equivalent to nonlinear cut (3.9) if the following condition holds.

$$\{x \in \mathbb{L}^n : \beta c_1^\top x > c_{2,0}, c_2^\top x > c_{2,0}\} = \emptyset.$$

Kılınç-Karzan and Yıldız [KY14] give the following theorem which describes an inequality that gives the convex hull of the disjunction.

Theorem 3.2.5. *Assume $c_1 - c_2 \notin \pm\mathbb{L}^n$, then the inequality*

$$2c_{2,0} - (c_1 + c_2)^\top x \leq \sqrt{((c_1 - c_2)^\top x)^2 + N\left(x_1^2 - \|x_{2:n}\|^2\right)} \quad (3.16)$$

is valid for $\overline{conv}(\mathcal{C}_1 \cup \mathcal{C}_2)$ *with* $N := \|c_{1:2:n} - c_{2,2:n}\|_2^2 - (c_{1,1} - c_{2,1})^2$. *Furthermore,*

$$\overline{conv}(\mathcal{C}_1 \cup \mathcal{C}_2) = \{x \in \mathbb{L}^n : x \text{ satisfies } (3.16)\}$$

when, in addition, we have

(i) $c_1 \in \mathbb{L}^n$, *or* $c_2 \in \mathbb{L}^n$, *or*

(ii) $c_{1,0} = c_{2,0} \in \{\pm 1\}$ *and undominated valid linear inequalities that are tight on both* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *are sufficient to describe* $\overline{conv}(\mathcal{C}_1 \cup \mathcal{C}_2)$.

Kılınç-Karzan and Yıldız [KY14] study disjunctions on a Lorentz cone, where Belotti et al. [Bel+15] study disjunctions on a standard form SOCP problem with a single cone. Kılınç-Karzan and Yıldız [KY14] give convex inequalities that are valid for the disjunctions. Inequalities generated are conic representable under some other conditions that they do not discuss how to validate. The conic cuts given by Kılınç-Karzan and Yıldız [KY14] do not yield the convex hull of the disjunction. Kılınç-Karzan and Yıldız [KY14] do come up with an inequality (Theorem 3.2.5) that gives convex hull but it is not second-order conic representable.

On the other hand, Belotti et al. [Bel+15] start with more restrictive assumptions ($\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$ and sets $\{x \in \mathbb{L}^n : c_1^\top x = c_{1,0}\}$ and $\{x \in \mathbb{L}^n : c_2^\top x = c_{2,0}\}$ are bounded) and work with disjunctions on standard form feasible sets rather than Lorentz cone. The cuts given by Belotti et al. [Bel+15] are second-order conic representable and give the convex hull of the disjunction.

An interesting future research question would be investigating whether the conditions where Kılınç-Karzan and Yıldız [KY14] yield conic cuts and starting assumptions of Belotti et al. [Bel+15] are related.

3.3 A Branch-and-Cut Algorithm

This section introduces an algorithmic framework for solution of MISOCP problems. This framework is flexible and accommodates a wide variety of options for controlling the algorithm execution. These options include algorithmic components such as choosing the subproblem relaxation type, the bounding algorithm to be used to solve the subproblem relaxations, the branching strategy, the search strategy, and the strategies for both generating and managing the valid constraints (i.e., cuts) used to strengthen the relaxations. Depending on the choices made for each component of the algorithm and the overall control strategies, a wide variety of branch-and-cut algorithms can be obtained. The goal of this flexibility is to discover what the most effective choices are by comparing various algorithmic approaches in a controlled environment.

Chapter 4 describes the implementation of the framework introduced here in a software package called *DisCO*. Chapter 6 further explores the computational results obtained with this framework and provide insights on the effect of various choices.

3.3.1 Relaxation and Bounding

Choosing the proper relaxation for bounding is an important component of a branch-and-bound/cut algorithm. In MISOCP there are three types of constraints, linear, conic and integrality. In this section, we describe the two possible choices of relaxation: relaxing both integrality and conic constraints initially (yielding an LP) and just relaxing integrality constraints initially (yielding an SOCP). In both cases, the resulting relaxation can be strengthened with the addition of the valid inequalities from the previous section.

Relaxing both conic and integrality constraints results in a relaxation that is an LP. LPs are well-studied and efficient algorithms are known. Moreover branch-and-bound methods that employ LP relaxations are one of the most studied algorithms in the optimization literature. The simplex method is a natural choice in LP-based branch and bound, since it

allows for efficient warm-starting after imposing valid inequalities or branching constraints. In rare cases, the IPM may be a more effective choice and has the advantage that it can be parallelized. However, the increased effectiveness must be traded off against the loss of effective warm-starting capability.

Note that when LP relaxations are employed, the role of branching is to impose the relaxed integer feasibility, while conic feasibility can only be restored by the addition of the cuts from Theorem 2.2.1. A crucial component in this case is balancing these two objectives. An important question that is answered by this study is how LP relaxations compare to SOCPs as the relaxation of choice.

Relaxations that preserve the conic constraints are more difficult computationally. In the presence of conic constraints, the relaxations become SOCPs, which can be solved in polynomial time, just like LP. With SOCP relaxation, an SOCP-based branch-and-bound that follows the classical outline of LP-based branch-and-bound for MILP is obtained. There are two different choices of algorithm for solving the relaxation: IPM and the cutting-plane algorithm we introduced in Section 2.2.2. IPM is faster than the cutting-plane algorithm for solving a single SOCP, but as in the LP case, there is also a trade-off involving warm-starting. With IPM, only a limited warm-start capability is available, whereas with the cutting-plane method, we employ polyhedral approximations and retain warm-start capability. Table 3.1 presents the subproblem relaxation and algorithm options.

In what follows, we propose an algorithmic framework in which these two relaxations can be used interchangeably and in which a hybrid strategy in which conic constraints are relaxed initially, but conic approximations are built up using an approach similar to that described in Chapter 2, but in which the process can be terminated at any time in favor of branching. Valid inequalities of other classes, such as the many know classes valid for MILPs, can also be dynamically integrated according to parameters to be described later.

Table 3.1: Subproblem Relaxation and Algorithm Choices

Relaxation choices	Algorithm options
SOCP	IPM
SOCP	Algorithm 2.2.2
LP	IPM
LP	simplex

From the perspective of bounding, there are various tradeoffs among the approaches discussed. The SOCP relaxations are stronger and can be solved efficiently using IPM. However, warm-starting is limited. State of art warm-starting approaches can save 30–75% computation time compared to cold start [SAY13]. The cutting-plane algorithm is slow for solving SOCP but has the potential of warm-starting when used in the context of a branch-and-bound algorithm. A research question we investigate is exactly what the trade-off is between the cutting-plane algorithm, which is slower but has a much better warm start capability, and IPM, which is faster from a cold start, but has poorer warm-starting capability. When SOCP is the relaxation of choice, which algorithm performs best?

3.3.2 Generation of Valid Constraints

The addition of valid inequalities to the LP relaxation for the dual purpose of improving the bound obtained by its solution and removing previously generated infeasible solutions is a well-established technique in mixed integer linear programming. The additional constraints are generated by solving the problem of separating a given solution to the relaxation from the feasible set of the original problem. Iteratively strengthening the relaxation by solving a sequence of separation problems can be seen as a truncated version of the cutting-plane algorithm described in Algorithm 2.

In the case of MILP, the valid inequalities generated in the course of the branch-and-cut algorithm remove solutions to the relaxation that violate the integrality condition.

In practice, generation of valid inequalities has proven to be an extremely important part of solution algorithms and numerous procedures have been suggested in the literature [Gom58; Bal79; Bal+96; Bix+00]. Valid inequalities approximate (or improve the approximation of) the convex hull of feasible solutions of the MILP and improve the continuous relaxations of it.

In the case of MISOCP solved with an LP relaxation-based algorithm, the procedure for generating valid inequalities must also consider solutions that violate the conic constraints and as a result, we must generate valid inequalities to enforce conic feasibility as well. To ensure that the relaxations remain linear, the valid inequalities generated and added to the relaxations must be linear. For this purpose, we generate valid inequalities using the results of Theorems 2.2.1 and 2.2.4. Research on other types of inequalities and constraints for MISOCP, such as those mentioned in Section 3.2, is relatively new and is currently an active field [Bel+15; ÇI05; KY14; MKV15]. Studies involving the generation of valid inequalities for more general convex problems [SM99] have also appeared.

In the proposed algorithm, both valid conic constraints and valid inequalities are generated to improve the subproblem relaxations and to obtain better bounds. The conditions under which these additional constraints get generated and added can be controlled by specifying the parameters. The classes of valid constraints used in the algorithm are

- disjunctive conic cuts,
- linear cuts to approximate conic constraints, and
- cuts from MILP literature.

Disjunctive conic cuts can be used for any choice of subproblem relaxation, whereas the other two are used only in LP relaxation case. Approximation cuts are not needed in the case of SOCP relaxations, since the relaxation solutions will satisfy conic constraints automatically. Generation of MILP cuts is not an option when relaxations are SOCP.

Generating Disjunctive Conic Constraints

The proposed algorithm provides the option of generating disjunctive constraints in the root node to improve the initial bound. We use the disjunctive conic cuts given by Belotti et al. [Bel+15] as described in Section 3.2.4. As we mentioned above, disjunctive conic cuts can be generated for both of the subproblem relaxation choices. In LP relaxation case, they are treated same as original conic constraints and outer approximation inequalities are generated using them.

Disjunctive conic constraints are different than outer approximation cuts in terms of the computational effort required to compute them and computational burden of solution of the relaxations resulting from adding them. Typically a linear cut is a single linear constraint added to the subproblem relaxation. A disjunctive conic constraint, when in conic dual form, is of the dimension of the starting conic constraint, when represented in dual form.

For a conic constraint given in the conic dual form, the generated disjunctive conic constraint will be the same size of the given constraint. Addition of this newly generated constraint is a large increase in the size of the problem if the problem has a low number of cones. For a problem with a single large cone, it means doubling the number of conic constraints. Nevertheless, disjunctive conic constraints improve the relaxation more than linear valid inequalities. Linear cuts are half spaces that improve the relaxation in some specific part of the feasible region. Conic cuts are cones and they might improve the relaxation not in a specific corner but all around the feasible region. See [Bel+13] and [Bel+15] for figures that demonstrate this. In summary, conic cuts are computationally expensive to generate and use, but improve the relaxation to a greater extent than linear cuts. This indicates that decisions regarding to generation and use of conic cuts should be made with care. There will be consequences. Computational experiments that investigate this can be found in Section 6.5.6 and 6.5.7.

Generating Approximation Cuts

As we have already mentioned, when LP relaxations are used, the solutions produced are not necessarily conic feasible. The valid inequalities given in Theorem 2.2.1 and 2.2.4 are used to remove these infeasible points from the feasible region of the relaxation. The procedure is very simple, since these inequalities are obtained by a closed-form function of the given infeasible solution.

Generating MILP cuts

When LP relaxations are employed, there is an opportunity to use cuts from the MILP literature to cut integer infeasible solutions. Since the feasible regions of the relaxations are polyhedra, we can consider valid inequalities with respect to the convex hull of integer points inside these approximating polyhedra. This convex hull is itself a relaxation of the original MISOCP, which means that cuts valid for it are also valid for the MISOCP.

While approximation cuts remove conic infeasible solutions, MILP cuts remove integer infeasible solutions. Among the research questions we investigate later in Sections 6.5.5 and 6.5.3 are the following. Is removing integer infeasible solutions from the MILP relaxation worth the effort? What cuts should be generated when a relaxation solution is both integer and conic infeasible? These questions are answered through computational experiments in the pointed sections. To generate inequalities valid for the MILP relaxations, COIN-OR's cut generation library is used. Details are discussed in Chapter 4.

3.3.3 Branching

In our algorithm, variable disjunctions are the only disjunctions used for branching. We use two different branching strategies for selecting the branching disjunction from among those violated by a given solution to the relaxation: (1) strong branching [App+95] and (2) pseudocost branching [Bén+71; LS99].

In strong branching, in case of LP relaxations, dual simplex algorithm is used for its warm starting capability and relaxations are not solved to optimality for efficiency reasons. Dual simplex iterations are limited to 50. For each variable, a score is computed by combining the bound improvement of the two children nodes. The variable with the largest score is used for creating the disjunction.

In case of SOCP based relaxations, relaxations are solved to optimality for each disjunction variable candidate and the best in terms of bound improvement is picked.

Compared to the pseudocost branching, strong branching is costly. Pseudocost branching uses some statistics from the past branching decisions to estimate the bound improvement of a variable without performing costly simplex iterations. We implement pseudocost branching as described by Achterberg [Ach07]. For each integer variable, we keep two statistics, average amount of change in the objective function value with respect to unit change in down and up directions. Let φ_j^- and φ_j^+ be the average objective value change for unit change in x_j for all the past branching of j^{th} variable in down and up directions respectively. Then a score s_j for variable x_j is computed as

$$\begin{aligned} s_j =& (1-\mu) \times \min\left\{ (\overline{x}_j - \lfloor \overline{x}_j \rfloor)\, \varphi_j^-, (\lceil \overline{x}_j \rceil - \overline{x}_j)\, \varphi_j^+ \right\} \\ &+ \mu \times \max\left\{ (\overline{x}_j - \lfloor \overline{x}_j \rfloor)\, \varphi_j^-, (\lceil \overline{x}_j \rceil - \overline{x}_j)\, \varphi_j^+ \right\}. \end{aligned}$$

μ is a parameter of the branching strategy that is between 0 and 1. There are various suggestions for the value of μ in the literature. We follow Achterberg [Ach07] and use $\mu := \frac{1}{6}$. Pseudocost branching picks the variable with the highest score to branch.

When branch-and-cut algorithm is solved in parallel mode, pseudocost statistics are not shared by different processes. Each process solves subtree assigned to it and keeps its own pseudocost statistics for the assigned subtree.

3.3.4 Search Strategy

We use *hybrid search* strategy given by Xu [Xu07] in the computational experiments of the branch-and-cut algorithm. The search strategy part is implemented by the framework used to implement the proposed branch-and-cut algorithm. Hybrid search is the default search strategy in this framework. Implementational details of the branch-and-cut algorithm is given in Chapter 4. Implementational details of the search strategy is given by Xu et al. [Xu+05] and Xu [Xu07].

3.3.5 Cut Strategies

One of the important decisions in a branch-and-cut algorithm is when to generate valid inequalities. Valid inequalities might improve relaxations but requires computational effort to generate and also increase the size of the relaxation. In the case of MILP, two strategies that are tested and commonly used are adding cuts at the root node only and adding periodically for every fixed number of nodes.

In this study, we test adding disjunctive conic constraints in the root node. Variable disjunctions are used to generate disjunctive conic constraints as explained in Section 3.2.4. A variable should be decided to use for generating the disjunction. We test two different approaches (1) generating all possible disjunctive conic constraints and adding them as long as they are numerically robust. The second approach is generating every possible disjunctive conic constraint and selecting the best according to the criteria of bound improvement. Note that in this approach, we solve the SOCP relaxation after adding each disjunctive conic cut. At the end we pick the most bound improving and ignore the others. The first approach is better at improving the relaxation, but increases the problem size. The second approach improves the relaxations less then the first, but relaxations are relatively smaller.

Note that once we generate disjunctive constraints in the root node, we add them to

our initial relaxation. These cuts remain in all of the relaxations in the branch-and-bound tree.

For LP based relaxations we have the option to add MILP cuts (Gomory, MIR, etc.). We add MILP cuts periodically for every fixed number of nodes when the LP solutions are both conic and integer infeasible. The cuts are added as long as they improve the LP relaxation during bounding loop.

We do not accept all MILP cuts produced by our cut generators. We reject cuts that have a (1) bad scaling, (2) almost parallel to existing constraints or cuts already in the relaxation or (3) dense.

3.3.6 Heuristics

We use a rounding heuristic to search for solutions. The heuristic used is an implementation of the rounding heuristic given in Nemhauser and Wolsey [NW88]. The implementation is based on the heuristic search provided in COIN-OR's BLIS solver. When SOCP relaxations are used, heuristic is run ignoring conic constraints. After an integer solution satisfying linear constraints is found, conic feasibility is checked.

Running time of rounding heuristic is $O(\max\{\tilde{n}m, nk\})$ where n is the number of variables, $\tilde{n}$ is the number of integer variables, m is the number of rows and k is the number of cones in the problem. The heuristic is computationally cheap and called for each node after the bounding process.

3.3.7 Control Mechanism

A branch-and-cut algorithm need to make a decision between branching or cutting after each subproblem relaxation solve call with an infeasible solution. This is true for the algorithm being proposed. Both cutting and branching is aimed to close the optimality gap. In the branch-and-cut algorithm being proposed the decision is between disjunctive

conic cuts, MILP cuts, and conic approximation cuts versus branching.

Disjunctive conic cuts and MILP cuts work toward integer feasibility, whereas cone approximation cuts work towards conic feasibility. Branching works towards integer feasibility. The algorithm is flexible enough to make these type of branch-and-cut decisions for LP and SOCP type of subproblem relaxations.

In case of SOCP relaxations, algorithm we propose use cuts only at the root node. This use is parameterized and can be turned on and off when desired. We implemented and conducted experiments with the disjunctive conic cuts given by Belotti et al. [Bel+13; Bel+15]. Any other cut procedure can be used instead or together with the disjunctive conic cuts without the need of modification of the rest of the algorithm.

In case of LP relaxations, the algorithm keeps some statistics and makes the decision of whether to branch or cut using these statistics. The goal is to make the decision such that the branch-and-bound tree and hence the computation time is minimized.

A pure branch strategy, ignoring conic feasibility, and branching all the way until integer feasibility is obtained will lead integer solutions that are very unlikely to satisfy conic constraints. Once conic constraints are enforced the integer feasibility is lost. Moreover we might end up with an irrelevant part of the branch-and-bound tree and enforcing not the ideal disjunctions.

On the other hand, a pure cutting strategy, obtaining conic feasibility first at each node before branching is also not ideal. Note that this approach defaults to solving subproblem relaxations with the cutting-plane approach to optimality as in Algorithm 3 before branching. As discussed before, Algorithm 3 is slow compared to IPM. This means we have ended up with an algorithm that does pure branch and bound, as in SOCP relaxation case, and slower solving relaxations. However, a difference compared to using IPM solver is having optimal simplex tableaus at each node at the end of the bounding process. There are two advantages to this, (1) child nodes can be warm started, (2) integer

infeasible solutions can be cut using rich cut procedures of MILP literature. An immediate question that should be asked is whether these two advantages are good enough to beat the IPM based branch-and-bound method.

The control mechanism of the algorithm proposed in the forthcoming section is parameterized and user can ask for a behavior of the discussed pure strategies by updating three basic parameters. Default parameters do not follow a pure strategy. Default behavior of the algorithm is in a middle ground that aims to satisfy both integrality and conic constraints simultaneously.

3.3.8 Overall Algorithm

In this section we give the overall outer-approximation-based branch-and-cut algorithm. Branch-and-cut algorithm based on LP relaxations is referred as *bb-lp*. The algorithm based on SOCPs is referred as *bb-socp*. The algorithm is fairly simpler when subproblem relaxations are SOCP. bb-lp case is more complicated and it is explained in detailed in this section.

We discussed that Algorithm 3 is slow on solving SOCP compared to IPM. This approach suffers most when the conic constraints are large. When conic constraints are large, many supporting hyperplanes are required during Algorithm 3. At some instances we observed that 80% of the solution time is spent at the root node solving the initial SOCP. Solving rest of the branch-and-bound tree takes 20% of the time even though it includes many relaxation problems to solve. Rest of the tree is easier to solve for two reasons, (1) we already built an approximation of the conic constraints in the root node, (2) we exploit warm starting capability of the simplex algorithm in the lower level nodes. This is the regular behavior we observed for instances with large cones. Even though we exploit warm starting capability of simplex and gain advantage in the lower level nodes, this gained advantage is not enough to cover the time lost in the root node. bb-socp with

Algorithm 3 as solver was slow compared to bb-socp with an IPM based solver, especially on problems with large cones.

Moreover, as we discussed before, in this approach SOCP solver itself uses LP relaxations but branch-and-bound algorithm is unaware of this. Things can be improved if we let branch and bound be aware of this and stop using the cutting-plane algorithm as a black box solver and control the whole process. In this new design branch-and-bound algorithm is aware that subproblem relaxations are LP and it aims to get a solution that is both integer and conic feasible. Note that bb-socp with Algorithm 3 becomes an edge case of the new design. It is the case when the new algorithm is tuned to achieve conic feasibility at each node and branch afterwards. Algorithm 5 gives the overall bb-lp algorithm.

First we define a parametric family of polyhedra that contain the feasible region $\mathcal{S}_{\text{MISOCP}}$ of (MISOCP) as follows.

Definition 3.1. ($\mathcal{P}(l, u, \mathcal{E})$) For a given set $\mathcal{E} \subseteq \mathbb{Q}^n$, $\mathcal{P}(l, u, \mathcal{E})$ is the polyhedron defined by the inequalities of Theorem 2.2.1 corresponding to the members of set $\mathcal{E}$ for each of the Lorentz cones comprising the cartesian product $\mathbb{K}$, plus the linear constraints from (MISOCP) and lower and upper bounds on variables denoted by l and u. The formal description of this polyhedron is

$$
\begin{aligned}
& Ax = b \\
& x_1^i \geq 0 \qquad i = 1, \ldots, k \\
& a^{\overline{x}^i}(x^i) \geq 0 \quad \overline{x} \in \mathcal{E},\ i \in \mathcal{K}(\overline{x}) \qquad\qquad (\mathcal{P}(l, u, \mathcal{E})) \\
& x \geq l \\
& x \leq u.
\end{aligned}
$$

Here, $\mathcal{K}(x)$ is the index set of violated Lorentz cones by x. $a^{\overline{x}^i}$ are vectors as given in Theorems 2.2.1.

Let $\mathcal{H}$ be the set of subproblems (corresponding to branch-and-bound tree nodes) to be explored during bb-lp. We represent each subproblem with $(l, u, LB, \mathcal{E})$, where l and u are lower and upper bounds enforced on the variables for the corresponding subproblem, LB is the objective value estimate of the subproblem and $\mathcal{E}$ is the set of points used to generate supports for the conic constraints for the current subproblem. bb-lp populates $\mathcal{E}$ dynamically as needed during solution process.

Algorithm 5 Outer-Approximation-Based Branch-and-Cut Algorithm (bb-lp)

1: Call warm start procedure.
2: $UB \leftarrow \infty$.
3: $LB \leftarrow -\infty$.
4: **while** $\mathcal{H} \neq \emptyset$ **do**
5: Call bounding loop procedure.
6: **end while**

Algorithm 6 Warm Starting Procedure

1: Solve SOCP using IPM. Let $\hat{x}$ be optimal solution.
2: $\mathcal{E}^0 \leftarrow \{\hat{x}\}$
3: $i \leftarrow 0$
4: **do**
5: $\tilde{z}^i \leftarrow \min_{x \in \mathcal{P}(l,u,\mathcal{E}^i)} c^\top x$, $\tilde{x}^i \leftarrow \operatorname{argmin}_{x \in \mathcal{P}(l,u,\mathcal{E}^i)} c^\top x$
6: $\mathcal{E}^{i+1} \leftarrow \mathcal{E}^i \cup \{\tilde{x}^i\}$
7: $i \leftarrow i + 1$
8: **while** $i < iter\_limit$ and $\tilde{x}^i \notin \mathbb{K}_\epsilon$
9: $\mathcal{H} \leftarrow \{(-\infty, \infty, \tilde{z}^i, \mathcal{E}^i)\}$

bb-lp algorithm for MISOCP is similar to MILP branch-and-cut algorithm. The difference is, when relaxing subproblems *not only integrality constraints, but also conic constraints are relaxed*. Subproblem relaxations become LPs when the conic constraints are relaxed. A solution to the MISOCP is found when a relaxation solution satisfies *both integrality and conic constraints*. Moreover linear cuts introduced in Section 2.2 are used to cut solutions that are conic infeasible. MILP cut procedures (Gomory, mixed integer rounding, etc.) can also be used to improve the bounds of the relaxations. The algorithm

Algorithm 7 Bounding Loop

Select a problem (do not remove) from $\mathcal{H}$, denote it $(l, u, LB, \mathcal{E}^0)$.
$j \leftarrow 0$.
while true **do**
  if $LB \geq UB$ **then**
    $\mathcal{H} \leftarrow \mathcal{H} \setminus \{(l, u, LB, \mathcal{E}^j)\}$.
    Break loop.
  end if
  $\tilde{z}^j \leftarrow \min_{x \in \mathcal{P}(l,u,\mathcal{E}^j)} c^\top x$, $\tilde{x}^j \leftarrow \operatorname{argmin}_{x \in \mathcal{P}(l,u,\mathcal{E}^j)} c^\top x$
  $lastImp \leftarrow \tilde{z}^j - LB$.
  $\overline{LB} \leftarrow min_{(l,u,LB,\mathcal{E}) \in \mathcal{H}} LB$
  $gap \leftarrow \frac{UB - \overline{LB}}{UB} \times 100$.
  if $\tilde{z}^j \geq UB$ **then**
    $\mathcal{H} \leftarrow \mathcal{H} \setminus \{(l, u, LB, \mathcal{E}^j)\}$.
    Break loop.
  end if
  Apply cut cleaning.
  if $\tilde{x}^j$ is both integer and conic feasible **then**
    $UB \leftarrow \tilde{z}^j$.
    Break loop.
  else
    Call branch, constraint or price routine.
  end if
  if generateOAcuts or generateMILPcuts **then**
    if generateOAcuts **then**
      $\mathcal{E}^{j+1} = \mathcal{E}^j \cup \{\tilde{x}^j\}$.
    end if
    if generateMILPcuts **then**
      Generate MILP cuts and add to $\mathcal{P}(l, u, \mathcal{E}^j)$.
    end if
  else
    $\mathcal{H} \leftarrow \mathcal{H} \setminus \{(l, u, LB, \mathcal{E}^j)\}$.
    Pick i from $\{i \in I \mid \tilde{x}^j_i \notin \mathbb{Z}\}$.
    $\hat{u} \leftarrow u$, $\hat{u}_i \leftarrow \lfloor \tilde{x}_i \rfloor$.
    $\hat{l} \leftarrow l$, $\hat{l}_i \leftarrow \lfloor \tilde{x}_i \rfloor + 1$.
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(l, \hat{u}, \tilde{z}, \mathcal{E}^j), (\hat{l}, u, \tilde{z}, \mathcal{E}^j)\}$.
    Break loop.
  end if
  $j \leftarrow j + 1$.
end while

Algorithm 8 Branch, constrain or price routine.

if $\tilde{x}^j \notin \mathbb{K}_\epsilon$ **then**
  if ($j < \alpha$ or $lastImp > \beta \times gap$) and $j < \gamma$ **then**
    generateOAcuts $\leftarrow$ true
  else
    generateOAcuts $\leftarrow$ false
  end if
end if
if $\{i \in I \mid \tilde{x}_i^j \notin \mathbb{Z}\} \neq \emptyset$ **then**
  if $0 \equiv node\_number \,(\mathrm{mod}\kappa)$ and $lastImp \geq \delta$ **then**
    generateMILPcuts $\leftarrow$ true
  else
    generateMILPcuts $\leftarrow$ false
  end if
end if

has different flavors depending on the details of how/when the separation and MILP cuts are added.

An important decision during bb-lp algorithm is whether to branch or add cuts for a subproblem relaxation solution that violates both integrality and conic constraints. bb-lp algorithm aims to achieve both integrality and conic feasibility. Adding conic approximation cuts works toward conic feasibility and branching works toward integer feasibility. A pure strategy might ignore one and work toward the other. bb-lp tries to balance this objectives by using parameters α, β and γ to decide between cut generation and branching.

At a relaxation solution both integer and conic infeasible, for α times we run the cut generation procedure and add cuts to the subproblem. Moreover if number of cut generation rounds is over α but cut generation is improving bounds more than $\beta \times gap$ then we keep generating cuts. We stop generating cuts if number of cut generation rounds exceed γ.

Note that bb-lp with large α, γ and small β values is same as bb-socp together with Algorithm 3 to solve relaxations. bb-socp with Algorithm 3 implements a pure strategy where cuts are generated for each subproblem until a conic feasible solution is found or

problem becomes infeasible.

bb-lp is an outer-approximation algorithm like the one proposed in Vielma, Ahmed, and Nemhauser [VAN08], but it differs on building the outer-approximation. Vielma, Ahmed, and Nemhauser [VAN08] build the outer approximation fully (for some accuracy level) before starting the branch-and-bound process, where the algorithm proposed here builds the outer approximation gradually. The outer approximation supports are added to the problem when they are necessary to cut a conic infeasible solution. They are also added only in the neighborhood of the related portion/corner of the feasible region, regarding the objective direction.

Having a good approximation of conic constraints in the root node improves the subproblem relaxation objective value in the subsequent nodes. A warm starting procedure is used in the root node for this purpose. In warm starting procedure, integrality constraints are relaxed and the resulting continuous relaxation is solved. Supports to the conic constraints are added using the solution of this relaxation and results obtained in Section 2.2. LP approximation of the problem is improved with the addition of these supports. After this, the LP problem is solved repetitively. For each conic infeasible solution obtained, cuts are generated using the procedures introduced in Section 2.2. This procedure stops once an iteration limit is reached or relaxation solution is conic feasible. In other words Algorithm 3 is used, but the problem is not solved to the optimality and the algorithm is interrupted after a fixed number of iterations.

In the bounding loop procedure we proceed similar to a typical branch-and-cut algorithm for MILP. The difference is when we check for the feasibility of solutions, we check both integrality and conic feasibility. Note that when a subproblem relaxation solution violates both integrality and conic constraints, algorithm should decide to branch or cut. Branch, constrain or price (bcp) procedure decides on this. Behavior of this procedure is controlled by OA cut parameters α, β and γ and MILP cut parameters κ and δ. If

iteration number is equal or greater than γ we decide to branch. If not, then, if iteration number is less than α or the last time we generated OA cut, the objective improvement was greater that β times the optimality gap, bcp procedure decides to generate more OA cuts. At every κ number of nodes MILP cuts are generated in case the solution violates integrality. MILP cuts are generated as long as the bound improvement is larger than δ.

bb-lp can be made to behave like bb-socp with Algorithm 3 as solver by choosing appropriate values for α, β and γ parameters of branch, constraint and price procedure. $\alpha := \infty$, $\beta := -1$ and $\gamma = \infty$ would give such a behavior.

In absence of conic constraints (MILPs) bb-lp reduces to branch and cut for MILP. At each node in Algorithm 4, Vielma, Ahmed, and Nemhauser [VAN08] solve the linear approximation problem (CXLP), that approximates (CXP). Note that a solution to (CXLP) is only approximately feasible to conic constraints, similar to solutions of $(\mathcal{P}(l, u, \mathcal{E}))$ in bb-lp case. Solutions of (CXLP) are conic feasible for a fixed parameter of Ben-Tal and Nemirovski approximation, where conic feasibility of solutions of $(\mathcal{P}(l, u, \mathcal{E}))$ depends on the set $\mathcal{E}$. Algorithm 4 obtains conic feasible solutions by solving (CXP) (SOCP in our case). bb-lp obtains conic feasibility by a cutting-plane approach, i.e., generates cuts and adds them to $(\mathcal{P}(l, u, \mathcal{E}))$ until a conic feasible solution is obtained.

3.4 Conclusion

In this chapter we propose a new outer-approximation algorithm to solve MISOCP. This algorithm uses a closed form linear cut, introduced in Chapter 2, to separate a given point. The algorithm introduced is a branch-and-cut algorithm based on LP relaxations. In subproblems not only integrality constraints but also conic constraints are relaxed. Conic constraints are enforced through linear cuts generated.

Algorithm is motivated by the warm start capability of simplex algorithm in branch-and-bound frameworks. Algorithm introduced aims to satisfy integrality and conic con-

straints simultaneously. Level of enforcement of conic constraints are controlled through parameters. Algorithm defaults to branch and bound with SOCP relaxations for specific values of parameters.

Many questions are raised in this chapter at introduction of the algorithm regarding the performance of various strategies and parameter values. These questions are answered in Chapter 6 through computational experiments.

Algorithm introduced in this chapter is implemented within the COIN-OR framework. Chapter 4 introduces the software libraries created for implementations. It explains the design of these libraries and their relationship to the existing COIN-OR projects.

Chapter 4

Software for MISOCP

This chapter presents independent software libraries that are designed to solve MISOCP. These libraries implement the algorithms introduced in Chapter 3. This chapter provides the details of these software libraries and how they are used for solving MISOCP.

The software projects introduced in this chapter are tightly integrated within the current projects of COIN-OR (**CO**mputational **IN**frastructure for **O**perations **R**esearch) [Lou03] initiative. Projects introduced here depend on various COIN-OR projects. These dependent projects are,

- BuildTools [For+b],
- CoinUtils (**Coin**-or **Util**ities) [For+d],
- OSI (**O**pen **S**olver **I**nterface) [For+e],
- CLP (**C**OIN-OR **L**inear **P**rogramming) [For+c] solver,
- CGL (**C**ut **G**eneration **L**ibrary) [For+a],
- Ipopt (**I**nterior **P**oint **OPT**imizer) [WB06],

- CHiPPS (**C**OIN-OR **Hi**gh-**P**erformance **P**arallel **S**earch) framework [RLS04; Xu+05; Xu+09; Xu+a; Xu+b].

The libraries we present share the same build and compilation process with current COIN-OR projects. This is by design to achieve the two following goals: (1) taking advantage of the current software provided by COIN-OR rather than re-inventing the wheel, (2) attracting the current audience of the COIN-OR tools and making the software accessible and natural to them.

The software libraries presented here are planned to be a part of the COIN-OR initiative. Libraries introduced can be listed as follows.

- OsiConic: A generic interface class for SOCP solvers. This interface provides a way to build and solve SOCPs that is uniform across a variety of solvers, as well as a standard interface for querying the results.
- OsiXxxxx: Implementations of the interface for various open source and commercial solvers.
- COLA: A solver for SOCP that implements the cutting-plane Algorithm 3.
- CglConic: A library of procedures for generating valid inequalities for MISOCP.
- DisCO: A solver library for MISOCP that uses all the libraries mentioned. This library implements classical branch-and-bound type of algorithm and the outer approximation branch-and-cut algorithm given in Algorithm 5.

All libraries introduced here are implemented in `C++`. `C++` is chosen as the language of development due to its superior performance and compatibility with the current COIN-OR initiative projects.

4.1 OsiConic, A Solver Interface for SOCP

In this section, we introduce OsiConic [BR16g], a generic interface for SOCP solvers. OsiConic is built on top of COIN-OR's linear solver interface OSI and extends OSI to SOCP. It is composed of classes for storing and manipulating the data required to describe an SOCP, as well as a class containing the actual solver interface.

4.1.1 Classes in OsiConic

`OsiCone`

`OsiCone` is an abstract base class that provides an interface for storing a cone, i.e., a conic constraint, and querying properties of the cone. It declares two virtual functions, `feasible` and `project`. Function `feasible` checks feasibility of a given point. Function `project` projects a given point that violates the conic constraint onto the cone boundary. Function `project` is used at the linear approximation process of the cone.

`OsiLorentzCone`

`OsiLorentzCone` is a class for representing both Lorentz cones ($\mathbb{L}$) and rotated Lorentz cones. It inherits `OsiCone` and implements `feasible` and `project` functions.

`OsiScaledCone`

`OsiScaledCone` is a class to represent second-order cones given in conic dual form. `OsiScaledCone` inherits and implements `OsiCone`. `OsiScaledCone` keeps coefficient matrix in sparse matrix data structures provided by CoinUtils.

OsiConicSolverInterface

`OsiConicSolverInterface` is an abstract base class for a SOCP solver. It provides functions to build and solve SOCP.

`OsiConicSolverInterface` extends the OSI library's `OsiSolverInterface` class for LPs by methods specific to conic optimization problems, such as adding/removing/querying cones, etc.. It is designed to be compatible with OSI and easily understandable by the current users of OSI. Current applications that depend on `OsiSolverInterface` can compile with `OsiConicSolverInterface` (full backward compatibility). Moreover it provides methods related to conic constraints.

A user of `OsiSolverInterface` has to learn the cone related functions of OsiConic only, and can start writing applications on top of `OsiConicSolverInterface` immediately. The interface of the `OsiConicSolverInterface` class is roughly as in Listing 4.1.

Listing 4.1: `OsiConicSolverInterface` class design

```
1 class OsiConicSolverInterface : virtual public OsiSolverInterface {
2 public:
3   virtual void getConicConstraint(...) const = 0;
4   virtual void addConicConstraint(...) = 0;
5   virtual void removeConicConstraint(...) = 0;
6   virtual void modifyConicConstraint(...) = 0;
7   virtual int getNumCones() const = 0;
8   virtual int getConeSize(int i) const = 0;
9   virtual OsiConeType getConeType(int i) const = 0;
10  virtual int readMps(const char * filename);
```

The `OsiConicSolverInterface` defines `readMps` function to read SOCP instances from MPS input files. MPS format was originally developed for linear problems and extended for SOCP. At the time this thesis is being written, there are two different extended MPS formats from two different commercial solver companies, CPLEX and Mosek. `OsiConicSolverInterface` assumes Mosek format. `OsiConicSolverInterface` uses CoinUtils to implement `readMps` and CoinUtils assumes Mosek format.

4.1.2 Interfacing to CPLEX, Mosek, and Ipopt

`OsiConicSolverInterface` is implemented for CPLEX, Mosek and Ipopt. OsiCplex [BR16b], OsiMosek [BR16d] and OsiIpopt [BR16c] implements `OsiConicSolverInterface` for the corresponding solvers.

A user developing applications using `OsiConicSolverInterface` can use her choice of solver without modifying her code. She can use these solvers interchangeably without any additional development effort. This flexibility is exploited further in the software that is introduced in the forthcoming sections.

A simple application build on top of `OsiConicSolverInterface` is given in the following.

Listing 4.2: `OsiConicSolverInterface` class design

```
1 // Solves conic problems, reads them in mosek mps input format
2 // usage: ./cont_solver input.mps
3 #include <OsiIpoptSolverInterface.hpp>
4 #include <iostream>
5
6 int main(int argc, char ** argv) {
7   OsiConicSolverInterface * solver = new OsiIpoptSolverInterface();
8   solver->readMps(argv[1]);
9   solver->initialSolve();
10  std::cout << "Objective is " << solver->getObjValue() << std::endl;
11  delete solver;
12  return 0;
13 }
```

The application given in 4.2 uses Ipopt to read a problem from an MPS input file and solve it. Similarly, CPLEX or Mosek can be used by updating line 3 to include the corresponding header and line 7 to create the right class instance. Users can also create problems from scratch by adding constraints, variables etc. For this, check the examples directory of the desired interfaces.

The OSI library already provides implementations of CPLEX and Mosek for linear problems. OsiCplex and OsiMosek depends on these linear implementations. OsiCplex and OsiMosek implements the relevant parts of the conic solver interface, i.e., conic spe-

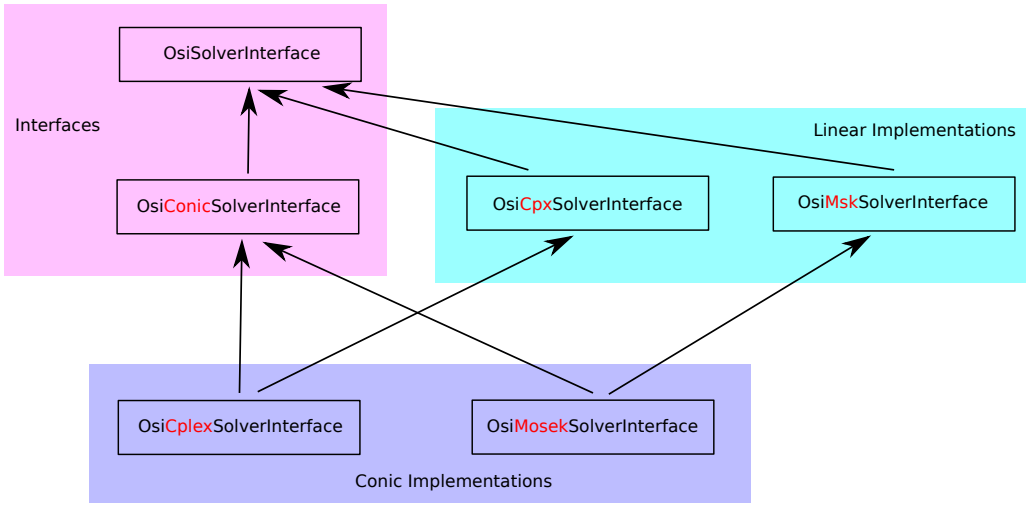


Figure 4.1: CPLEX and Mosek conic interface inheritance diagram

cific interface. The linear interface is inherited from the existing implementations in OSI library. Inheritance diagram presented in Figure 4.1 demonstrates this relationship.

The design is a little different for the case of OsiIpopt. OsiIpopt conic interface is implemented from scratch since no previous implementation of the interface existed. Data structures provided by the CoinUtils library is used to store the linear part of the SOCP, i.e., coefficient matrix, variable bounds, etc. Data structures provided by OsiConic (`OsiLorentzCone`, `OsiScaledCone`) are used to store the conic part of the problem. `OsiIpoptSolverInterface` inherits `TNLP` class of Ipopt. The inheritance and membership diagram presented in Figure 4.2 demonstrates the OsiIpopt design.

In OsiIpopt conic constraints are modeled using quadratic inequality $x_1^2 - x_2^2 \cdots - x_n^2 \geq 0$ and nonnegativity of the leading variable $x_1 \geq 0$.

4.2 COLA: A solver library for SOCP

COLA (Conic Optimization with Linear Approximations) [BR16a] is a solver library that implements the outer-approximation algorithm given in Algorithm 3. COLA is built on top

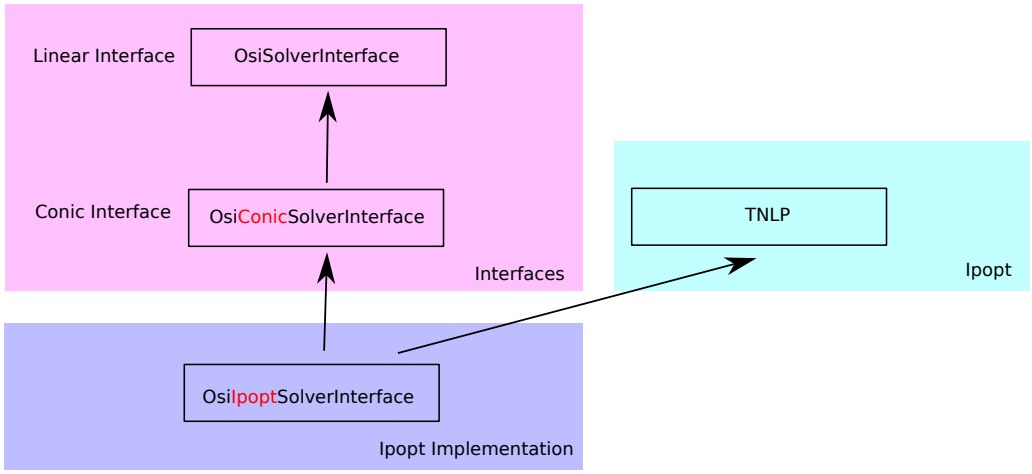


Figure 4.2: Ipopt conic interface inheritance diagram

of CLP to solve linear optimization problems. COLA inherits CLP's OSI implementation, i.e., class named `OsiClpSolverInterface`, and extends it for conic needs by implementing `OsiConicSolverInterface` abstract base class. Inheritance diagram of COLA's problem representation class `ColaModel` is presented in Figure 4.3. Following are the main classes of COLA.

`ColaModel`

`ColaModel` is a class to represent an SOCP. It inherits two other classes, `OsiClpSolverInterface` and `OsiConicSolverInterface`. `OsiClpSolverInterface` is the CLP implementation of the liner solver interface of COIN-OR, i.e. `OsiSolverInterface`. `OsiConicSolverInterface` is an abstract base class that defines an interface for SOCP. `OsiConicSolverInterface` inherits `OsiSolverInterface`, rather than defining an interface from scratch. SOCP is an extension of LP, similarly `OsiConicSolverInterface` is extension of `OsiSolverInterface`. `OsiSolverInterface` already fixes a language for LP, `OsiConicSolverInterface` adds the missing pieces for conic constraints.

Note that `ColaModel` inherits `OsiSolverInterface` twice, one from `OsiConicSolver-`

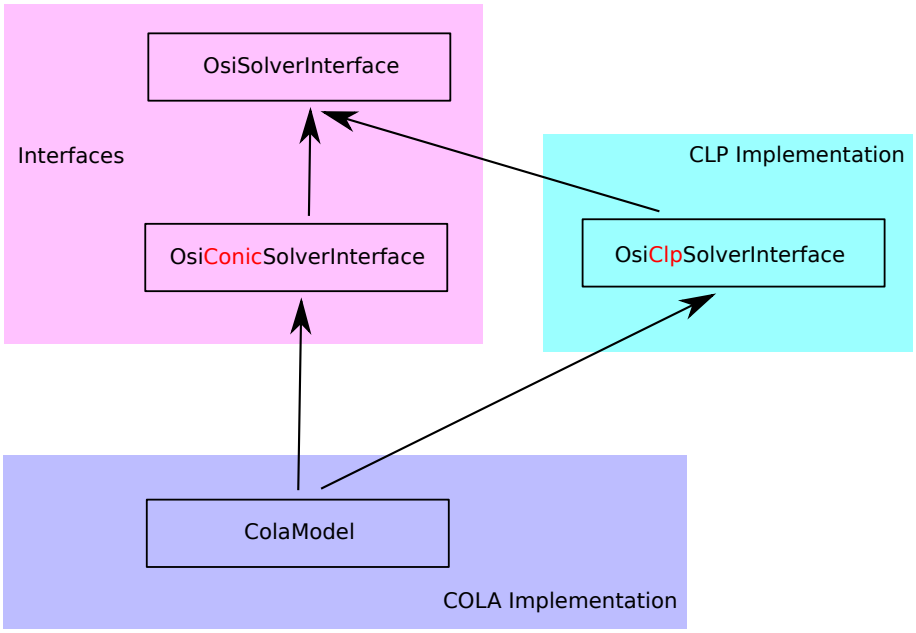


Figure 4.3: COLA inheritance diagram

`Interface` and once from `OsiClpSolverInterface`. This inheritance scheme might create a problem known as the *diamond inheritance* if not handled carefully. The diamond problem would occur if `ColaModel` would inherit a function that is implemented in both `OsiConicSolverInterface` and `OsiClpSolverInterface`. `ColaModel` is designed to ensure that this does not happen. Moreover, since `OsiSolverInterface` is inherited twice, another problem could arise, which is the possibility of having two copies of the same object. This problem is prevented since `OsiSolverInterface` is inherited as *virtual* in `OsiConicSolverInterface` (it is also inherited as virtual in `OsiClpSolverInterface`). Virtual inheritance ensures that one and only one instance of `OsiSolverInterface` is initiated when a new `ColaModel` object is created.

`ColaModel` only implements the conic related virtual functions of `OsiConicInterface`. It does not need to implement the linear functions since they are already implemented in the inherited `OsiClpSolverInterface`. ColaModel overwrites the definitions of the following functions inherited from `OsiClpSolverInterface`,

- `initialSolve`,
- `resolve`,
- `clone`,
- `writeMps`,
- `isAbandoned`,
- `isProvenOptimal`,
- `isProvenPrimalInfeasible`,
- `isProvenDualInfeasible`,
- `isPrimalObjectiveLimitReached`,
- `isDualObjectiveLimitReached`,
- `isIterationLimitReached`.

COLA can be *warm* started by using *resolve* function. Warm starting is used when COLA is the choice of solver in a branch-and-bound algorithm. Warm starting does use CLP's warm start capability.

`LorentzCone`

`LorentzCone` is COLA's representation of Lorentz and rotated Lorentz cones. `LorentzCone` has functionality to check feasibility of a given point. It also has a function to compute a supporting hyperplane that separates a given infeasible point.

`Options`

`Options` class is used to store user provided options. It provides an interface to set COLA options.

`Separate`

`Separate` class is used to separate a given point from all conic constraints of the problem. `Separate` has functions to generate a set of linear constraints that separate the point from the conic constraints.

4.3 CglConic, A Cut Library for MISCOP

CglConic [BR16e] is a library of procedures for generating valid inequalities for MISOCP. CglConic can be thought as an extension of COIN-OR's CGL for SOCP. It requires an SOCP solver that implements OsiConic. CglConic use interface fixed by OsiConic to communicate with the SOCP solver and is solver independent. User can provide the SOCP solver to be used during compilation.

CglConic is designed after COIN-OR's CGL and shares the same principles. As with CGL, there is a generic interface to cut generators, `CglConicCutGenerator`, and implementations of this interface for various cut procedures.

CglConic is used to generate valid inequalities that improve the relaxations of underlying MISOCP. Cuts generated by CglConic can be linear or second-order conic. Generated cuts are valid for the MISOCP feasible set and are violated by given integer infeasible or conic infeasible solutions of relaxations. CglConic currently implements procedures for the generation of inequalities in the following classes.

- Disjunctive Cuts given by Belotti et al. [Bel+13],
- Outer-approximation inequalities given in Chapter 2,
- Interior point method approximation cuts.

Procedures for cuts in the following classes are being considered for future implementation.

- Conic mixed-integer rounding (MIR) cuts by Atamtürk and Narayanan [AN10],
- Two-term disjunctions on the second-order cone by Kılınç-Karzan and Yıldız [KY14],
- Cuts for 0-1 mixed convex programming by Stubbs and Mehrotra [SM99],
- Cuts for mixed 0-1 conic programming by Çezik and Iyengar [ÇI05].

CglConic has two abstract base classes, `CglConicCutGenerator` and `CglConicParam`. `CglConicCutGenerator` is a base class for defining the standard interface to procedures for generating valid inequalities. Classes containing various implementations are build on top of this base class and implement the interface it specifies. `CglConicParam` is a base class for parameters, implementations extend it to implement parameters of their own. Other than the base classes, CglConic also has implementations of the procedures listed above. One such class is `OsiConicGD1`. It inherits the cut generator interface `CglConicCutGenerator` and implements it for disjunctive cuts given by Belotti et al. [Bel+13].

Figure 4.4 presents the inheritance diagram of CGL and CglConic library classes and demonstrates the analogy between them. CGL has various implementations of cut procedures from MILP literature (MIR, Gomory, etc.). Figure 4.4 includes MIR only for simplicity. Similarly in conic case only general disjunctions and conic MIR cut procedures are included for simplicity, excluding OA and IPM approximation cuts. Following sections explains the implementations in detail.

4.3.1 Implementing Disjunctive Cuts

`CglConicGD1` class implements the disjunctive cuts given by Belotti et al. [Bel+13]. It inherits and implements the `CglConicCutGenerator` base class. `generateAndAddCuts` function creates a clone of the input solver interface, generates cuts, adds them to the clone and returns a pointer to it. The resulting solver interface is an improved version of the starting solver with the generated cuts.

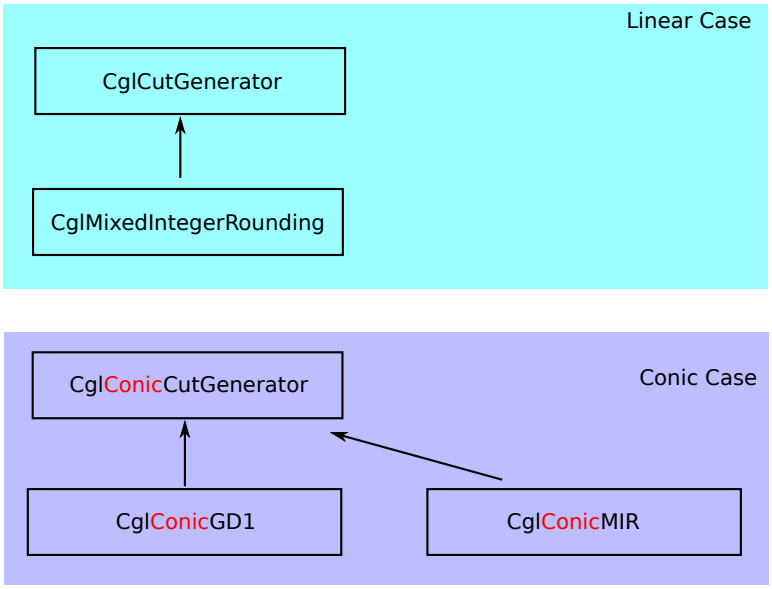


Figure 4.4: CGL and CglConic inheritance diagrams

`CglConicGD1Cut` is a class to generate and represent a disjunctive conic cut. It stores the disjunction used to generate the cut and the resulting cut. It might return a linear cut if one of the disjunctions is infeasible.

Computation of disjunctive cuts require singular value decomposition and solving a system of equations with positive definite coefficient matrix. Basic Linear Algebra Subprograms (BLAS) package is used for these type of computations.

Listing 4.3: `CglConicGD1` example application

```
int main(int argc, const char *argv[]) {
  string mpsFileName = argv[1];
  // Instantiate a specific solver interface
  OsiConicSolverInterface * si = new SOCP_SOLVER();
  // Read file describing problem
  si->readMps(mpsFileName.c_str(),"mps");
  // Solve continuous problem
  si->initialSolve();
  // Save the orig socp relaxation value for
  // comparisons later
  double origSocpObj = si->getObjValue();
  // Instantiate cut generator
  CglConicGD1 cg(si);
  bool equalObj;
  CoinRelFltEq eq(0.0001);
```

```
16   int num_cut = 0;
17   double obj;
18   // Keep applying cuts until no more cuts are generated
19   do {
20     // Get current solution value
21     obj = si->getObjValue();
22     // Generate and apply cuts
23     OsiConicSolverInterface * nsi = cg.generateAndAddCuts(*si);
24     delete si;
25     si = nsi;
26     si->resolve();
27     equalObj = eq(si->getObjValue(), obj);
28   } while (!equalObj);
29   // Print total number of cuts applied,
30   // and total improvement in the SOCP objective value
31   cout << endl << endl << endl;
32   cout << "------------------------------------------------------------------";
33   cout << "Cut generation phase completed:" <<endl;
34   cout << "   " << cg.getNumCutsAdded() << " many cuts added." << endl;
35   cout << "   changing the SOCP objective value from " << origSocpObj
36        << " to " << si->getObjValue() <<endl;
37   cout << "------------------------------------------------------------------"
38        << endl << endl << endl;
39   delete si;
40   return 0;
41 }
```

Listing 4.3 presents the `main` function of an example application that use disjunctive conic cuts. This application generates disjunctive cuts until objective function value improvement diminishes. The function returns once cut statistics are printed. At each iteration of the while loop, the `generateAndAddCuts` function returns a new SOCP solver interface that contains the cuts generated. Main function assumes `SOCP_SOLVER` macro is defined with a valid SOCP solver interface. See examples directory of CglConic for the complete application.

`generateAndAddCuts` function generates and adds all possible disjunctive cuts. It uses the current solution stored in the input solver interface instance to create disjunctions. It enumerates the fractional valued integer variables to generate simple variable disjunctions. For each such variable it determines the cone it is in and then looks for equality constraints that have nonzero coefficients for only members of this cone. The fractional variable and the linear constraints determined are used to create the disjunctive conic cut. `generateAndAddCuts` functions generates all possible cuts it can by searching variables

and constraints that satisfy these criteria. In computational experiments, we modify the behavior of this function to experiment with the performance with respect to number of cuts added.

The disjunctive cut is in scaled cone form initially when computed. `generateAndAddCuts` function converts it to standard form by introducing a new set of variables. Once converted to standard form, a conic cut can be considered as a linear equality system containing the starting variables and the newly introduced variables, plus restriction of new variables into a Lorentz cone of relevant dimension. Adding a disjunctive conic cut to a relaxation increase the number of variables and linear constraints of the problem. Moreover it introduces a new conic constraint in Lorentz cone form.

4.3.2 Conic Outer-Approximation (OA) Inequalities

CglConic has an implementation for the OA inequalities introduced in Chapter 2. These inequalities are used to approximate second-order cones when the conic constraints are relaxed in solving an MISOCP. Listing 4.4 uses CglConic's OA inequalities and presents a cutting-plane solver for SOCP. This can be considered a more basic implementation of the Algorithm 3. This example can be found in CglConic project's examples directory.

Listing 4.4: OA algorithm using CglConic

```
1 // Implements simple cutting-plane solver for SOCP.
2 // usage: cutting_plane_solver input_file.mps
3 #include <CglConicOA.hpp>
4 #include <ColaModel.hpp>
5
6 int main(int argc, char ** argv) {
7   // create a solver instance
8   ColaModel * conic_solver = new ColaModel();
9   // read problem including conic constraints
10  conic_solver->readMps(argv[1]);
11  // solve initial problem ignoring conic constraints.
12  conic_solver->OsiClpSolverInterface::initialSolve();
13  // create cut generator
14  CglConicOA cg(1e-5);
15  OsiCuts * cuts;
16  int total_num_cuts = 0;
17  clock_t start_time = clock();
```

```
18   // solve problem while we can generate cuts.
19   do {
20     // ignore conic constraints and solve LP
21     conic_solver->OsiClpSolverInterface::resolve();
22     // generate cuts
23     cuts = new OsiCuts();
24     cg.generateCuts(*conic_solver, *cuts);
25     // add cuts to the problem
26     int num_cuts = cuts->sizeRowCuts();
27     if (num_cuts==0) {
28       delete cuts;
29       break;
30     }
31     else {
32       std::cout << num_cuts << " many cuts produced." << std::endl;
33     }
34     total_num_cuts += num_cuts;
35     conic_solver->OsiSolverInterface::applyCuts(*cuts);
36     delete cuts;
37   } while(true);
38   clock_t duration = clock() - start_time;
39   // print solution status
40   conic_solver->report_feasibility();
41   std::cout << "Total number of cuts: " << total_num_cuts << std::endl;
42   std::cout << "Objective value:      " << conic_solver->getObjValue()
43             << std::endl;
44   std::cout << "CPU time:             "
45             << double(duration)/double(CLOCKS_PER_SEC) << std::endl;
46   delete conic_solver;
47   return 0;
48 }
```

4.3.3 IPM Approximation Cuts

IPM approximation cuts are also used to approximate second-order cones when the conic constraints are relaxed. It produces linear constraints that support the underlying conic constraint. Generation of IPM approximation cuts is different than OA cuts, however. To generate IPM approximation cuts, the underlying SOCP is solved using IPM method and linear supports are added at the optimal solution (and optionally points near optimal solution) for the binding conic constraints. The intent of IPM approximation cuts is to reduce the number of iterations when the point being separated is integer feasible but violates the conic constraints in the branch-and-cut algorithm.

4.4 DisCO, A Distributed-Memory-Parallel MISOCP Solver

In this section, we introduce the DisCO solver library [BR16f]. DisCO is a distributed-memory-parallel solver library for MISOCP. DisCO is motivated by various research questions which are explicitly voiced in corresponding chapters while introducing the relevant concepts. This section restates them briefly.

An important motivation for creating DisCO is to provide the ability to compare a wide variety of branch-and-bound algorithms for MISOCP while holding as many aspects of the algorithm constant as possible in order to make more rigorous comparisons possible. Among other things, the goal is to compare LP-based algorithms with SOCP-based algorithms. DisCO is built to be able to perform detailed experiments comparing various aspects of performance.

Another motivation of DisCO is to investigate the effectiveness of different strategies for generation of valid inequalities. DisCO can be used to experiment with settings for the OA cut generation parameters, for example, in the case of an LP relaxation-based algorithm. Moreover, DisCO is aimed to resolve the question of whether the integration of procedures for generating valid inequalities from the MILP literature can help in solving MISOCPs. DisCO provides options for experimenting with various strategies for generating MILP cuts. Finally, another motivation is to be able to experiment with the generation and addition of disjunctive conic cuts. DisCO provides interfaces to enable generation of disjunctive conic cuts in the root node.

DisCO can be used together with COLA, which implements the cutting-plane algorithm introduced in Algorithm 3, for the case of SOCP relaxations. From preliminary experiments with COLA, we know that this algorithm is slow compared to IPM for solving SOCP. However, the warm-starting capability of simplex can be exploited once this algorithm is used in a branch-and-bound framework. A motivation of DisCO is to resolve whether the expected performance gain of cutting-plane algorithms from simplex

warm-starting in branch and bound is good enough to compete with IPM.

Another use of DisCO is to assess performance of various branching strategies for different subproblems cases. The performance of branching strategies has been well-studied in the case of MILP [Bén+71; Mit73; LS99; FM05b]. DisCO aims to explore Algorithm 5 under different branching strategies. Moreover, DisCO is aimed to answer whether a given specific branching strategy performs similarly for each subproblem case. DisCO also explores strategies to balance cutting versus branching for branch-and-cut algorithm given in Algorithm 5.

DisCO is used to answer the question of whether parallelizing branch-and-bound algorithm help. The motivation is to measure parallel performance for each subproblem case. There are studies that experiment and measure the scalibility of CHiPPS for MILP case. DisCO is aimed to answer the same questions for MISOCP case. It is designed to measure the scalibility of the CHIPPS framework for MISOCP problems for both of the subproblem cases.

An important motivation of DisCO is to investigate these questions in a precisely controlled environment. DisCO is flexible enough to implement all the research questions raised in this study. It provides interfaces to experiment with different parameter variations of the underlying algorithms.

DisCO is built using the CHiPPS framework. It is tightly integrated with other COIN-OR libraries. It uses COIN-OR's BuildTools for building. It depends on CLP to solve LP subproblems. It uses OSI to communicate with CLP and CoinUtils' data structures for logging messages and storing sparse matrices and vectors. CGL is used to generate cuts from MILP literature.

DisCO depends on OsiConic and its implementations for COLA, CPLEX, Ipopt and Mosek to solve SOCP. CglConic is used to generate disjunctive conic cuts and approximation cuts introduced in this dissertation.

DisCO is similar to CHiPPS' BLIS, MILP solver of the CHiPPS framework, in design. The next section gives an overview of the CHiPPS framework. Understanding the CHiPPS framework is vital to understand the design of DisCO. DisCO, its classes and design is explained in the following section.

4.4.1 COIN-OR High-Performance Parallel Search (CHiPPS)

CHiPPS is a distributed memory parallel search framework for parallelizing searching the branch-and-bound tree. It is composed of three layers of different software libraries, ALPS, BCPS, and BLIS. In this section we briefly introduce these layers, their motivation and design. But first, we start with explanation of the fundamental parallelization scheme of the framework in the next subsection.

A different parallelization paradigm

The first layer in CHiPPS framework is ALPS. It is an abstract library for parallel tree search. ALPS does not follow a master–slave parallelization paradigm. It inserts another level in-between master and slave called *hub*. Master–hub–slave paradigm is proposed by Ralphs, Ladanyi, and Saltzman [RLS04] to overcome the scaling problem of master–slave paradigm. Master–slave paradigm does not scale well, since master becomes a communication bottleneck as the number of workers increases. In master–hub–slave paradigm, master node communicates with hub nodes, hub nodes communicate with slave nodes. Slave nodes do not communicate with master node directly. Number of hubs is determined by ALPS or can be given as an input by the user to balance the communication. Note that master–hub–slave paradigm reduces to master–slave paradigm when master is the only hub. In master–hub–slave paradigm, communication workload of master is pushed to hubs. The price paid to relieve the communication bottleneck is any hub–slave cluster will not immediately have access to all the knowledge objects produced

in other hub–slave clusters. Note that in master–slave paradigm, all slave nodes have access to all the knowledge produced in any other slave node through master node. In the master–hub–slave paradigm a knowledge produced in a slave will be available to other hub–slave clusters (through master) only when the slave propagates the knowledge to its hub and the hub to the master.

ALPS

ALPS was introduced by Xu et al. [Xu+05] and is designed to conduct parallel tree search using master–hub–slave paradigm. ALPS parallelizes the branch-and-bound search by distributing subtrees to be searched among hubs. A hub create more subtrees from the given one and distribute them to its slaves.

Any tree search algorithm, i.e., DFS, BFS, Dijkstra's algorithm or the branch-and-bound algorithm for optimization problems, can be implemented as an application on top of the ALPS. An application should implement the following classes that inherit the correspondent classes from ALPS.

- A model class to represent the problem being solved and to keep the problem data.
- A tree node class to represent a tree node of the specific tree search algorithm being implemented.
- A node description class to hold the data corresponding to a node of the tree, node class and node description classes are separated for convenience. ALPS tree node instances have a member that points to underlying node description instance.
- A solution class to store solutions of the problem.

The correspondent classes in ALPS are implemented as abstract base classes. ALPS will be able to call the user implemented functions and carry the search since these functions are defined as pure virtual.

ALPS depends on Message Passing Interface (MPI) standard [CGH94] for network communication. It can be compiled with both MPICH [Gro02] and OpenMPI [Gab+04] implementations of MPI. ALPS can also work in serial when desired or an MPI library is not available.

Branch, Constrain, and Price (BCPS) Library

BCPS given in Xu et al. [Xu+b] is the second layer of the CHiPPS framework. It is another abstract library built on top of ALPS. ALPS is designed for an arbitrary tree search without any assumptions regarding to the specifics of the search. BCPS is built on top of ALPS and has two crucial assumptions, (1) an optimization problem is being solved with (2) a branch–cut–price algorithm is being used. It does not assume anything more about the problem (linear/nonlinear, discrete/continuous, convex/non-convex etc.). It provides a basis for any kind of branch-and-bound solver. BCPS itself is not a solver but rather a base library for building branch-and-bound based solvers.

BCPS implements the common framework that is shared by all branch-and-bound type of algorithms. This is done without assuming the structure of the underlying optimization problem. It inherits related abstract base classes of ALPS and extends them with the two assumptions mentioned.

The backbone of BCPS can be considered as the node processing function provided in its tree node class. This function calls the pure virtual functions that are implemented by the solver application to process a node of the branch-and-bound tree. This function, together with bounding loop function it calls, is presented in Listing 4.5 with some abstraction for saving the readers with copious details.

Listing 4.5: BCPS process and bounding loop functions

```
1 int BcpsTreeNode::process(bool isRoot, bool rampUp) {
2   AlpsNodeStatus status = getStatus();
3   BcpsModel * model = dynamic_cast<BcpsModel*>(broker()->getModel());
4   CoinMessageHandler * message_handler = model->bcpsMessageHandler_;
```

```
5   // check if this can be fathomed
6   if (getQuality() > broker()->getBestQuality()) {
7     // debug message
8     message_handler->message(0, "Bcps",
9                              "Node fathomed due to parent quality.",
10                             'G', BCPS_DLOG_PROCESS);
11    // end of debug message
12    setStatus(AlpsNodeStatusFathomed);
13    return AlpsReturnStatusOk;
14  }
15
16  if (status==AlpsNodeStatusCandidate or
17      status==AlpsNodeStatusEvaluated) {
18    boundingLoop(isRoot, rampUp);
19  }
20  else if (status==AlpsNodeStatusBranched or
21           status==AlpsNodeStatusFathomed or
22           status==AlpsNodeStatusDiscarded) {
23    // this should not happen
24    message_handler->message(BCPS_NODE_UNEXPECTEDSTATUS,
25                             model->bcpsMessages_)
26      << static_cast<int>(status) << CoinMessageEol;
27  }
28  return AlpsReturnStatusOk;
29}
30
31int BcpsTreeNode::boundingLoop(bool isRoot, bool rampUp) {
32  AlpsNodeStatus status = getStatus();
33  BcpsModel * model = dynamic_cast<BcpsModel*>(broker_->getModel());
34  CoinMessageHandler * message_handler = model->bcpsMessageHandler_;
35
36  bool keepBounding = true;
37  bool fathomed = false;
38  bool do_branch = false;
39  bool genConstraints = false;
40  bool genVariables = false;
41  BcpsConstraintPool * constraintPool = new BcpsConstraintPool();
42  BcpsVariablePool * variablePool = new BcpsVariablePool();
43  // installs problem to the underlying relaxation solver
44  installSubProblem();
45  while (keepBounding) {
46    keepBounding = false;
47    // solve subproblem corresponds to this node
48    BcpsSubproblemStatus subproblem_status = bound();
49    // call heuristics to search for a solution
50    callHeuristics();
51    // decide what to do next, branch, constrain or price?
52    branchConstrainOrPrice(subproblem_status, keepBounding, do_branch,
53                           genConstraints,
54                           genVariables);
55    // branchConstrainOrPrice might decide fathoming this node
56    if (getStatus()==AlpsNodeStatusFathomed) {
57      // node is fathomed, nothing to do.
58      break;
59    }
60    else if (keepBounding and genConstraints) {
61      generateConstraints(constraintPool);
```

```
62        // add constraints to the model
63        applyConstraints(constraintPool);
64        // clear constraint pool
65        constraintPool->freeGuts();
66        // set status to evaluated
67        setStatus(AlpsNodeStatusEvaluated);
68      }
69      else if (keepBounding and genVariables) {
70        generateVariables(variablePool);
71        // add variables to the model, set status of this node to evaluated
72        setStatus(AlpsNodeStatusEvaluated);
73      }
74      else if (keepBounding==false and do_branch==false) {
75        // put node back into the list.
76        // this means update node status as evaluated and end processing
77        // the node.
78        setStatus(AlpsNodeStatusEvaluated);
79      }
80      else if (keepBounding==false and do_branch) {
81        // prepare for branch() call
82        BcpsBranchStrategy * branchStrategy = model->branchStrategy();
83        branchStrategy->createCandBranchObjects(this);
84        // prepare this node for branching and set status as pregnant
85        processSetPregnant();
86      }
87      else {
88        message_handler->message(9998, "Bcps",
89                                 "This should not happen. "
90                                 " branchConstrainOrPrice() is buggy.",
91                                 'E', 0)
92          << CoinMessageEol;
93      }
94    }
95    delete constraintPool;
96    delete variablePool;
97    return AlpsReturnStatusOk;
98  }
```

In Listing 4.5, the `boundingLoop` function calls pure virtual functions `installSubproblem`, `callHeuristics`, `branchConstrainOrPrice`, `generateConstraints` and `generateVariables`. These pure virtual functions are expected to be implemented by solver developers in their applications.

This might be reminiscent of the callback mechanism of various commercial solvers (CPLEX, Mosek etc.) to the experienced solver developers. Note that BCPS provides much more than mere callback mechanism. It provides a very flexible framework that enables users to implement any kind of tree-based optimization algorithm, not only specific kind (usually MILP in case of commercial solvers). The price developers pay is they need

to input more development than in the case of callback mechanisms. This is necessary since they need to specify details of their optimization problem. BCPS offers much more than the callback mechanisms but also asks for more development effort from the user side. Moreover, BCPS works in parallel since it is built on top of ALPS.

BCPS uses a differencing scheme to store the data that corresponding to a branch-and-bound node (subproblem). BCPS does not store the whole data corresponding to the subproblem. It just stores the difference (cuts and variables added, warm start information etc.) from the parent node. This is by design to use memory efficiently.

BCPS Linear Integer Solver (BLIS)

BLIS [XRV] is a distributed-memory parallel MILP solver built on top of BCPS. BLIS is used as a guide in implementing DisCO.

4.4.2 Discrete Conic Optimization (DisCO) Solver Library

DisCO (Discrete Conic Optimization) is a solver for MISOCP. It is built on top of BCPS. DisCO implements branch-and-cut type of algorithms to solve MISOCP. DisCO depends on many other projects. It depends OsiConic on communicating with its relaxation solvers. It depends on CglConic to cut infeasible solutions. DisCO can use different solvers to solve relaxation subproblems. DisCO uses solvers through their OSI (OsiClp) and OsiConic (OsiIpopt, OsiCplex, OsiMosek) interfaces. DisCO acts as a MILP solver when the problem does not have conic constraints.

This section explains the design and implementation of DisCO in details. We start by explaining the classes that constitute the DisCO.

Main classes of DisCO

DisCO is implemented following object oriented programming and this section introduces the main classes of DisCO.

`DcoModel` represents the problem being solved. Stores problem data and other problem related information generated during the solution process. It inherits and implements `BcpsModel`.

`DcoTreeNode` represents a node of the branch-and-bound tree. Stores a pointer to a `DcoNodeDesc` instance, inherits and implements `BcpsTreeNode`. `DcoNodeDesc` stores information corresponding to the subproblem represented by the node. `DcoNodeDesc` inherits and implements `BcpsNodeDesc`.

`DcoHeuristic` is an abstract base class for heuristic methods. `DcoHeuristic` declares pure virtual functions to find solutions from the current subproblem solution. `DcoHeuristicRounding` implements `DcoHeuristic` for a simple rounding heuristic. This implementation is based on the rounding heuristic of BLIS.

`DcoParams` defines parameters of DisCO solver library. Provides default values for the defined parameters.

`DcoVariable` represents a variable. Inherits and implements `BcpsVariable`.

`DcoConstraint` is an abstract base class for a constraint. Inherits `BcpsConstraint`. `DcoConstraint` inherits pure virtual functions for checking feasibility from `BcpsConstraint`. `DcoLinearConstraint` is a class to represent linear constraints. It implements `DcoConstraint` interface. `DcoConicConstraint` is a class to represent second-order cone constraints. It implements `DcoConstraint` abstract base class.

`DcoBranchStrategyMaxInf`, `DcoBranchStrategyPseudo` and `DcoBranchStrategyStrong` implement maximum infeasibility, pseudo and strong branching strategies respectively. These classes inherit and implement `BcpsBranchStrategy`.

`DcoBranchObject` stores information to perform a branching operation. Each tree

node has a member pointer to a `DcoBranchObject`. This object is created after the node is decided to be branched.

DisCO main function

All that DisCO main function does is to create an instance of a subproblem solver, a DisCO model, an ALPS broker, and call the search function of the broker. The search call looks for solutions and collect them in a pool. Search results and a report of feasibility of the best solution is printed before the main function returns. A simplified version of the main function is given in Listing 4.6.

Listing 4.6: DisCO main function

```
1 int main(int argc, char *argv[]) {
2 #if defined(__OA__)
3   OsiSolverInterface * solver = new OsiClpSolverInterface();
4 #elif defined(__OSI_MOSEK__)
5   OsiConicSolverInterface * solver = new OsiMosekSolverInterface();
6 #elif defined(__OSI_CPLEX__)
7   OsiConicSolverInterface * solver = new OsiCplexSolverInterface();
8 #elif defined(__COLA__)
9   OsiConicSolverInterface * solver = new ColaModel();
10 #endif
11   // Create DisCO model
12   DcoModel model;
13   model.setSolver(solver);
14 #ifdef  COIN_HAS_MPI
15   AlpsKnowledgeBrokerMPI broker(argc, argv, model);
16 #else
17   AlpsKnowledgeBrokerSerial broker(argc, argv, model);
18 #endif
19   // Search for best solution
20   broker.search(&model);
21   broker.printBestSolution();
22   model.reportFeasibility();
23   delete solver;
24   return 0;
25 }
```

The relaxation solver instance is created depending on the algorithm/solver of choice. The algorithm of choice is indicated in the build time and macros indicating the choices are created. DisCO uses the macros created during the build.

Algorithms implemented in DisCO

DisCO implements two main algorithms, bb-lp, which is an LP-based branch-and-bound algorithm, given in Algorithm 5; and bb-socp, which is a branch-and-bound algorithm with socp relaxations. Users are required to specify the algorithm choice during the build. bb-lp is the default algorithm and will be used if no algorithm is specified by the user.

Disco parameters

DisCO provides many parameters to control the algorithms implemented. The most relevant and important parameters are explained in this section. Interested readers can check the DisCO library for an exhaustive list of parameters, including parameters of features that are work in progress.

Users can change the default behavior of DisCO solver library by using the available parameters. Parameters for DisCO solver can be specified in command line when calling the solver or through a file. In command line first parameter name should be specified, then the value for the parameter.

Listing 4.7 demonstrates a use where DisCO reads problem from an MPS input file and solves it using branching strategy 3, i.e., strong branching.

Listing 4.7: Running DisCO with parameters

```
1 disco Alps_instance input_problem.mps Dco_branchStrategy 3
```

Following is the list of parameters that are available when DisCO is used in parallel mode.

`cutRampUp`: Determines whether cuts should be generated during ramp up phase in parallel mode. Default value is `true`.

`branchStrategyRampUp`: Determines the branching strategy to be used during ramp up phase in parallel mode.

`shareConstraints`: In parallel mode, constraints will be sent/received through network if this is `false`. Constraint sharing is not implemented yet.

`shareVariables`: In parallel mode, variables will be sent/received through network if this is `false`. Variable sharing is not implemented yet.

Presolve methods are heuristics to improve the problem formulation. Their main purpose is to modify the problem such that it will be easier and numerically more stable to solve. They achieve this by (1) elimination of variables and constraints, (2) scaling the problem, (3) tightening the variable bounds and constraints. Presolve procedures might yield solutions on easy problem instances. Presolve procedures for DisCO is work in progress and disabled by default. Following are the presolve parameters,

`presolve`: Whether the problem should be presolved first. The default value is `false`.

`presolveNumPass`: Presolve procedure can be applied multiple times by calling the procedure to the new problem obtained from the previous call. presolveNumPass parameter determines the number of presolve calls. The presolve procedure will be called multiple times until this number is hit, or the procedure fails to improve the problem.

DisCO implements max infeasibility, pseudocost and strong branching. The branching strategy is controlled with the following parameters.

`branchStrategy`: Determines the branching strategy to be used.

`pseudoWeight`: Weight used to calculate pseudocosts in case of pseudocost branching.

DisCO has two different classes of cut generation parameters, cut strategy and cut generation frequency. Cut strategy parameter can take one of the following four values, 1

to generate cuts in the root node only, 2 to generate cuts periodically on every specific number of nodes, 3 not to generate cuts and 4 is to let DisCO decide automatically. The specific number of nodes for strategy 2 is determined by parameter `cutGenerationFreq`. It determines the frequency of the cut generation. In strategy 4, DisCO decides whether to generate cuts automatically by judging the previous performance of the generator. Following is the list of cut generation parameters.

These parameters are global parameters for all cut generation strategies. Users can override these for a specific cut procedure (like MIR, Gomory, etc.) by specifying the parameters specific to the procedure. Parameters specific to cut procedures are categorized into two classes, linear and conic. Linear generators can be applied only when the outer-approximation algorithm is used.

Linear cut generation parameters are `cutCliqueStrategy`, `cutGomoryStrategy`, `cutFlowCoverStrategy`, `cutKnapsackStrategy`, `cutMirStrategy`, `cutOddHoleStrategy`, `cutKnapsackStrategy`, `cutMirStrategy`, `cutOddHoleStrategy`, `cutProbingStrategy`, `cutCliqueFreq`, `cutGomoryFreq`, `cutFlowCoverFreq`, `cutKnapsackFreq`, `cutMirFreq`, `cutProbingFreq`.

Conic cut generation parameters are `cutIpmStrategy`, `cutIpmIntStrategy`, `cutOaStrategy`, `cutIpmFreq`, `cutIpmIntFreq`, `cutOaFreq`.

DisCO implements a simple rounding heuristic that is effective in case of outer-approximation algorithm. Heuristic strategies are controlled similar to cut generation strategies. There are 4 different strategies, `heurStrategy` is set to 1 to call heuristic routines in the root node only, 2 to call heuristics periodically on every specific number of nodes, 3 to do not call heuristics and 4 to let DisCO decide when to call heuristics. Following is the list of heuristic related parameters of DisCO.

`heurStrategy`: Global heuristic strategy.

`heurCallFrequency`: Global heuristic call frequency.

`heurRoundStrategy`: Rounding heuristic strategy.

`heurRoundFreq`: Rounding heuristic call frequency.

A parameter that is used in outer-approximation algorithm is the number of approximation passes. DisCO generates OA cuts to improve the outer approximation of the problem being solved. Parameter `approxNumPass` controls the number of times OA cut generation procedures are called. DisCO might stop this procedure in fewer iterations if no cuts are generated. Default value for `approxNumPass` is 400.

Another important parameter is `logLevel`, that controls the information printed to standard output. There are 5 log levels from 0 to 4. 0 is the least verbose and 4 is the maximum. `logLevel` parameter can also be used for debugging purposes. DisCO log messages are implemented using messaging framework provided in CoinUtils. DisCO uses bit masking for debugging. `logLevel` 8 prints information related to branching process, 16 for cut procedures, 32 for node processing and 64 for presolving. 32 is also used for MPI, GrUMPy and heuristics debugging. A user that want to enable all debug output should use level 255.

Following is the list of DisCO parameters related to optimality.

`objTol`: This parameter is used in fathoming the unpromising nodes. Subproblems with bounding value less than incumbent solution value plus `objTol` are fathomed. Default value of it is 1.0e-6.

`optimalRelGap`: The search stops when the relative gap between lower and upper bounds fall below this value. Default value of this parameter is 1.0e-6.

`optimalAbsGap`: The search stops when the absolute gap between lower and upper bounds fall below this value. Default value of this parameter is 1.0e-4.

Following are the DisCO parameters on measuring feasibility.

`integerTol`: Tolerance to accept a number as an integer. If its distance to the closest integer is less than `integerTol` the number is accepted as an integer. Default value of `integerTol` is 1.0e-5.

`coneTol`: Tolerance to accept a given point as feasible to a given conic constraint. Default value of `coneTol` is 1.0e-5.

DisCO accepts MPS files as input. MPS files do not indicate the direction of optimization. `objSense` parameter of DisCO can be used to specify this. Value 1 indicate minimization and -1 indicates maximization. DisCO assumes minimization if the parameter is not specified by the user.

In addition to DisCO parameters introduced in this section, ALPS parameters can also be used for changing the behavior of the tree search.

Visualizing DisCO Algorithms with GrUMPy

GrUMPy (Graphics for Understanding Mathematical Programming in Python) given by Bulut and Ralphs [BR] and Ozaltin, Hunsaker, and Ralphs [OHR07] is a Python (Python Software Foundation, https://www.python.org/) package for visualizing problems and algorithms of mathematical optimization. GrUMPy can be used to visualize the solution process of DisCO. For specific debug levels that can be set using `logLevel` parameter DisCO writes branch-and-bound tree information to the standard output. Grumpy can be used to read this output and generate visualizations of the branch and bound tree as DisCO prints them during the execution. The visualizations are generated and displayed in real time during the execution of the DisCO solver.

Figure 4.5 to 4.11 display branch-and-cut tree images generated by GrUMPy for problem instance estein5_A. The figures are generated at every 500 DisCO's GrUMPy specific log lines. These figures are generated with bb-lp algorithm for instance estein5_A. Each node in the branch and cut tree is represented as a dot in the figures. Horizontal place-

ment of each node is decided based on the optimal objective value of the subproblem corresponding to the node. estein5_A is a minimization problem, hence, the deeper the nodes in the tree, the larger the optimal objective values. The nodes are colored depending on their statuses. Yellow indicates that the node is branched. Green indicates that the node is promising, i.e., candidate subproblem to be solved. Infeasible nodes are colored as brown. Nodes that resulted feasible solutions are colored as light blue. A red line indicates the upper bound of the problem at the time of snapshot. Fathomed nodes are indicated by red color. Branch-and-cut algorithm stops when there is no candidate problem with a promising objective value, i.e., no green dots above red line. These figures are drawn in a guided user interface window while DisCO is working on the problem. Note that GrUMPy and DisCO should not necessarily run in the same machine to generate these figures. DisCO might run on a remote high processing power server where the output is piped through network into a laptop where GrUMPy is running and animating the tree.

4.5 Conclusion

We introduced mathematical optimization software used to implement the algorithms proposed in Chapter 2 and Chapter 3 in this chapter. These software libraries are open source and fit with the current line of work developed under the COIN-OR organization for the last decade. The libraries are designed to be modular, flexible and easy to maintain. The resulting discrete conic optimization solver is obtained by combining solver interfaces, implementations of these interfaces, libraries for generating valid inequalities and the source code of the solver itself. DisCO is flexible to work with various commercial and open source solvers to solve relaxation subproblems. Moreover it is flexible enough to work with either SOCP or LP based subproblems.

The DisCO tree search is built on top of COIN-OR's CHiPPS framework. CHiPPS is a

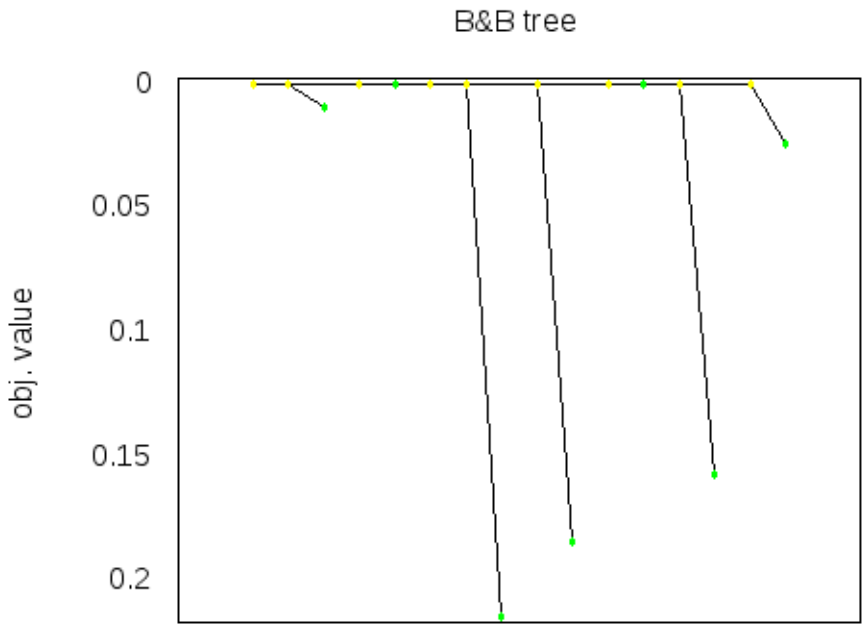


Figure 4.5: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 1

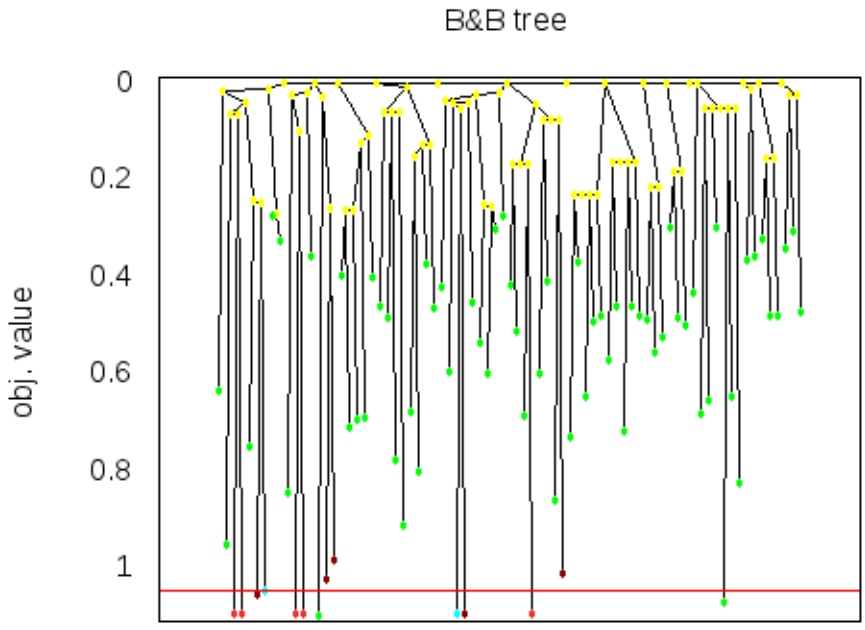


Figure 4.6: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 2

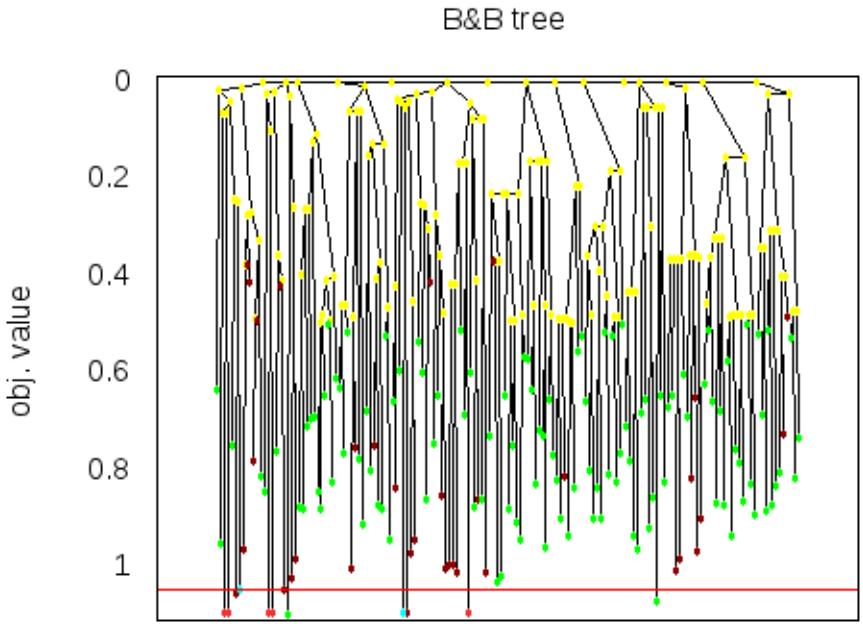


Figure 4.7: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 3

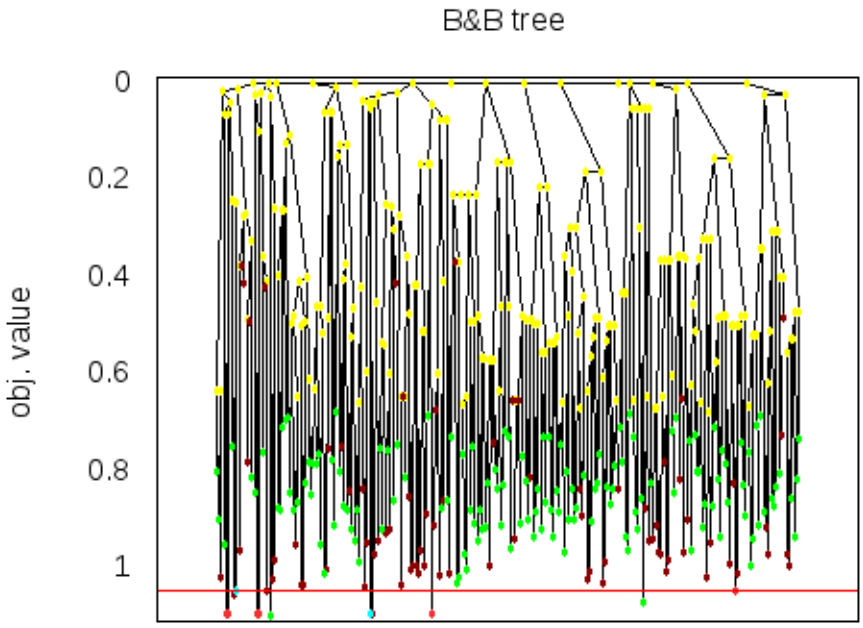


Figure 4.8: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 4

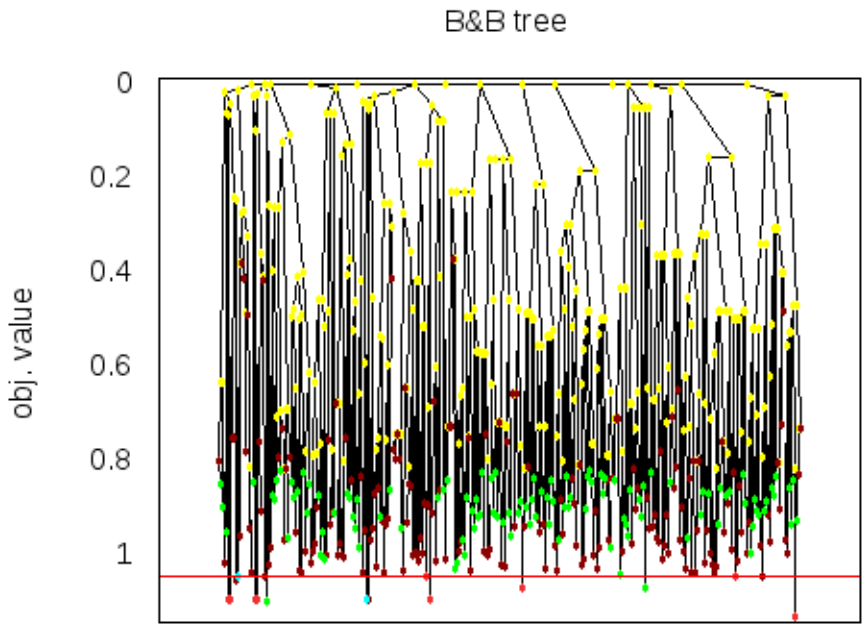


Figure 4.9: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 5

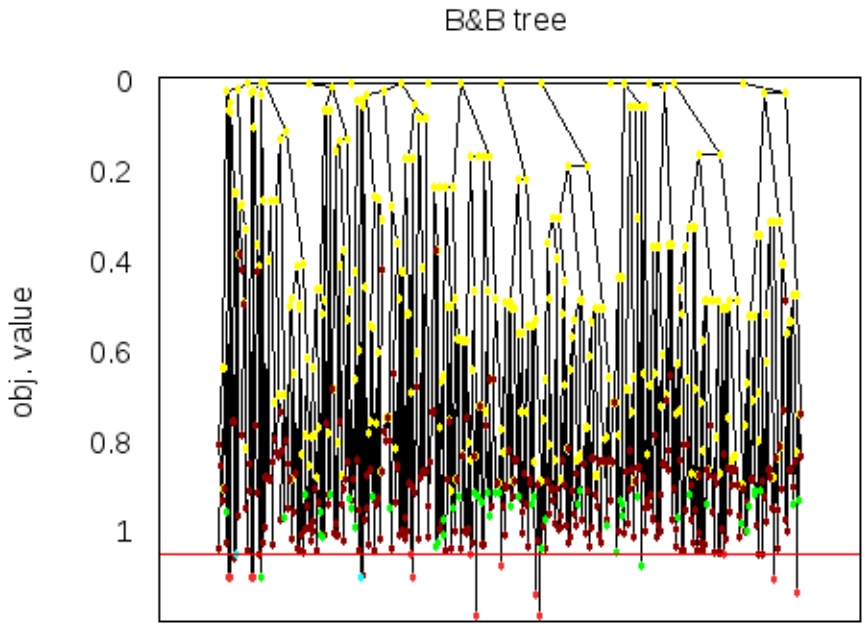


Figure 4.10: estein5_A branch-and-bound tree of DisCO generated by GrUMPy – 6

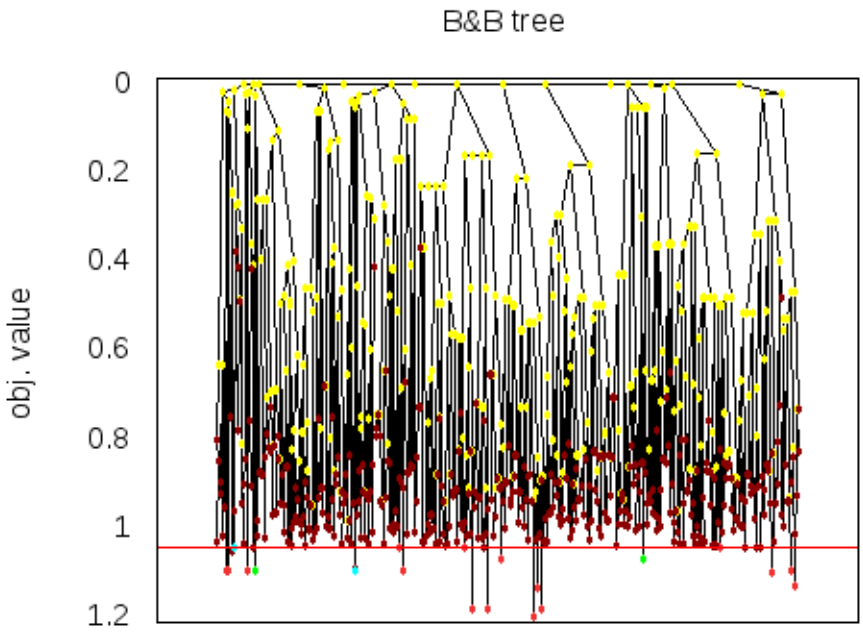


Figure 4.11: estein5_A branch-and-bound tree of DisCO generated GrUMPy – 7

collection of libraries to build parallel optimization solvers. With CHiPPS's parallelization capability DisCO works distributed memory parallel. Chapter 6 contains computational experiments that test the performance of DisCO with various parameters and solvers both in serial and parallel.

DisCO comes with extensive debugging capabilities. The user can obtain debugging information related to branching, cutting, bounding etc. processes by using special parameters. Moreover DisCO works with GrUMPy, a python package for mathematical optimization, to visualize the information related to the branch and cut trees.

Chapter 5

Inverse MILP

5.1 Introduction

Optimization problems arise in many fields and the literature abounds with techniques for solving various classes of such problems. In general the goal of optimization is to determine a member of a given feasible set (an *optimal solution*) that minimizes the value of a given *objective function*. The feasible set is typically described as the points in a vector space satisfying a given set of equations, inequalities, and disjunctions (the latter are usually in the form of a requirement that the value of a certain element of the solution take on an integral value).

An *inverse optimization problem*, in contrast, is a related problem in which the description of the original optimization problem, which we refer to as the *forward problem*, is not complete (some parameters are missing or cannot be observed), but a full or partial solution *can* be observed. The goal is to determine values for the missing parameters with respect to which the given solution would be optimal for the resulting problem. Estimates for the missing parameters may be given, in which case the goal is to produce a set of parameters that is as "close" to the given estimates as possible.

5.1.1 Formal Definitions

The optimization problem of interest in this paper is the *mixed integer linear optimization problem*

$$\min_{x \in \mathcal{S}} d^\top x \tag{MILP}$$

where $d \in \mathbb{Q}^n$ and

$$\mathcal{S} = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\} \cap (\mathbb{Z}^r \times \mathbb{R}^{n-r}).$$

for $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$.

One can define a number of different inverse problems associated with (MILP), depending on what parts of the description (A, b, d) are unknown. Here, we study the case in which the objective function d is unknown, but we are given A and b, as well as an estimate $c \in \mathbb{Q}^n$ of the true objective d and a solution $x^0 \in \mathbb{Q}^n$. Formalizing the statement of this problem requires a careful attention to details that we now highlight by discussing several candidate formulations for this inverse problem.

We first consider the following formulation of the inverse problem as a semi-infinite optimization problem:

$$\begin{aligned} \min \quad & \|c - d\| \\ \text{s.t.} \quad & d^\top x^0 \leq d^\top x \quad \forall x \in \mathcal{S}, \end{aligned} \tag{INVMILP}$$

where $\|\cdot\|$ can be any norm. In (INVMILP), d is the unspecified vector to be determined (and is thus a vector of variables here rather than being fixed), while $c \in \mathbb{Q}^n$ is the estimate or target value. Note that in (INVMILP), if we instead x^0 vary, replacing it with a variable x, and interpret d as a fixed objective function, replacing $\|c - d\|$ with the objective $d^\top x$ of the forward problem, we get a reformulation of the forward problem (MILP) itself.

Problem (INVMILP) can also be re-formulated as a conic problem. In terms of the conic sets

$$\mathcal{K}(y) = \{\alpha d \in \mathbb{R}^n : \|c - d\| \leq y, \alpha > 0, \alpha \in \mathbb{R}\} \text{ and}$$
$$\mathcal{D} = \{d \in \mathbb{R}^n : d^\top (x^0 - x) \leq 0 \; \forall x \in \mathcal{S}\},$$

(INVMILP) can be reformulated as

$$\min_{d \in \mathcal{K}(y) \cap \mathcal{D}} y. \tag{INVMILP-C}$$

The set $\mathcal{D}$ can be interpreted either as the set of objective function vectors for which x^0 is preferred over all points in $\mathcal{S}$ or, alternatively, as the set of all hyperplanes containing x^0 that define inequalities valid for $\mathcal{S}$. The latter interpretation leads to a third formulation in terms of the so-called 1-polar. For a given polyhedron $\mathcal{P}$, e.g., conv($\mathcal{S}$), the 1-polar is defined as

$$\mathcal{P}^1 = \{\pi \in \mathbb{R}^n \mid \pi^\top x \geq 1, \forall x \in \mathcal{P}\},$$

assuming that $\mathcal{P}$ is a polytope. When $\mathcal{P}$ is full-dimensional and $0 \in \mathcal{P}$ (this latter requirement is without loss of generality by translating $\mathcal{P}$), the 1-polar is the normalized set of all inequalities valid for $\mathcal{P}$ (see [Sch86] for formal definitions). Under these assumptions, (INVMILP) can also be reformulated as

$$\begin{aligned} \min \quad & \|c - d\| \\ \text{s.t.} \quad & \pi \in \mathcal{P}^1 \\ & \pi^\top x^0 \leq 1 \\ & d = \alpha \pi \\ & \alpha \in \mathbb{R}_+. \end{aligned} \tag{INVMILP-1P}$$

In formulation (INVMILP-1P), the constraint $d = \alpha\pi$ allows d to be scaled in order to improve the objective function value. We might also require $\|c\| = 1$ or normalize in some other way to avoid this scaling. The constraint $\pi^\top x^0 \leq 1$ ensures that d is feasible to (INVMILP). Observe also that relaxing the constraint $\pi^\top x^0 \leq 1$ yields a problem something like the classical separation problem, but with a different objective function. We revisit this idea in Section 5.2.

We have so far avoided an important point and that is what assumptions we make about the point x^0. On the one hand, the problem, as informally stated, can only have a solution if $x^0 \in \mathcal{S}$, since otherwise, x^0 cannot be optimal for any objective function. On the other hand, the formulations above can be interpreted whether or not $x^0 \in \mathcal{S}$. As a practical matter, this subtle point is not very important, since membership in $\mathcal{S}$ can be verified in a pre-processing step if necessary. However, in the context of formal complexity analysis, this point *is* important and we will return to it. For now, we do not assume $x^0 \in \mathcal{S}$, in which case d can be more accurately interpreted as specifying a valid inequality which is satisfied at equality by x^0.

In order to paint a complete picture, there is one other pathological case to be considered and that is when x^0 is in the (relative) interior of conv($\mathbb{S}$). In this case any objective vector in the subspace orthogonal to the affine space containing conv($\mathbb{S}$) is feasible for the inverse problem, i.e., optimizes x^0. Define $c_{\mathbb{S}}$ as the projection of c onto affine space that contains $\mathbb{S}$. Define $c_{\mathbb{S}}^\perp$ as the projection of c onto orthogonal subspace. Then $c = c_{\mathbb{S}} + c_{\mathbb{S}}^\perp$ and $c_{\mathbb{S}} \perp c_{\mathbb{S}}^\perp$. When conv($\mathbb{S}$) is full dimensional, then $c = c_{\mathbb{S}}$, $d^* = c_{\mathbb{S}}^\perp = 0$ and optimal value to inverse problem is $\|c_{\mathbb{S}}\| = \|c\|$. When c is in the orthogonal subspace, then $c = c_{\mathbb{S}}^\perp$, $d^* = c_{\mathbb{S}}^\perp$ and optimal value to inverse problem is 0.

When x^0 is in the (relative) interior of conv($\mathbb{S}$), inverse problem reduces to finding the closest point to c in orthogonal subspace. Optimal value for d in this case is the projection of c onto orthogonal subspace, i.e., $d^* = c_{\mathbb{S}}^\perp$. Optimal objective value of inverse problem

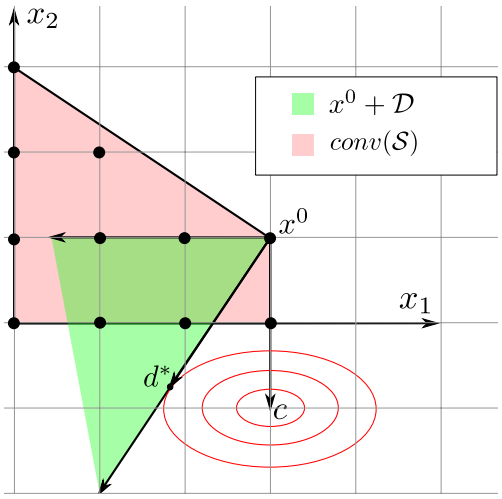


Figure 5.1: Two dimensional inverse MILP

is $\|c_{\mathbb{S}}\|$.

In general (no assumption about position of x^0), optimal value of inverse problem is bounded by 0 from below and $\|c_{\mathbb{S}}|$ from above.

Figure 5.1 demonstrates the inverse MILP geometrically. $\mathcal{S}$ is a discrete set indicated by the black dots. The vector $c = (0, -2)$ and $x^0 = (3, 1)$. The convex hull of $\mathcal{S}$ and the cone $\mathcal{D}$ (translated to x^0) are shaded. The ellipsoids show the sets of points with a fixed distance to $x^0 + c$ for some given norm. The optimal objective function in this example is vector d^*, and the point indicated in the figure is $x^0 + d^*$.

5.1.2 Previous Work

There are a range of different flavors of inverse optimization problem. The inverse problem we investigate is to determine objective function coefficients that make a given solution optimal, but other flavors of inverse optimization include constructing a missing part of either the coefficient matrix or the right-hand side that makes a given solution optimal. The work presented here is based on Bulut and Ralphs [BR15].

Heuberger [Heu04] provides a detailed survey of inverse combinatorial optimization

problems. In this paper, different types of inverse problems, including types for which the inverse problem seeks parameters other than objective function coefficients, are examined. A survey of solution procedures for specific combinatorial problems is provided, as well as a classification of the inverse problems that are common in the literature. According to this classification, the inverse problem we study in this paper is an *unconstrained*, *single feasible object*, and *unit weight norm* inverse problem. Our results can be straightforwardly extended to some related cases, such as multiple given solutions.

Cai, Yang, and Zhang [CYZ99] examine an inverse center location problem in which the aim is to construct part of the coefficient matrix, in this case the distances between nodes from a given optimal solution. It is shows that even though the center location problem is polynomially solvable, this particular inverse inverse problem is NP–hard. This is done by way of a polynomial transformation of the satisfiability problem to the decision version of the inverse center location problem. This analysis indicates that the problem of constructing part of the coefficient matrix is harder than the forward version of the problem.

Huang [Hua05] examines the inverse knapsack problem and inverse integer optimization problems. In this paper, a pseudo–polynomial algorithm for the inverse knapsack problem is presented. It is also shown that inverse integer optimization with a fixed number of constraints is pseudo–polynomial by transforming the inverse problem to a shortest path problem on a directed graph. When the number of constraints are fixed, this results a pseudo–polynomial algorithm for inverse integer optimization.

Schaefer [Sch09] studies general inverse integer optimization problems. Using superadditive duality, a polyhedral description of the set of all feasible objective functions is derived. This description has only continuous variables but an exponential number of constraints. A solution method using this polyhedral description is proposed. Finally, Wang [Wan09] suggests a cutting-plane algorithm similar to the one suggested herein

and presents computational results on several test problem with an implementation of this algorithm.

The case when the feasible set is an explicitly described polyhedron is well–studied by Ahuja and Orlin [AO01]. In their study, they analyze the shortest path, assignment, minimum cut, and minimum cost flow problems under the l_1 and l_∞ norms in detail. They also conclude that inverse optimization problem is polynomially solvable when the forward problem is polynomially solvable. The present study aims to generalize the result of Ahuja and Orlin [AO01] to the case when the forward problem is not necessarily polynomially solvable, as well as to make connections to other well-known problems.

In the remainder of the paper, we first introduce address the computational complexity of (INVMILP). As written, this is a semi-infinite program, but it is easy to see that we can replace the infinite set of constraints with a finite set corresponding to the extreme points of conv($\mathcal{S}$). This still leaves us with what is ostensibly a nonlinear objective function. We show in Section 5.2 that for the ℓ_∞ and ℓ_1 norms, this problem can be expressed as a standard *linear optimization problem* (LP), albeit one with an exponential number of constraints. The reformulation can be readily solved in practice using a standard cutting-plane approach. On the other hand, we show in Section 5.3 that the formal complexity does not depend on the norm.

5.2 Algorithmic Approach to Inverse MILP

We now show how to formulate (INVMILP) explicitly for two common norms using standard techniques for linearization. The objective function of an inverse MILP under the l_1 norm can be linearized by the introduction of variable vector θ, and associated constraints

as

$$
\begin{aligned}
z_1^{-1} = \quad & \min \quad y \\
\text{s.t.} \quad & y = \sum_{i=1}^{n} \theta_i \\
& c_i - d_i \leq \theta_i \quad \forall i \in \{1, 2, \ldots, n\} \\
& d_i - c_i \leq \theta_i \quad \forall i \in \{1, 2, \ldots, n\} \\
& d^\top x^0 \leq d^\top x \quad \forall x \in \mathcal{S}.
\end{aligned}
\tag{INVMILP-1}
$$

The objective function of inverse MILP under l_∞ norm can be linearized by the introduction of variable y and two sets of constraint sets as

$$
\begin{aligned}
z_\infty^{-1} = \quad & \min \quad y \\
\text{s.t.} \quad & c_i - d_i \leq y \quad \forall i \in \{1, 2, \ldots, n\} \\
& d_i - c_i \leq y \quad \forall i \in \{1, 2, \ldots, n\} \\
& d^\top x^0 \leq d^\top x \quad \forall x \in \mathcal{S}.
\end{aligned}
\tag{INVMILP-∞}
$$

This formulation is a continuous problem, but is a semi-infinite program when written in the form above, as mentioned earlier.

To obtain a finite problem, we can limit the set of constraints to only those involving the finite set of extreme points and rays of $\text{conv}(\mathcal{S})$. Although this yields a finite formulation, the number of extreme points and rays may still be very large and it is not practical to write this formulation explicitly via a priori enumeration.

A better approach is to use a separation–optimization procedure and generate these inequalities dynamically, as suggested by Wang [Wan09]. This is a natural application of the separation–optimization procedure described in, e.g., Grötschel, Lovász, and Schrijver

[GLS93]. Although this approach has already been described in the literature, our purpose in describing it again here is is to make the connection to a similar existing algorithm for solving the standard separation problem, which provides the intuition behind the complexity results to be presented in Section 5.3.

The form of (INVMILP-1) and (INVMILP-∞) makes it clear that these two formulations, although of exponential size, can be solved by a standard cutting-plane approach. We describe such a cutting-plane algorithm for the case of the l_∞ norm (formulation (INVMILP-∞)) and note that a similar algorithm can be applied to the model (INVMILP-1) for the case of the l_1 norm.

First, let us define two parametric problems, P_k and $InvP_k$, as follows:

$$\min_{x \in \mathcal{S}} d^{k\top} x \tag{P_k}$$

$$\begin{aligned} \min \quad & y & & \\ \text{s.t.} \quad & c_i - d_i \leq y & \forall i \in \{1, 2, \ldots, n\} & \\ & d_i - c_i \leq y & \forall i \in \{1, 2, \ldots, n\} & \\ & d^\top x^0 \leq d^\top x & \forall x \in \mathcal{E}^k. & \end{aligned} \tag{$InvP_k$}$$

where $\mathcal{E}^k$ is the set of solutions found by solving $P_1, \ldots, P_{k-1}$. Note that (P_k) is an MILP with the same feasible region as the original forward problem (MILP), but with objective function d^k. This is precisely the problem of separating d^k from the feasible region of (INVMILP). $(InvP_k)$ is the relaxation of MILP (INVMILP-∞) considering only valid inequalities that correspond to solutions to the forward problem contained in $\mathcal{E}^k$.

The overall procedure is given in Algorithm 9. In this algorithm, we solve an instance of the forward problem in each iteration in order to generate a cut. The algorithm stops

when the current d^k is feasible. When (P_k) is unbounded, then $d = 0$ is an optimal

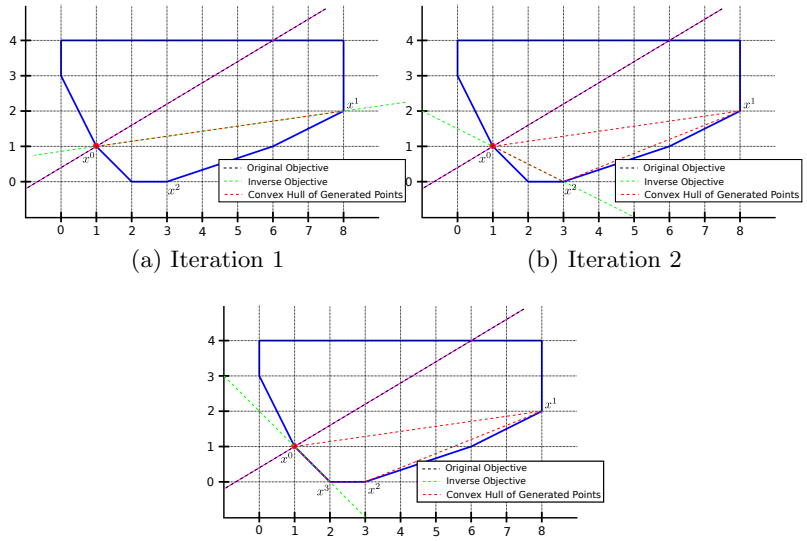
Algorithm 9 Cutting-plane method for inverse MILP under l_∞ norm

$k \leftarrow 0$, $\mathcal{E}^1 \leftarrow \emptyset$.
do
  $k \leftarrow k + 1$.
  Solve $(InvP_k)$, $d^k \leftarrow d^*$.
  Solve (P_k).
  if (P_k) unbounded **then**
    $y^* \leftarrow \|c\|_\infty$, $d^* \leftarrow 0$, STOP.
  else
    $x^k \leftarrow x^*$.
  end if
  $\mathcal{E}^{k+1} \leftarrow \mathcal{E}^k \cup \{x^k\}$.
while ${d^k}^\top(x^0 - x^k) > 0$
$y^* \leftarrow \left\|c - d^k\right\|_\infty$, $d^* \leftarrow d^k$, STOP.

solution, since this shows that only $d = 0$ satisfies $d^\top(x^0 - x) \leq 0$ for all x in $\mathcal{S}$.

Before illustrating with a small example, we would like to again point out the close relationship of the inverse problem and the separation problem. First, note that another way of interpreting $(InvP_k)$ is as the problem of generating an inequality valid for $\operatorname{conv} \mathcal{E}^k$ and for which the associated hyperplane, $\{x \in \mathbb{R}^n \mid {d^k}^\top x = {d^k}^\top x^0\}$, contains x^0. In this case, (P_k) can then be interpreted as the problem of determining whether there is an $x^k \in \mathcal{S}$, such that ${d^k}^\top x^k < {d^k}^\top x^0$, i.e., is violated by the associated valid inequality. This shows both that the inequality is not valid for $\operatorname{conv}(\mathcal{S})$ and that d^k is not feasible for (INVMILP). Figure 5.2 illustrates how the algorithm might proceed for an example where the set $\mathcal{S}$ is the integer points inside the blue polyhedron.

Algorithm 9 can be easily modified to solve the generic separation problem for $\operatorname{conv}(\mathcal{S})$ by interpreting x^0 as the point to be separated and replacing the objective function (and associated auxiliary constraints) of $(InvP_k)$ with one measuring the degree of violation of x^0. In this case, $(InvP_k)$ can be interpreted as the problem of separating x^0 from $\operatorname{conv} \mathcal{E}^k$. Roughly, the dual of $(InvP_k)$ is to determine whether x^0 can be expressed as a convex



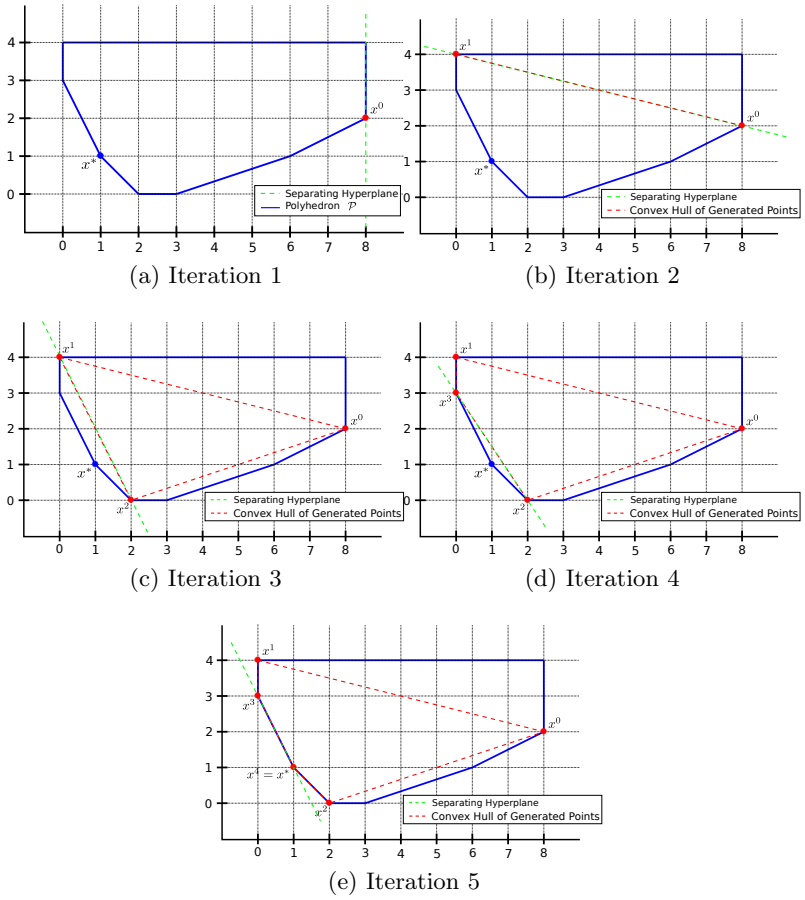
(a) Iteration 1 (b) Iteration 2

(c) Iteration 3

Figure 5.2: Pictorial illustration of Algorithm 9

combination of the members of $\mathcal{E}^k$. If not, then the proof is a separating hyperplane, which is an inequality valid for conv $\mathcal{E}^k$. As in the inverse case, (P_k) is interpreted as the problem of determining whether there is an $x^k \in \mathcal{S}$ that is violated by the associated valid inequality. The generated valid inequalities are sometimes called *Fenchel cuts* [Boy94]. Figure 5.3 illustrates how the algorithm for generating Fenchel cuts might proceed for for the same polyhedron as in Figure 5.2.

A Small Example: Let $c = (-2, 1)$, $x^0 = (0, 3)$ and $\mathcal{S}$ given as in Figure 5.4 where both x_1 and x_2 are integer and convex hull of $\mathcal{S}$ is given. k, d^k and x^k values through iterations are given in Table 5.1.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

(e) Iteration 5

Figure 5.3: Pictorial illustration of algorithm for generating Fenchel cut

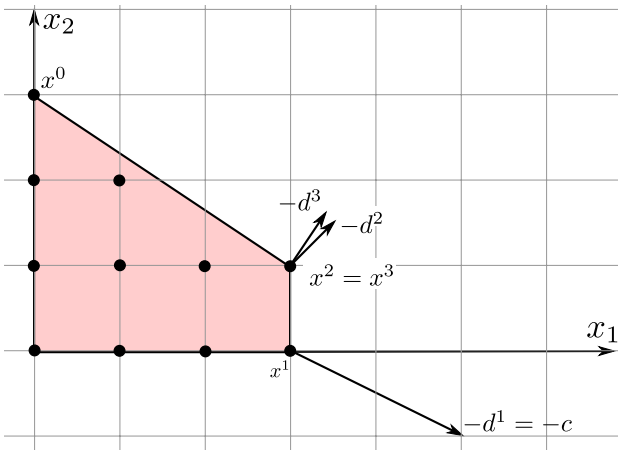


Figure 5.4: Feasible region and iterations of example problem

Table 5.1: k, d^k, x^k and $\mathcal{E}^k$ values through iterations

	k	$\mathcal{E}^k$	d^k	x^k	$\|d^k - c\|_\infty$
initialization	1	$\emptyset$	$(-2, 1)$	$(3, 0)$	0
iteration 1	2	$\{(3, 0)\}$	$(-0.5, -0.5)$	$(3, 1)$	1.5
iteration 2	3	$\{(3, 0), (3, 1)\}$	$(-0.4, -0.6)$	$(3, 1)$	1.6

Inverse MILP optimal value is $y^* = \|c - d^3\|_\infty = 1.6$. Inverse MILP optimal solution is $d^3 = (0.4, 0.6)$.

5.3 Complexity of Inverse MILP

In what follows, we discuss inverse MILP in the traditional framework of computational complexity theory. See Section 1.3.2 for a brief review of computational complexity theory.

5.3.1 Complexity of MILP

It will be convenient to refer in what follows to several decision problems associated with forward problem (MILP) and inverse problem (INVMILP). The most commonly associated decision version of forward problem (MILP) is a feasibility problem involving an extra scalar parameter γ, as follows.

Definition 5.3.1. ***MILP decision problem (MILPD)****: Given $\gamma \in \mathbb{Q}$, $d \in \mathbb{R}^n$, and an MILP with feasible region $\mathcal{S}$, does there exist $x \in \mathcal{P}$ such that $d^\top x \leq \gamma$?*

It is well-known that this problem is in the complexity class NP–complete and that the optimal solution of problem (MILP) can be determined with a polynomial number of calls to an NP oracle (MILPD) using bisection search. The formal *input* to this decision problem is the quintuplet (A, b, d, r, γ) and the set of such inputs that yields the answer

YES is the *language* recognized by an algorithm for solving this problem (formally specified as a Turing machine).

It is useful to recall that a well-known characterization of the class NP is as the class of problems for which the is a *short certificate* for the YES answer. Roughly speaking, a *certificate* is a proof that the answer is correct. A short certificate is one that can be verified in polynomial time. In the case of MILPD, the short certificate is any feasible solution.

The complement of NP is the class coNP of problems for which there is a short certificate for the NO answer. The problem of determining whether a given γ is a lower bound on the value of an optimal solution is an example of a decision problem that is in the class coNP.

Definition 5.3.2. ***MILP lower-bounding problem (MILPL)****: Given $\gamma \in \mathbb{Q}$, $d \in \mathbb{R}^n$, and an MILP with feasible region $\mathcal{S}$, is $\min_{x \in \mathcal{P}} d^\top x \geq \gamma$?*

When the answer is NO, a feasible solution in $\mathcal{S}$ with an objective value strictly less than γ is a short certificate. The question of whether a given point is in $\text{conv}(\mathcal{S})$ (membership problem) is equivalent to MILPD. Similarly, asking whether a given inequality is valid (validity problem) for $\text{conv}(\mathcal{S})$ is equivalent to MILPL. The validity problem is itself a membership problem over the 1-polar.

Finally, we consider a third decision problem mentioned earlier, which is that of determining whether the optimal solution value is *exactly* γ.

Definition 5.3.3. ***MILP optimal value verification problem (MILPV)****: Given $\gamma \in \mathbb{Q}$, $d \in \mathbb{R}^n$, and an MILP with feasible region $\mathcal{S}$, is $\min_{x \in \mathcal{P}} d^\top x = \gamma$?*

This problem is in the class D^P of problems defined by Papadimitriou and Yannakakis [PY82]. Complexity class D^P is the class of problems for which the language to be recognized is the intersection of two languages one in class NP and the other in class coNP.

Papadimitriou and Yannakakis [PY82] showed that MILPV is complete (and (MILP) is hard) for the class D^{P}.

A related problem is deciding whether a given point is on the boundary of conv($\mathcal{S}$). It is also in D^{P} since it is equivalent to verifying optimal value.

5.3.2 Complexity of Inverse MILP

Ahuja and Orlin [AO01] show that the inverse problem can be solved in polynomial time when the forward problem is polynomially solvable.

Theorem 5.3.4. *([AO01]) If a forward problem is polynomially solvable for each linear cost function, then the corresponding inverse problems under l_1 and l_∞ norms are polynomially solvable.*

They use the well-known result of Grötschel, Lovász, and Schrijver [GLS93] to conclude that inverse LP, in particular, is polynomially solvable. Note that this result already indicates that if a given MILP is polynomially solvable, then the associated inverse problem is also polynomially solvable. On the other hand, for general MILPs, the result of Grötschel, Lovász, and Schrijver [GLS93] cannot tell us precisely what complexity class the inverse problem is in for general MILPs, since the result is about polynomial equivalence, not complexity class equivalence, as we describe in more detail below. The main contribution of this study is to provide the theory that places the inverse problem in the tightest possible class without resolving P versus NP problem. To obtain a formal complexity result, we first consider the decision version of the inverse problem. The decision version is derived in a fashion similar to that of *MILPD*. It asks whether a solution with an objective value less then some given threshold exists.

Definition 5.3.5. ***Inverse MILP decision problem (INVD)**: Given $\gamma \in \mathbb{Q}$, $c \in \mathbb{R}^n$, $x^0 \in \mathbb{R}^n$, and polyhedron $\mathcal{S} \subseteq \mathbb{R}^n$, is the set $\mathcal{K}(\gamma) \cap \mathcal{D}$ non-empty?*

The result of Grötschel, Lovász, and Schrijver [GLS93] bounds the running time for optimizing a linear objective function over an implicitly defined polyhedron in terms of calls to a separation oracle. Their result can be stated as follows.

Theorem 5.3.6. *([GLS93]) Given an oracle for the separation problem, the optimization problem over a given polyhedron with linear objective can be solved in time polynomial in φ, n and the encoding length of objective coefficient vector, where φ is the facet complexity of the given polyhedron.*

A polyhedron has facet-complexity at most φ if there exists a rational system of inequalities describing the polyhedron in which the encoding length of each inequality is at most φ. The facet complexity thus measures the complexity of a polyhedron independent of its representation. Theorem 5.3.6 indicates that, given an oracle for inverse MILP separation, the inverse MILP optimization problem can be solved in time polynomial in φ and n, where the feasible set of (INVMILP-∞) has facet-complexity at most φ, since the objective function of (INVMILP-∞) has an encoding length polynomial in n.

To find a bound on φ, consider the third set of constraints of the formulation (INVMILP-∞). The encoding length of the first two sets of constraints depends on the maximum encoding length of c_i, $i \in 1, \ldots, n$. The encoding length of the third set of constraints depends on the encoding length of x^0 and the largest encoding length of any extreme point of the convex hull of $\mathcal{S}$. This latter quantity is known as the *vertex complexity* of $\mathcal{S}$ and is a related measure of the complexity of a polyhedron that is bounded by a polynomial function of the facet complexity. Thus, we can say that the running time of the separation–optimization algorithm is polynomial in the encoding length of c_i for $i = 1, \ldots, n$, x^0 and the vertex complexity of the convex hull of $\mathcal{S}$. Note that in the case of binary integer optimization problems, the vertex complexity of $\text{conv}(\mathcal{S})$ is always polynomial in n.

These conclusions can be interpreted as stating that the inverse MILP separation

problem is equivalent to the MILP optimization problem, but it is important to note that this equivalence is only a polynomial equivalence, not a complexity-wise equivalence. The MILP optimization problem can be solved in polynomial time, given an oracle for the MILP decision problem. Similarly, we conclude that the inverse MILP optimization problem can be solved in polynomial time, given an oracle for the MILP decision problem, which we know to be NP–complete. The following theorem summarizes this result.

Theorem 5.3.7. *The inverse MILP optimization problem under the l_∞/l_1 norms is solvable in time polynomial in φ and n, given an oracle for the MILP decision problem.*

This theorem hints at the complexity of inverse optimization problem. We now know that Algorithm 9 solves inverse MILP in polynomial time, given an NP oracle. This algorithm can be used to solve the decision version. In complexity terms. this shows that the inverse problem is in Δ_2^{P}. The following is the restatement of Theorem 5.3.7 in complexity terms.

Theorem 5.3.8. *INVD under l_1 and l_∞ norms is in Δ_2^{P}.*

The next natural question that comes to mind is whether INVD is complete for this class. Somewhat surprisingly (though not in hindsight), the answer is no. This indicates that GLS result does not yield the tightest complexity class.

The first main result of this paper is the following theorem that shows INVD with an arbitrary norm (l_1, l_∞, or any other p-norm) is in coNP.

Theorem 5.3.9. *INVD is in* coNP.

Proof We show existence of a short certificate when the answer to INVD problem is NO. Note that when answer is NO then $\gamma < \|c\|$ (in fact $\gamma < \|c_{\mathbb{S}}\|$, but this is OK for the proof), since $d = 0$ is a valid solution otherwise. Furthermore, when $\gamma = 0$ the problem reduces to MILPL (is $c^\top x^0$ a lower bound for minimization along c over $\mathcal{S}$),

which is already known to be in coNP. Therefore it is enough to consider the case where $0 < \gamma < \|c\|$.

When the answer to INVD is NO, then, for each d in $\mathcal{K}(\gamma)$, there exists an x in $\mathcal{S}$ such that $d^\top(x - x^0) < 0$. Hence, the NO answer can be validated by enumeration over $\mathcal{S}$ in principle. What we will show is that we do not need to check the inequality for all x in $\mathcal{S}$, but only for a subset of polynomial size. For this we define the following set first,

$$\mathcal{X}(\gamma) = \{x \in \mathcal{S} \,|\, \exists d \in \mathcal{K}(\gamma) \text{ s.t. } d^\top(x - x^0) < 0\}.$$

$\mathcal{X}(\gamma)$ is the set of points in $\mathcal{S}$ that are better than x^0 for at least one direction d in $K(\gamma)$. Note that set $\mathcal{X}(\gamma)$ is not empty since answer to problem is NO. Moreover since $\mathcal{X}(\gamma)$ is a subset of $\mathcal{S}$, it is a discrete set. We define another set, $\mathcal{K}^*(\gamma)$ as

$$\mathcal{K}^*(\gamma) = \{x \in \mathbb{R}^n \,|\, d^\top(x - x^0) \leq 0 \; \forall d \in \mathcal{K}(\gamma)\}.$$

$\mathcal{K}^*(\gamma)$ is the set of points that are better than x^0 for all the directions in $\mathcal{K}(\gamma)$. Note that $\mathcal{K}^*(\gamma)$ is nothing but the dual cone of $\mathcal{K}(\gamma)$ moved along x^0. Both $\mathcal{K}(\gamma)$ and $\mathcal{K}^*(\gamma)$ are full dimensional pointed cones, since $0 < \gamma < \|c\|$.

Cone $\mathcal{K}^*(\gamma)$, set $\mathcal{X}(\gamma)$ and set $\mathcal{S}$ can be considered to be in the primal space, i.e., space of primal solution values. Cones $\mathcal{D}$ and $\mathcal{K}(\gamma)$ can be considered in the dual space, i.e., space of directions.

We claim the following holds when the answer is negative and continue to construct our short certificate. We prove our claim after the short certificate is constructed.

Claim 1. $\text{conv}(\mathcal{X}(\gamma)) \cap \text{int}(\mathcal{K}^*(\gamma)) \neq \emptyset$.

Let $\overline{x} \in \text{conv}(\mathcal{X}(\gamma)) \cap \text{int}(\mathcal{K}^*(\gamma))$, then a subset of $\mathcal{X}(\gamma)$ that can give $\overline{x}$ as a convex combination is a certificate. Moreover, it is a short certificate since we need $n+1$ elements from $\mathcal{X}(\gamma)$ at most. Let $\{x^1, x^2, \ldots, x^k\} \subseteq \mathcal{X}(\gamma)$ be such a subset and $\{\lambda_1, \ldots, \lambda_k\}$ be the

corresponding values such that $\overline{x} = \sum_{i=1}^{k} \lambda_i x^i$, $\sum_{i=1}^{k} \lambda_i = 1$ and $\lambda_i \geq 0$ for $i = 1, \ldots, k$ and $k \leq n + 1$. Next we show how sets $\{x^1, x^2, \ldots, x^k\}$ and $\{\lambda_1, \ldots, \lambda_k\}$ can be used to validate the NO answer.

For any given $d \in \mathcal{D}$, $\overline{x}$ being an element of $\text{int}(K^*(\gamma))$ gives us the following,

$$d^\top \left(\overline{x} - x^0\right) < 0.$$

We can write $\overline{x}$ as a convex combination of x^i values. When we replace $\overline{x}$ using this we get the following inequality,

$$d^\top \left(\sum_{i=1}^{k} \lambda_i x^i - x^0\right) < 0.$$

We can manipulate this inequality to get the following inequalities,

$$d^\top \left(\sum_{i=1}^{k} \lambda_i x^i - \sum_{i=1}^{k} \lambda_i x^0\right) < 0,$$
$$\sum_{i=1}^{k} \lambda_i d^\top \left(x^i - x^0\right) < 0.$$

Then, there exists at least one index j in $\{1, \ldots, k\}$ such that $d^\top(x^j - x^0) < 0$. Then x^j being in $\mathcal{S}$, and being a better solution for direction d means x^0 can not be optimal. Direction d is arbitrary, meaning this result holds for all d in set $\mathcal{K}(\gamma)$. Using sets $\{x^1, \ldots, x^k\}$ and $\{\lambda_1, \ldots, \lambda_k\}$ we validated the NO answer for an arbitrary d in set $\mathcal{K}(\gamma)$. This shows that sets $\{x^1, \ldots, x^k\}$ and $\{\lambda_1, \ldots, \lambda_k\}$ together is a short certificate for the inverse problem defined. □

Proof of Claim 1 Assume $\text{conv}(\mathcal{X}(\gamma)) \cap \text{int}(\mathcal{K}^*(\gamma)) = \emptyset$ for a contradiction. $\text{conv}(\mathcal{X}(\gamma))$ and $\mathcal{K}^*(\gamma)$ are both convex sets. Then there exists a hyperplane that separates these two sets. Let $\{x \in \mathbb{R}^n : a^\top x = \alpha, a \in \mathbb{R}^n, \alpha \in \mathbb{R}\}$ be such a hyperplane that separates

$\text{conv}(\mathcal{X}(\gamma))$ and $\mathcal{K}^*(\gamma)$ as

$$a^\top x \geq \alpha \; \forall x \in \text{conv}(\mathcal{X}(\gamma)),$$
$$a^\top x \leq \alpha \; \forall x \in \mathcal{K}^*(\gamma).$$

Then we can write the following inequality,

$$\min_{x \in \text{conv}(\mathcal{X}(\gamma))} a^\top x \geq \max_{x \in K^*(\gamma)} a^\top x. \tag{SEP}$$

Note that problem on the right-hand side is unbounded when a is not in $\mathcal{K}(\gamma)$. Then we can conclude that $a \in \mathcal{K}(\gamma)$ for a valid separating hyperplane. This indicates that x^0 maximizes $a^\top x$ over cone $\mathcal{K}^*(\gamma)$. Then we have the following inequality,

$$\min_{x \in \text{conv}(\mathcal{X}(\gamma))} a^\top x \geq a^\top x^0.$$

Since direction a is in $\mathcal{K}(\gamma)$ and answer to our problem is NO, there exists an $\overline{x}$ in $\mathcal{X}(\gamma)$ such that $a^\top(\overline{x} - x^0) < 0$. Point $\overline{x}$ being a feasible solution for the optimization problem over $\text{conv}(\mathcal{X}(\gamma))$ we have the following inequality,

$$a^\top \overline{x} \geq \min_{x \in \text{conv}(\mathcal{X}(\gamma))} a^\top x \geq a^\top x^0.$$

Using $a^\top(\overline{x} - x^0) < 0$, we can rewrite the inequality as

$$a^\top x^0 > a^\top \overline{x} \geq \min_{x \in \text{conv}(X)(\gamma)} a^\top x \geq a^\top x^0.$$

which is a contradiction. This indicates that the contradiction assumption, existence of a separating hyperplane, is wrong. This proves that $\text{conv}(\mathcal{X}) \cap int(\mathcal{K}^*) \neq \emptyset$. □

Figure 5.5 shows sets $\text{conv}(\mathcal{S})$, $\mathcal{K}(\gamma)$, $\mathcal{K}^*(\gamma)$ and $\text{conv}(\mathcal{X}(\gamma))$ for the example introduced

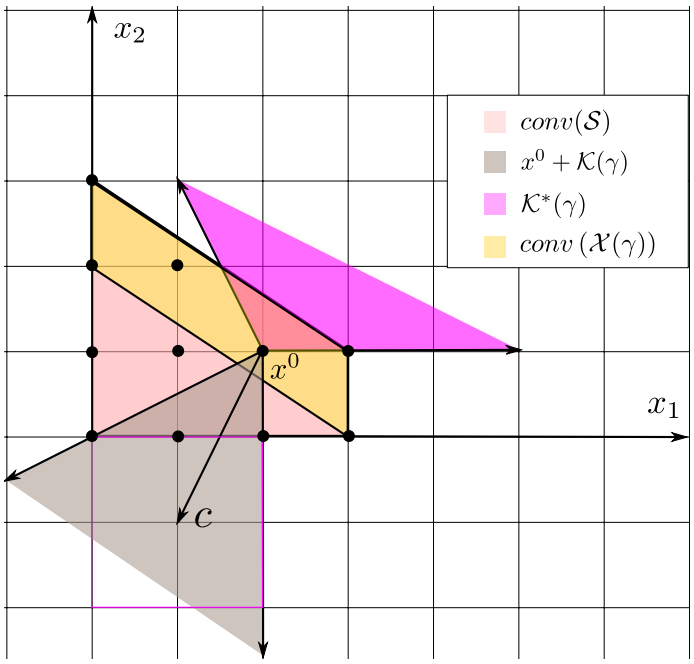


Figure 5.5: A small example demonstrates conv($\mathcal{S}$), $\mathcal{K}(\gamma)$, $\mathcal{K}^*(\gamma)$, conv($\mathcal{X}(\gamma)$)

in Section 5.2, where c and x^0 are redefined. In this case $c = (-1, -2)$, $x^0 = (2, 1)$ and $\gamma = 1$.

All this theory indicates that complexity of INVD is same as MILPL. The difference is the certificate for MILPL is just a feasible point where the certificate for INVD problem is at most $n + 1$ points with corresponding weights. Certificate for INVD problem is a little more complicated than certificate of MILPL.

Theorem 5.3.10. *INVD is* coNP*–complete.*

Proof MILPL can be reduced to INVD. Let inputs of MILPL be $(c, \gamma, \mathcal{S})$ then MILPL can be decided by deciding INVD with inputs $(c^2 \leftarrow c, \gamma^2 \leftarrow 0, \mathcal{S}^2 \leftarrow \mathcal{S}, x^0 \leftarrow \frac{\gamma c}{\|c\|^2})$. INVD asks whether some d in cone $\{d \in \mathbb{R}^n | d^\top \left(\frac{\gamma c}{\|c\|^2} - x\right) \leq 0 \ \forall x \in \mathcal{S}\}$ satisfies $\|c - d\| \leq 0$. Only d that satisfies $\|c - d\| \leq 0$ is $d = c$. For answer to be positive c must be in this

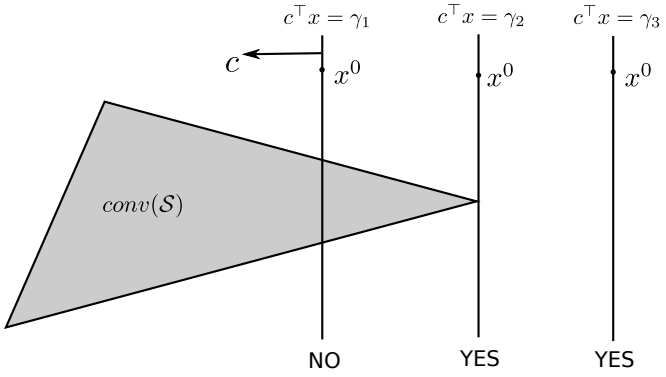


Figure 5.6: Reduction Example

cone. c is in this cone if and only if

$$\begin{aligned} c^\top \left(\frac{\gamma c}{\|c\|^2} - x \right) &\leq 0 \qquad \forall x \in \mathcal{S}, \\ \gamma - c^\top x &\leq 0 \qquad \forall x \in \mathcal{S}, \\ \gamma &\leq c^\top x \quad \forall x \in \mathcal{S}, \end{aligned}$$

which means answer to MILPL is positive. This indicates answer to INVD is positive if and only if answer to MILPL is positive. □

Figure 5.6 shows x^0 for various γ values. Answer for γ_1 is negative and for γ_2 and γ_3 is positive. Position of x^0 is just for presentation. For γ_1 case x^0 is displayed to be outside of conv($\mathcal{S}$). This is just for display and the result is independent of x^0 being in conv($\mathcal{S}$) or not. The answer is negative for both of the cases.

Lower bound problem for inverse MILP can be defined as follows.

Definition 5.3.11. ***Inverse MILP lower-bounding problem (INVL)****: Given $\gamma \in \mathbb{R}$, $c \in \mathbb{R}^n$, $x^0 \in \mathbb{R}^n$ and an MILP with feasible set $\mathcal{S}$, is $\min_{d \in \mathcal{K}(y) \cap \mathcal{D}} y \geq \gamma$?*

Theorem 5.3.12. *INVL problem is in* NP.

Proof We need to show existence of a short certificate that can validate YES answer.

When answer is YES, optimal value of the inverse problem is greater than equal to γ. We show existence of a short certificate that validates optimal value can not be less than γ, i.e. no feasible direction d that optimizes x^0 and its distance to c is less than γ. This is same as validating NO answer for INVD. The only difference is now the directions are strictly less than γ-distance to c. Remember the claim we proved,

$$\text{conv}(\mathcal{X}(\gamma)) \cap int(\mathcal{K}^*(\gamma)) \neq \emptyset.$$

Note that $\mathcal{K}^*(\gamma)$ is the set of points that are at least as good as x^0 for all directions d at most γ-distant to c. $int(\mathcal{K}^*(\gamma))$ is the set of points that are strictly better than x^0 for all directions d that are strictly less than γ-distant to c.

After this point the proof goes on same as proof of INVD being in coNP. The short certificate is the same and it can be used to show that the optimal value of inverse problem can not be less than γ.

Theorem 5.3.13. *INVL is* NP*–complete.*

Claim 2. *There exists a positive ϵ, such that if $c^\top x > \gamma$ holds for all x in $\mathcal{S}$ then $c^\top x > \gamma + \epsilon$ holds for all x in $\mathcal{S}$.*

Proof of Claim 2 Such an ϵ can be found using vertex complexity of $\mathcal{S}$. Its encoding will be a polynomial of c and vertex complexity of $\mathcal{S}$.

Claim 3. *If there exists an x in $\mathcal{S}$ such that $\gamma \geq c^\top x$ holds, then one can find a positive δ such that $\|c - d\| \geq \delta$ holds for all feasible d for the inverse problem with the following input ($c^2 \leftarrow c$, $\mathcal{S}^2 \leftarrow \mathcal{S}$, $x^0 \leftarrow \frac{(\gamma+\epsilon)c}{\|c\|^2}$).*

Proof of Claim 3 To prove the claim we will manipulate the inverse problem constraint. Inverse problem constraint is given below.

$$d^\top(x^0 - x) \leq 0 \; \forall x \in \mathcal{S}$$

Let c optimizes $\overline{x}$ over $\mathcal{S}$. Inverse problem constraint will hold for $\overline{x}$. We can write it using $\overline{x}$ as

$$\begin{aligned}
d^\top(x^0-\overline{x}) &\leq 0\\
d^\top(x^0-\overline{x}) - c^\top(x^0-\overline{x}) + c^\top(x^0-\overline{x}) &\leq 0\\
(d-c)^\top(x^0-\overline{x}) &\leq -c^\top x^0 + c^\top\overline{x}\\
(d-c)^\top(x^0-\overline{x}) &\leq -\gamma-\epsilon + c^\top\overline{x}\\
\gamma+\epsilon-c^\top\overline{x} &\leq (c-d)^\top(x^0-\overline{x})\\
\epsilon &\leq (c-d)^\top(x^0-\overline{x})\\
\epsilon &\leq \|c-d\|\|x^0-\overline{x}\|\\
\frac{\epsilon}{\|x^0-\overline{x}\|} &\leq \|c-d\|.
\end{aligned}$$

Using γ, c and vertex complexity of $\mathcal{S}$, such a positive δ can be computed.

Proof of Theorem 5.3.13 MILPD can be reduced to INVL. Let inputs of MILPD be (c, γ, $\mathcal{S}$) then MILPD can be resolved by deciding INVL with inputs ($c^2 \leftarrow c$, $\gamma^2 \leftarrow \delta$, $\mathcal{S}^2 \leftarrow \mathcal{S}$, $x^0 \leftarrow \frac{(\gamma+\epsilon)c}{\|c\|^2}$). ϵ and δ are small positive rationals computed from inputs of MILPD as explained in Claim 2 and Claim 3.

INVL asks whether $\|c-d\| \geq \delta$ holds for all d in $\{d \in \mathbb{R}^n | d^\top \left(\frac{(\gamma+\epsilon)c}{\|c\|^2} - x\right) \leq 0 \; \forall x \in \mathcal{S}\}$.

Deciding INVL with described inputs resolves MILPD. If answer to INVL is positive, then $d = c$ is not feasible for inverse problem. This indicates that c does not optimize x^0 over $\mathcal{S}$, there exists $\overline{x}$ in $\mathcal{S}$ such that,

$$c^\top\overline{x} < c^\top x^0 = \gamma+\epsilon.$$

Using Claim 2 we can deduct $c^\top\overline{x} \leq \gamma$. This means answer to MILPD is positive.

When answer to INVL is negative, optimal value of inverse problem is 0 by our design

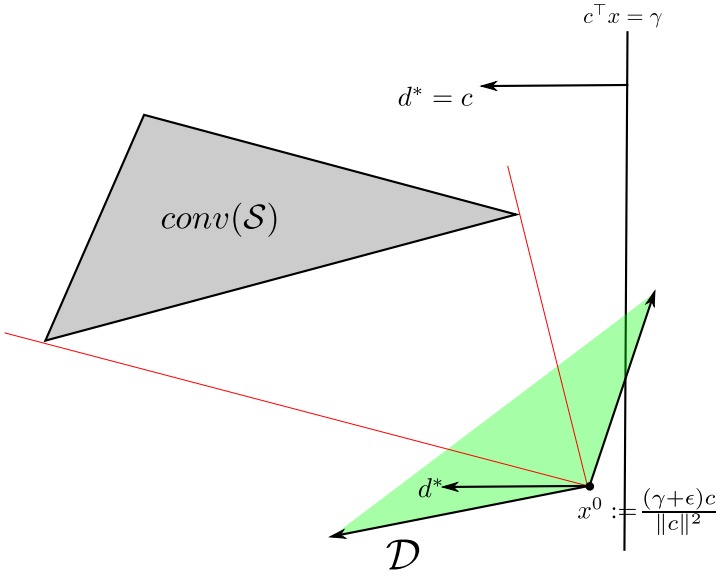


Figure 5.7: Claim 2 on a Simple Example

of δ, Claim 3. This indicates c optimizes x^0,

$$\gamma < c^\top x^0 = \gamma + \epsilon < c^\top x \quad \forall x \in \mathcal{S}.$$

This means answer to MILPD is negative. □

Note that the reduction presented in Theorem 5.3.13 can also be used in Theorem 5.3.10. The one presented in Theorem 5.3.10 is just simpler and does not require introduction of ϵ and δ.

Figure 5.7 illustrates the case described in Claim 2, inequality $c^\top x > \gamma$ holds for all x in $\mathcal{S}$. Figure displays the cone of feasible d directions and optimal d as d^*. Answer to both MILPD and INVL problems is negative.

Figure 5.8 illustrates a case where optimal value of MILP is exactly γ. It can also be considered as an illustration of the case described in Claim 3, forward problem optimal value is exactly gamma. Inverse optimal value is denoted by d^*. Positive δ as described in Claim 3 is a lower bound for the inverse problem. Answer to both MILPD and INVL

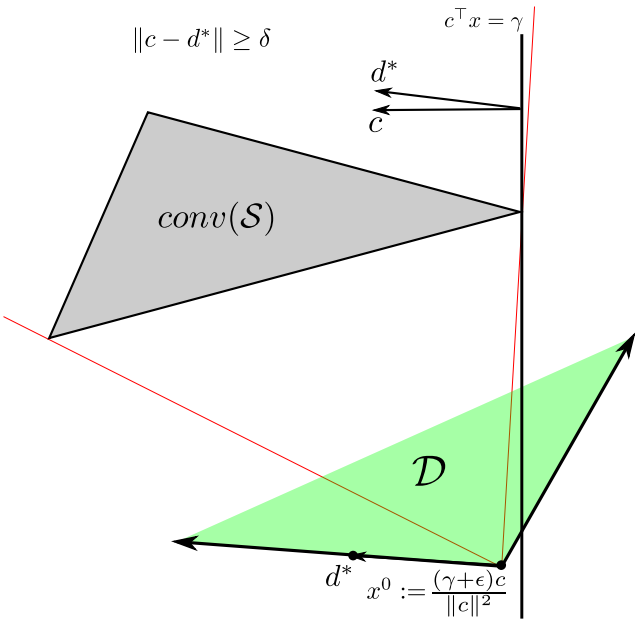


Figure 5.8: Claim 3 on a Simple Example

problems for the displayed inputs is positive. From the figure it is easy to see that result in Claim 3 holds when the forward problem optimal value is not exactly γ but strictly less.

Definition 5.3.14. ***Inverse MILP optimal value verification problem (INVO)****: Given $\gamma \in \mathbb{Q}$, $x^0 \in \mathbb{R}^n$, $c \in \mathbb{R}^n$, and an MILP with feasible region $\mathcal{S}$, is $\min_{d \in \mathcal{K}(y) \cap \mathcal{D}} y = \gamma$?*

Theorem 5.3.15. *INVO problem is in class D^P–complete.*

Proof As noted before, reduction presented in Theorem 5.3.13 can be used to reduce both MILPL and MILPD to INVD and INVL problems respectively. Using this reduction, language of INVO can be written as an intersection of languages of INVD and INVL that are in coNP and NP respectively. This proves that INVO is in class D^P. INVO problem is complete for D^P since MILPV can be reduced to INVO using the same reduction. □

Note that verifying exact optimal value of both inverse and forward problems are D^P–complete.

5.4 Conclusion and Future Directions

In this paper, we formally defined various problems related to the inverse MILP in which we try to derive an objective function d closest to a given estimate c that make a given solution x^0 optimal over the feasible region $\mathcal{S}$ to an MILP. This problem can be seen as an optimization problem over the set of all inequalities valid for $\mathcal{S}$ and satisfied at equality by x^0. Alternatively, it can also be seen as optimization over the 1-polar with some additional constraints. Both these characterization make the connection the separation problem associated with $\mathcal{S}$ evident.

After defining the problem formally, we gave a cutting-plane algorithm for solving it under the l_1 and l_∞ norms and observed that the separation problem for the feasible region is equivalent to the original forward problem, enabling us to conclude by the framework of Grötschel, Lovász, and Schrijver [GLS93] that the problem can be solved with a polynomial number of calls to an oracle for solving the forward problem.

This algorithm places the decision version of inverse MILP in the complexity class Δ_2^{P}, but it is possible to prove a stronger result. The main contribution of this study is to show that this decision problem is complete for the class coNP, which is on the same level of the polynomial-time hierarchy of that of the forward problem. We proved the problem is in coNP by giving a short certificate for the negative answer and then show it is complete for coNP by reducing the MILP lower bound (MILPL) problem to inverse MILP decision (INVD) problem. We also provide a reduction for the inverse lower bound problem. Finally, we show that the inverse optimal value verification problem is complete to the class D^{P}, which is precisely the same class containing the MILP optimal value verification problem.

Theorem 5.3.6 states that an optimization problem (over a convex set) can be solved in polynomial time given an oracle for the separation problem. Technically, this does not allow us to place the optimization and separation problems on precisely the same level of

the polynomial hierarchy. It is likely that the GLS result can be modified slightly in order to show that optimization and separation are indeed on the same level of the hierarchy. There are also some interesting open questions remaining to be explored with respect to complexity.

Finally, we have implemented the algorithm and a computationally oriented study is left as future work. Such a study would reveal the practical performance of the separation-optimization procedure and investigate the possible relationship between the number of iterations (oracle calls) and the polyhedral complexity (vertex/facet complexity), among other things. This may provide practical estimates for the number of iterations required to solve certain classes of problems.

Chapter 6

Computational Experiments

This chapter presents computational experiments designed, and their results, to answer to the questions raised throughout the dissertation. These experiments explore the practical performance of the algorithms and their implementations introduced in Chapter 2, 3 and 4.

Software libraries introduced in Chapter 4 are used to conduct the experiments. This chapter can also be considered as a demonstration of the capability of the software introduced.

Problem sets used in the experiments are introduced first. Parameters and experimental setting of the software libraries is explained afterwards. Performance profiles, given by Dolan and Moré [DM02], are used to compare the performance of different algorithms, solvers and parameter settings. For MISOCP, two different performance measures are used, CPU time and number of nodes processed to find the optimal solution.

6.1 Problem Set

Experiments are conducted using three different problem sets, conic benchmark library 2014 (CBLIB) problems given by Friberg [Fri16], random problems given by Góez [Góe13] and Euclidean Steiner tree problems given by Beasley [Bea].

There are already 6 Euclidean Steiner tree problems modeled as MISOCP instances in CBLIB. These problem instances in CBLIB are provided by Drewes [Dre09] and based on the model given by Fampa and Maculan [FM04] using data provided by Beasley [Bea]. The model provided by Fampa and Maculan [FM04] involves a parameter called M which denotes the maximum distance between any two terminals. Steiner tree problem instances provided in CBLIB compute M within the optimization model. The problem instances in CBLIB can be improved by pre-computing M and embedding the computed value into the problem instances as a number. Pre-computing M reduces the size of the instances. For example, estein5_A.mps in CBLIB has 132 variables, 211 linear constraints, and 258 nonzero values in the coefficient matrix of linear constraints. When M is pre-computed, the problem reduces to 96 variables, 61 linear constraints and 150 nonzero values in the coefficient matrix. Pre-computing M does not change the number of conic constraints in the problem. As a side note, in preliminary experiments, we observed that this improves the performance of the DisCO solver, as well as CPLEX and Mosek solvers on solving these instances.

The Euclidean Steiner tree problem instances in CBLIB are with 4 and 5 terminals (estein4 and estein5 instances) and each terminal has 2 coordinates. We created problem instances with 6 and 7 terminals using the model provided by Fampa and Maculan [FM04] and data provided by Beasley [Bea]. Parameter M is pre-computed and embedded into the instances. The smallest problem instance provided by Beasley [Bea] contains 10 terminals. There are 15 different problems with 10 terminals. We used the first 6/7 terminals from the first three problems to obtain our instances. The problem instances created are denoted as `estein6_0`, `estein6_1`, `estein6_2`, `estein7_0`, `estein7_1` and `estein7_2`. This is the same procedure used in generating instances in CBLIB (estein4_A, estein4_B, estein5_A etc.). The difference is we used the improved formulation as described.

In addition to CBLIB problems, randomly generated problems are also used in the

experiments. Feasible regions of the random problems are high dimensional ellipsoids when integrality constraints are relaxed. Current implementation of disjunctive conic cut procedure described in Section 3.2 generates conic disjunctive cuts for simple variable disjunctions of ellipsoids. These problem instances are generated and used by Góez [Góe13].

Another reason is obtaining instances that are hard in terms of combinatorial difficulty. Most of the current CBLIB instances are relatively low in terms of number of integer variables and branch bound trees are relatively smaller. We would like to obtain problems that are harder to solve combinatorially and lead larger branch-and-bound trees. These type of instances are required to have a balanced set of problems for experiments that reveal the gains and losses of the algorithms proposed.

6.2 Algorithms and Parameter Settings

This section presents the details of the tested algorithm–parameter combinations with various solvers. The questions raised throughout the dissertation are answered through testing computational performance of these algorithm, parameter and solver combinations. DisCO solver framework, together with its dependencies, is used to implement the tested algorithms.

Algorithms tested can be categorized into two classes based on the type of subproblems. The first category is SOCP based relaxations and the second category is LP based relaxations. Table 6.1 and 6.2 present the algorithms based on SOCP and LP relaxations respectively.

Algorithms presented in Table 6.1 are variations and different implementations of SOCP relaxation based branch-and-bound algorithm. In these implementations, branch-and-bound algorithm solves SOCP at each node. We test different flavors of this algorithm, these are disco-cplex, disco-cplex-strong, disco-ipopt, disco-mosek, disco-cplex-dc-all, disco-cplex-dc-best and disco-cplex-mpi. disco-cplex, disco-cola, disco-ipopt and

Table 6.1: Algorithms based on SOCP relaxations and Solvers That Implement Them

Algorithm	Parameters	Solver	referred as
bb-socp		DisCO with CPLEX	disco-cplex
bb-socp	strong branching	DisCO with CPLEX	disco-cplex-strong
bb-socp		DisCO with COLA	disco-cola
bb-socp		DisCO with Ipopt	disco-ipopt
bb-socp		DisCO with Mosek	disco-mosek
bb-socp with disjunctive cuts	add all cuts	DisCO with CPLEX	disco-cplex-dc-all
bb-socp with disjunctive cuts	add only best	DisCO with CPLEX	disco-cplex-dc-best
parallel bb-socp		DisCO with CPLEX and OpenMPI	disco-cplex-mpi

Table 6.2: Algorithms based on LP relaxations and Solvers That Implement Them

Algorithm	Parameters	Solver	referred as
Algorithm 5, bb-lp		DisCO with OA method, CLP as solver	disco-oa
Algorithm 5, bb-lp	strong branching	DisCO with OA method, CLP as solver	disco-oa-strong
Algorithm 5, bb-lp	$\alpha \leftarrow 2$	DisCO with OA method, CLP as solver	disco-oa-2
Algorithm 5, bb-lp	$\alpha \leftarrow 4$	DisCO with OA method, CLP as solver	disco-oa-3
Algorithm 5, bb-lp	$\beta \leftarrow 0.01$	DisCO with OA method, CLP as solver	disco-oa-4
Algorithm 5, bb-lp	$\beta \leftarrow 0.0001$	DisCO with OA method, CLP as solver	disco-oa-5
Algorithm 5, bb-lp	$\gamma \leftarrow 20$	DisCO with OA method, CLP as solver	disco-oa-6
Algorithm 5, bb-lp	$\gamma \leftarrow 100$	DisCO with OA method, CLP as solver	disco-oa-7
Algorithm 5 with disjunctive cuts	add all cuts	DisCO with OA method, CLP as solver	disco-oa-dc-all
Algorithm 5 with disjunctive cuts	add only best	DisCO with OA method, CLP as solver	disco-oa-dc-best
Algorithm 5 without MILP cuts		DisCO with OA method, CLP as solver	disco-oa-nomilpcuts
parallel Algorithm 5		DisCO with OA CLP and OpenMPI	disco-oa-mpi

disco-mosek are implementations where CPLEX, COLA, Ipopt and Mosek are used to solve continuous relaxations in each node. disco-cplex-strong refers to CPLEX implementation where strong branching is used as branching strategy (default for DisCO is pseudocost). disco-cplex-dc refer to disjunctive conic cut implementations. disco-cplex-dc experiments use Belotti et al. [Bel+13] disjunctive conic cut procedure to improve the SOCP relaxation at the root node. Two strategies are tested, disco-cplex-dc-best and disco-cplex-dc-all. In disco-cplex-dc-best, conic cuts are generated for all possible disjunctions and added to SOCP relaxation one by one. The cut that improves the relaxation bound most is selected and added to the SOCP relaxation to improve it. In disco-cplex-dc-all, all possible disjunctive cuts are generated and added to SOCP relaxation. In both disco-cplex-dc-best and disco-cplex-dc-all, generated cut(s) are kept in the subsequent nodes of the branch-and-bound tree. disco-cplex-mpi is the implementation where ALPS is compiled with an MPI library and the branch-and-bound search is conducted in parallel. We use OpenMPI implementation of the MPI standard [Gab+04].

Table 6.2 presents LP based branch-and-bound algorithms with different parameter variations. In these experiments, branch-and-bound algorithm solves LPs at each node. disco-oa-1, disco-oa-2, disco-oa-3 and disco-oa-strong are tests of bb-lp algorithm with different parameters. disco-oa-dc-best and disco-oa-dc-all test effectiveness of disjunctive cuts for bb-lp algorithm. The strategies tested are same as the cases in Table 6.1 as explained before. disco-oa-nomilpcuts refers to experiments where MILP cuts are disabled. When bb-lp algorithm is used, cuts from MILP literature can be used to improve the LP relaxation during branch-and-bound algorithm. MILP cuts are enabled by default. Effectiveness of MILP cuts is measured by comparing the default behavior to disco-oa-nomilpcuts results. disco-oa-mpi refers to the experiments where parallel search capability of ALPS is used, similar to disco-cplex-mpi case explained above. In all of the tests introduced in Table 6.2 CLP is used to solve the LP subproblems.

DisCO solver is used to implement all the variations given in Table 6.1 and 6.2. Experiments with different parameters are conducted by passing DisCO binary the right values for corresponding parameters. Different algorithm and solver combinations are specified at building DisCO. Parallel versions are obtained by linking ALPS to OpenMPI library when building DisCO.

In addition to the algorithms introduced in Table 6.1 and Table 6.2, performance of Algorithm 3, i.e., cutting-plane algorithm to solve SOCP, is tested. In these experiments, COLA solver implementation of Algorithm 3 is used and this experiment is referred as cola.

6.3 Hardware

Experiments are conducted on polyps cluster of COR@L Lab. Polyps cluster contains 15 nodes. Each node has 16 AMD processors clocked at 2 GHz and 32 Gb of memory. In disco-cplex and disco-mosek, CPLEX and Mosek solvers are used with 1 thread only. Memory allowed for a single run is limited to 2 Gb. Run is terminated when DisCO hits the 2 Gb memory limit. 7100 seconds of CPU time limit is imposed on each run.

For parallel runs memory is increased with the increasing number of processors. Memory per processors is kept constant at 2 Gb, i.e., a parallel run with 15 processors is limited to 30 Gb of memory. Parallel runs are limited to 7100 seconds of wall clock time.

6.4 COLA Experiments

This section presents COLA solver's performance on test problem set. Abbreviations of the performance measures used are as follows.

NC Number of conic constraints in the problem.

LC Size of largest conic constraint in the problem.

US Number of unboundedness supports generated. When conic constraints relaxed problem might become unbounded. Unboundedness supports are OA cuts generated to restrict unboundedness directions that are infeasible for the conic constraints.

MUS Maximum number of unboundedness support generated for a cone.

SS Number separation supports, i.e., OA cuts generated to cut conic infeasible points.

MSS Maximum number of separation supports generated for a cone.

NUMLP Number of linear optimization problems solved.

CPU Total CPU time spent on solving the problem in seconds.

Table 6.3 presents the performance of Cola on random problems. Table 6.4 and 6.5 presents the results on CBLIB problems.

All CBLIB problems are still bounded when conic constraints are relaxed. Same is not true for random problems. US and MUS columns are dropped in CBLIB results. Total number of OA cuts generated for a problem is US plus SS. It is SS for CBLIB problems since US is 0.

There are two problem characteristics that are crucial for the performance statistics, number of conic constraints (NC) and size of conic constraints. We measure size of conic constraints with the largest cone (LC) of the problem. For the problems tested here, conic constraints in a problem are of same size.

At each iteration of the cutting-plane algorithm, one cut is added for each conic constraints that is violated. Thus, number of cuts added (US+SS) in a single iteration is expected to increase with the number of conic constraints (NC) in the problem. Since only one cut is added for each cone at each iteration and approximation of larger cones

requires more cuts, one can expect that number of iterations (NUMLP) will increase with the cone size (LC).

In the following two paragraphs two different problems are examined, chainsing-1000-1 and classical_200_1. chainsing-1000-1 has smaller cones but many of them. classical_200_1 has a single large cone.

From Table 6.4, problem chainsing-1000-1 has 2994 number of conic constraints. Total number of cuts generated for this problem is 14479. Problem is solved in 11 iterations (1 initial LP solve call, 10 solve calls after cuts). Iteration number is low but number of cuts added in each iteration is high. For a specific conic constraint at most 10 cuts are generated.

From Table 6.4, problem classical_200_1 has 1 conic constraint. Total number of cuts generated for this problem is 1055. Problem is solved in 1055 iterations (1 initial LP solve call, 1055 solve calls after cuts). Iteration number is high but number of cuts added in each iteration is only 1.

COLA uses simplex algorithm to solve linear optimization problems. It exploits warm starting benefits of simplex algorithm. When many cuts/constraints are added in a single iteration simplex warm starting benefits deteriorate. It takes more simplex iterations to solve. On the other hand when problem has a single cone, only one cut is added to the problem. Simplex algorithm is expected to solve problem faster. There are 2 other complications arise in this case, (1) cuts are denser since cone is large, and (2) cuts might become almost parallel. (2) means rows of the linear problem are almost linearly dependent. This might cause numerical difficulties and other measures in simplex algorithm level should be taken to prevent these.

Table 6.3: COLA statistics on Góez's random instances

instance	NC	LC	US	MUS	SS	MSS	NUMLP	CPU
r12c15k5i10	5	3	0	0	15	4	5	0.01
r14c18k3i9	3	6	4	2	28	13	16	0.01
r17c30k3i12	3	10	12	4	192	66	74	0.07
r17c20k5i15	5	4	0	0	8	3	4	0.0
r22c30k10i20	10	3	5	1	26	5	8	0.02
r22c40k10i20	10	4	16	2	83	13	22	0.03
r23c45k3i21	3	15	13	5	397	140	148	0.25
r27c50k5i25	5	10	16	4	315	69	77	0.11
r32c45k15i30	15	3	8	1	36	4	6	0.0
r32c60k15i30	15	4	29	3	169	15	32	0.02
r52c75k5i35	5	15	7	2	293	71	74	0.15

6.5 DisCO Experiments

This section presents experiments with algorithms introduced in Chapter 2 and 3. Descriptions and details of these experimental settings are given in Table 6.1 and 6.2. Details of the tested algorithms are explained in the corresponding chapters. This section has further explanations on the implementational details of the algorithms where necessary.

Experiments include both serial and parallel runs with various number of processors. Parallel experiments investigate the scalibility of DisCO. DisCO can be used to solve MILPs. Testing of DisCO on MILPs is left out of scope of this dissertation.

Two different branching strategies are tested, namely pseudocost and strong branching. Details and explanations of these branching strategies on the algorithms tested are discussed in Section 3.3.3. Both bb-socp and bb-lp algorithm categories are tested with these two strategies, to determine the best branching strategy for each category. Note that strong branching is costly and performance depends highly on the warm starting capability of the relaxation solver. We expect this cost to be higher for the tests in bb-socp category where IPM is used to solve the SOCP subproblems.

Table 6.4: COLA statistics on CBLIB 2014 Part 1

instance	NC	LC	SS	MSS	NUMLP	CPU
chainsing-1000-1	2994	3	14479	10	11	13.01
classical_200_1	1	201	1055	1055	1056	114.11
classical_50_1	1	51	328	328	329	1.89
estein4_A	9	3	36	6	7	0.01
estein4_B	9	3	44	6	9	0.02
estein4_C	9	3	60	10	11	0.02
estein4_nr22	9	3	41	6	7	0.0
estein5_A	18	3	109	11	14	0.02
estein5_nr21	18	3	99	9	11	0.03
pp-n1000-d10000	1000	3	16107	18	19	7.19
pp-n100-d10000	100	3	1613	18	19	0.12
pp-n10-d10000	10	3	161	17	18	0.02
robust_50_1	2	52	260	134	135	0.78
robust_100_1	2	102	577	297	298	7.64
robust_200_1	2	202	960	499	500	64.86
shortfall_100_1	2	101	533	502	503	11.44
shortfall_100_2	2	101	674	630	631	19.28
shortfall_100_3	2	101	573	527	528	12.55
shortfall_200_1	2	201	719	690	691	53.67
shortfall_200_2	2	201	876	841	842	77.74
shortfall_50_1	2	51	307	284	285	1.73
shortfall_50_2	2	51	344	320	321	2.13
shortfall_50_3	2	51	451	408	409	3.58

Table 6.5: COLA statistics on CBLIB 2014 Part 2

instance	NC	LC	SS	MSS	NUMLP	CPU
sssd-strong-25-8	24	3	243	13	15	0.07
sssd-strong-30-8	24	3	260	13	14	0.07
sssd-weak-20-8	24	3	171	8	9	0.03
sssd-weak-25-8	24	3	171	8	9	0.04
sssd-weak-30-8	24	3	165	8	9	0.03
turbine07_aniso	25	3	53	9	11	0.01
turbine07GF	25	3	10	4	5	0.0
turbine07_lowb_aniso	25	3	64	10	12	0.03
turbine07_lowb	27	9	81	8	9	0.02
turbine07	26	9	67	12	14	0.02
turbine54GF	119	3	25	10	11	0.05
turbine54	120	9	220	11	13	0.05
uflquad-nopsc-10-150	1500	3	14281	16	20	14.68
uflquad-nopsc-20-150	3000	3	29063	17	30	74.84
uflquad-nopsc-30-100	3000	3	29108	23	39	66.91
uflquad-nopsc-30-150	4500	3	42809	19	39	156.26
uflquad-nopsc-30-200	6000	3	55650	19	40	332.71
uflquad-nopsc-30-300	9000	3	83624	16	41	819.0
uflquad-psc-10-150	1500	3	10837	13	23	14.08
uflquad-psc-20-150	3000	3	18164	15	37	70.09
uflquad-psc-30-100	3000	3	16595	22	49	69.07
uflquad-psc-30-150	4500	3	23675	19	50	128.58
uflquad-psc-30-200	6000	3	33972	19	50	291.65
uflquad-psc-30-300	9000	3	54083	19	50	978.33

DisCO implements maximum infeasibility, pseudocost and strong branching strategies. The default branching strategy of DisCO is pseudocost branching. If the branching algorithm is not specified for a test then default branching strategy of DisCO is used. DisCO branching strategy parameter is set to strong for strong branching tests. In pseudocost branching, cost of all variables are set to 0 initially. Among variables with same cost, variable with the smallest index is picked.

DisCO depends on COIN-OR's ALPS for tree search. ALPS implements best-first, best-estimate-first, breadth-first, depth-first and hybrid search strategies. ALPS's default search strategy is hybrid search.

Hybrid search strategy carries depth-first search until the objective value of the subproblems get worse than the current upper bound, i.e., node is fathomed. In this case, ALPS picks the sibling of the fathomed node as the next. Hybrid strategy stops diving if all the siblings are fathomed or their quality is worse than the best available by a certain threshold. Default search strategy of ALPS with its default parameter values are used in all the experiments conducted.

In the preliminary experiments we observed that different search strategies might perform slightly better (together with specific branching strategies) for some specific problem families with bb-lp algorithm. But when all the benchmark problems are considered, hybrid strategy performs the best and variance in the solution time is less.

Linear cuts generated can be categorized into two classes, OA cuts and MILP cuts. OA cuts are used to improve the LP approximations. These cuts are given in Theorem 2.2.1 and 2.2.2. For a given point these cuts can be computed using a closed form formula as explained in Section 3.3.2. OA cuts are used in bb-lp algorithm only. They are controlled by parameters of α, β and γ of bb-lp algorithm (Algorithm 5). Default values for these parameters are 1, 0.001 and 50. We test other sets of values for these parameters. These experiments are referred as disco-oa-2 to disco-oa-7, and OA cut parameter values used

are given in Table 6.2.

MILP cuts are used to cut integer infeasible points from LP relaxations. Note that MILP cuts are available only in case of bb-lp algorithm. Experiments where MILP cuts are disabled are referred as disco-oa-nomilpcuts. By default OA cuts and MILP cuts are used together to cut conic infeasible and integer infeasible points respectively. Six type of cuts from MILP literature are used, clique, flow cover, Gomory, knapsack, MIR and odd hole cuts. Generation of MILP cuts are controlled by κ and δ parameters of the bb-lp algorithm. COIN-OR's CGL is used to generate cuts. DisCO asks the relevant CGL generator to generate cuts. CGL may or may not return cuts. DisCO adds a cut to the relaxation if it cuts the current solution more than tail-off parameter specified (default $1e-8$).

As described in Chapter 3, bb-lp algorithm cleans the cuts that are not potentially helping at the current relaxation. The helping cuts are determined by checking their slack. The ones that have slacks above a threshold are removed from relaxation. Cut cleaning is carried after warm start procedure that approximates the cones in the root node, and at each iteration of the bounding loop. In bounding loop, cuts with slacks above a threshold are not removed immediately. They are removed if their slack is above threshold for 3 consecutive iterations.

6.5.1 bb-socp with Various Solvers

bb-socp algorithm is tested with various solvers. Different solvers are used to solve relaxed problems in each node. Tested solvers are COLA (disco-cola), CPLEX (disco-cplex), Ipopt (disco-ipopt) and Mosek (disco-mosek). All SOCP solvers are limited to single thread.

In Ipopt, in the root node value 1 is used as starting point value for all variables, except leading variables of conic constraints. Leading variables are set to $\sqrt{n}$ and $\sqrt{\frac{n}{2}}$ for Lorentz and rotated Lorentz cones, where n is the size of the cone. In lower level nodes,

solution of the parent node is used as starting point.

To obtain results in this subsection, DisCO is built to work with the corresponding solvers. Solvers disco-cola, disco-cplex, disco-mosek and disco-ipopt implement bb-socp algorithm. All solvers share the same DisCO parameters. Changes in the performance measures (number of nodes and CPU time) occur due to different solvers used. Number of nodes differs due to different solutions (result from alternative optimal solutions) reported by solvers.

Tables A.1, A.2 and A.3 present CPU time spent in seconds and number of nodes processed of bb-socp algorithm for the solvers discussed. TL, ML and SF denote time limit, memory limit and solver failure respectively. Solver failure occurs when the solver being used fails to solve a relaxation subproblem in a node.

Like any computer program solvers might fail. CPLEX fails (reports that problem is abandoned) to solve subproblems (in a node) for some of the problems. It fails on all estein4 problems, estein6_1, on some random problems and most of sssd problem family. Mosek solver also fails on some instances. It fails some instance of sssd, turbine and uflquad-psc families. Note that CPLEX and Mosek solvers do not fail when used to solve the same problems directly, rather than in DisCO framework. This might be due to their internal recover of the relaxation solver failure or other numerical operations to prevent failures. There are no such measures in DisCO yet. Whenever a solver fails for any of the subproblems, DisCO fail.

Note that Ipopt is a general solver and is unaware of the second-order conic structure of the underlying problems. Moreover the formulation used in Ipopt is not smooth. Smooth formulations are possible but result dense Hessian matrices. Ipopt fails frequently and this is expected.

Cola solver, hence CLP, does not fail in any of the instances, but hits time limit more often. disco-cola solves as many instances as disco-mosek but it is slower than disco-mosek

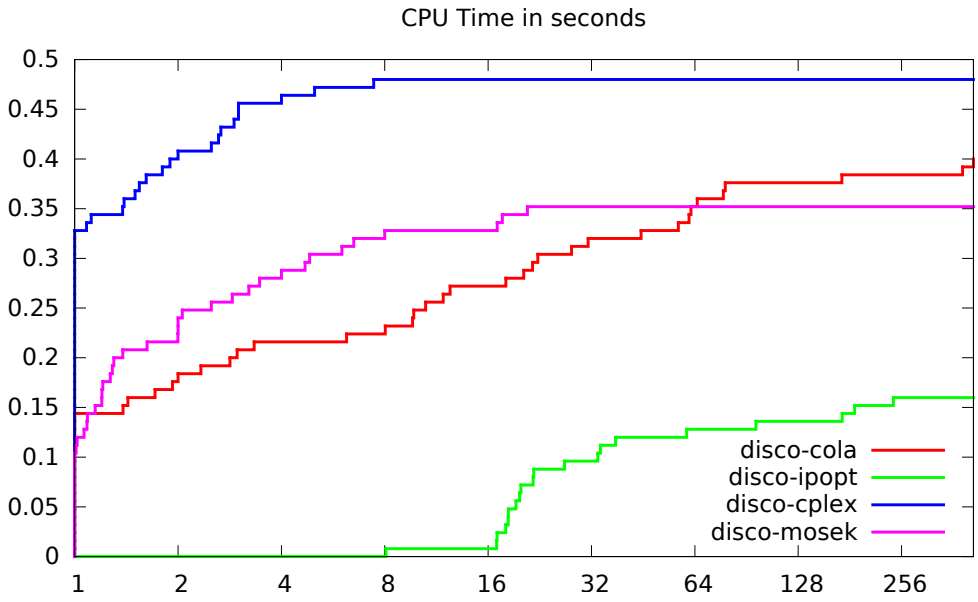


Figure 6.1: bb-socp Algorithm, Performance Profile of CPU Time with Various Solvers

in almost all instances.

Performance of bb-socp algorithm highly depends on the performance and robustness of the underlying solver. Figure 6.1 gives performance profile of CPU time of DisCO with discussed solvers. In terms of CPU time disco-cplex performs the best. It solves around 50% of the problems within the time and memory limits. Next best performing are disco-mosek and disco-cola. disco-cola is slower than disco-mosek but solves more problems within the limits. disco-cola solves around 40% of the test problems and disco-mosek solves close to 40%. disco-ipopt is the slowest and solves the least portion of the problems.

Figure 6.2 gives performance profile of number of nodes processed by different solvers. disco-cplex processes the least number of nodes. disco-cola is the second best in terms of number of nodes processed. disco-mosek comes the next, than disco-ipopt. As discussed before DisCO is run exactly with the same parameters for each solver. Difference in the

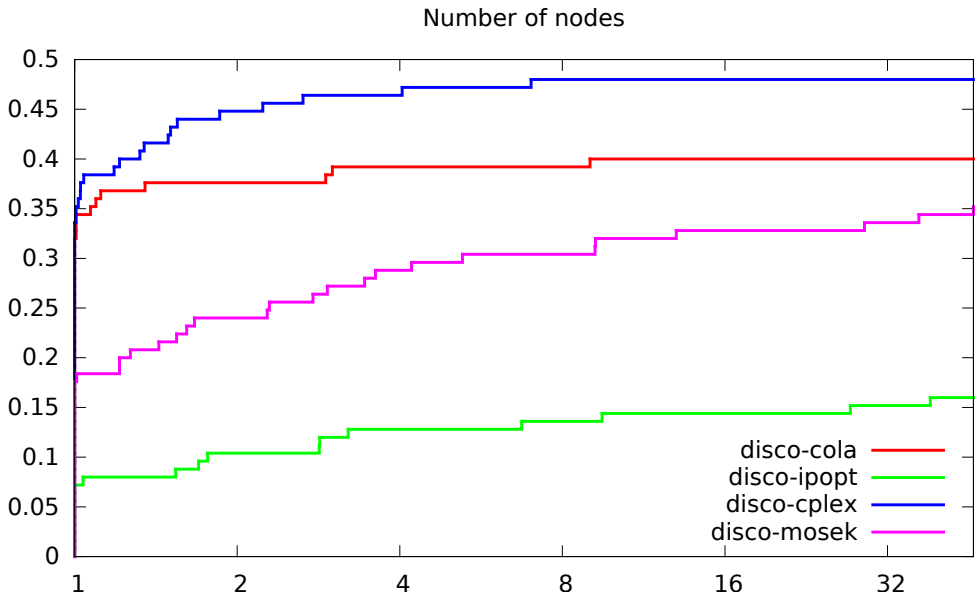


Figure 6.2: bb-socp Algorithm, Performance Profile of Number of Nodes Processed with Various Solvers

number of nodes are due to solutions reported by the solvers at each node.

Note that even though disco-cola solves problems in less number of nodes compared to disco-mosek, it is slower than disco-mosek for most of the problems. This means COLA spends more time at solving less number of nodes than Mosek. This is due to low performance of COLA solving subproblems. Results obtained here confirm the conclusions reached in Section 6.4. Similarly disco-cola is close to disco-cplex in terms of number of nodes processed but the gap is large for CPU time.

6.5.2 Branching Strategy for bb-socp

In this section we experiment with different branching strategies to determine the best performing for bb-socp algorithm. Two different branching strategies are tested, pseudo-cost and strong branching. CPLEX solver is used to solve SOCP problems at each node. Strong branching experiments are denoted as disco-cplex-strong. Pseudocost branching

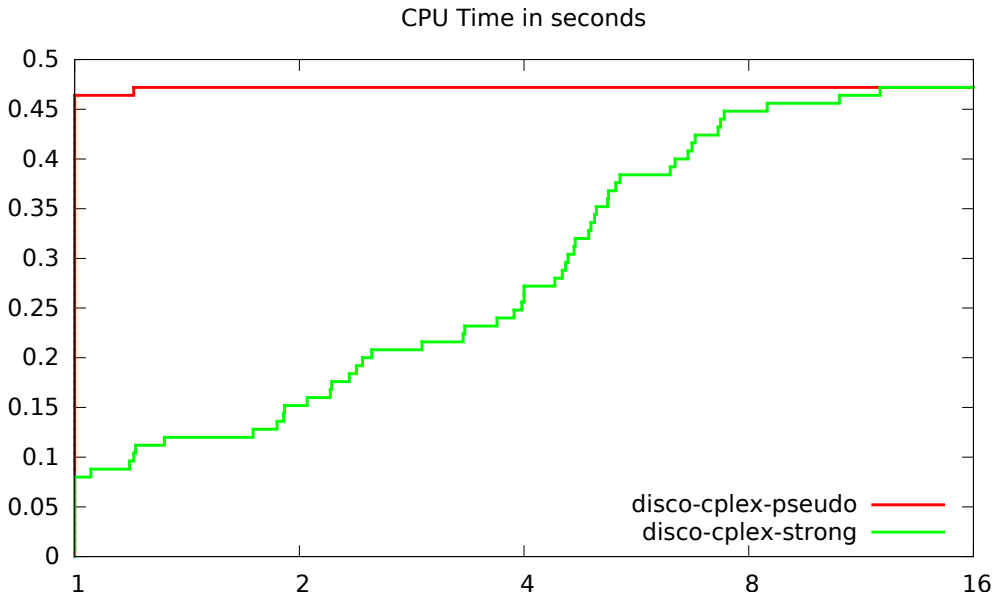


Figure 6.3: bb-socp Algorithm, Performance Profile of CPU Time with Different Branching Strategies

strategy experiments are denoted as disco-cplex, same as disco-cplex given in Section 6.5.1, since pseudocost branching strategy is the default one.

Table A.4, A.5 and A.6 give CPU time and number of nodes processed for pseudocost and strong branching strategies using disco-cplex. For problems solved by both branching strategies, pseudocost branching is consistently faster than strong branching. This is expected since strong branching is costly.

Figure 6.3 gives performance profile for the CPU time. Figure 6.4 gives performance profile of number of nodes processed by two different branching strategies. Strong branching processes less number of nodes for all problems as expected. In strong branching, branching process itself (determining the branching variable) is expensive and results in higher CPU time.

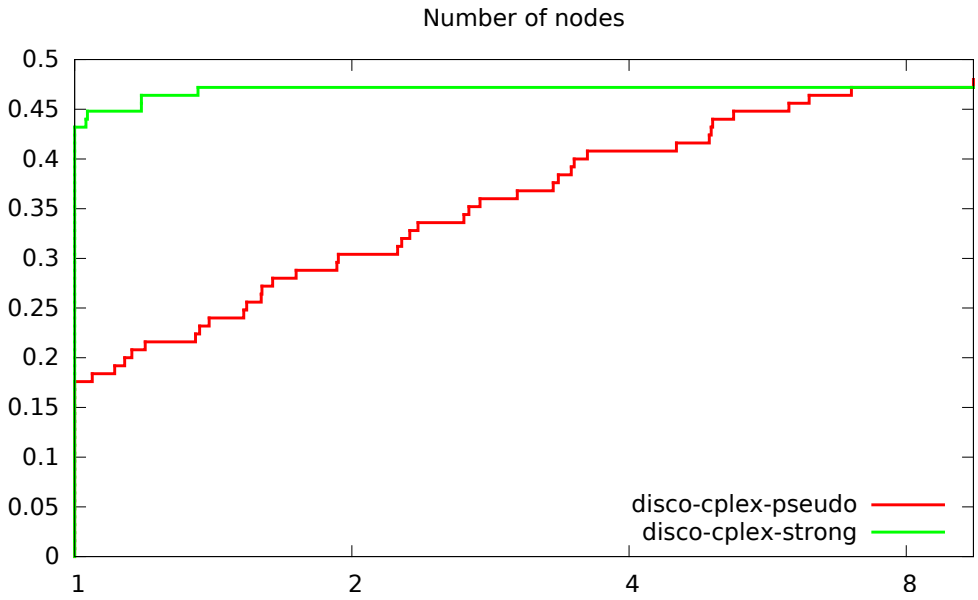


Figure 6.4: bb-socp Algorithm, Performance Profile of Number of Nodes Processed with Different Branching Strategies

6.5.3 Choosing OA Cut Parameters for bb-lp Algorithm

This section includes experiments to determine a good set of values for the parameters of bb-lp Algorithm. Note that bb-lp Algorithm has α, β and γ parameters that control the decision of cutting vs branching.

Default values for α, β and γ parameters are 1, 0.001 and 50. Different values are tried for these parameters. Different experiments and corresponding updates to the parameters are given in Table 6.2. In these experiments only one parameter value is changed at a time. disco-oa gives the results with the default parameters.

disco-oa-2 and disco-oa-3 increase α parameter and perform more cut iterations iteration (in bounding loop) before branching in case the current relaxation is both integer and conic infeasible.

In case of both integer and conic feasible subproblem solutions, β and γ parameters

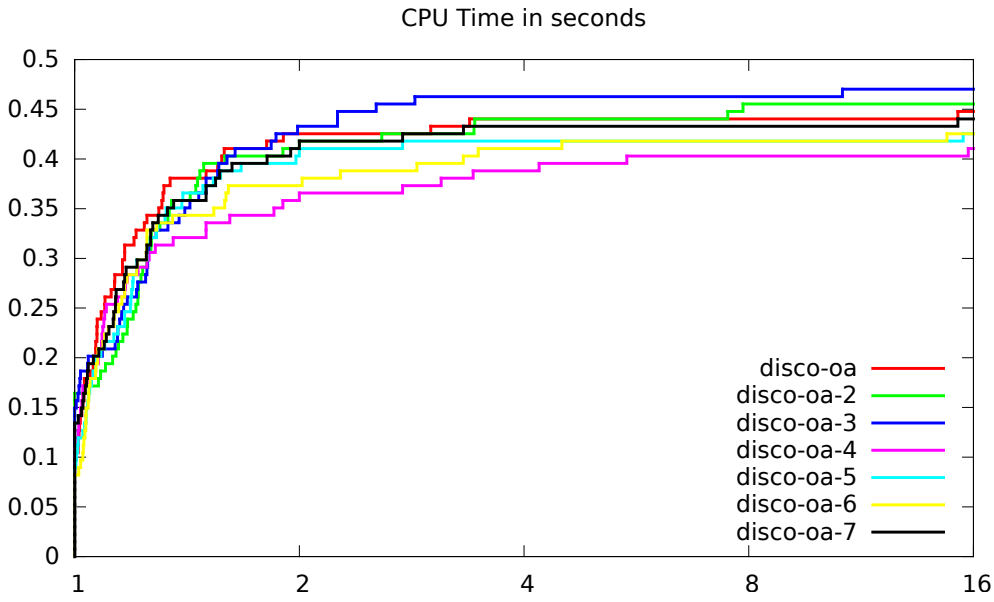


Figure 6.5: bb-lp Algorithm, Performance Profile of CPU Time with Different OA Cut Parameter Values

decide to cut if the current optimality gap seems achievable. Decreasing *beta* relaxes achievable criteria of the algorithm and leads more cuts. γ is the upper limit on cut rounds in this case. Increasing γ means adding more cuts. disco-oa-4 increases parameter β and leads less cuts. disco-oa-5 decreases parameter β and favors more cuts. disco-oa-6 reduces parameter γ, means less cut iterations. disco-oa-6 increases it and favors more cut iterations.

Figure 6.5 gives performance profile of the CPU time spent for the discussed settings. Figure 6.6 gives the performance profile of the number of nodes processed. Table A.7, A.8, A.9, A.10, A.11 and A.12 presents the results of the OA parameter experiments.

We observe that the default strategy is fastest for majority of the problems but disco-oa-3 solves the most number of instances. disco-oa-3 performs best in terms of number of nodes. This makes sense since it favors cutting more compared to the default strategy. disco-oa-4 and disco-oa-6 are the worst performers in terms of both CPU time and number

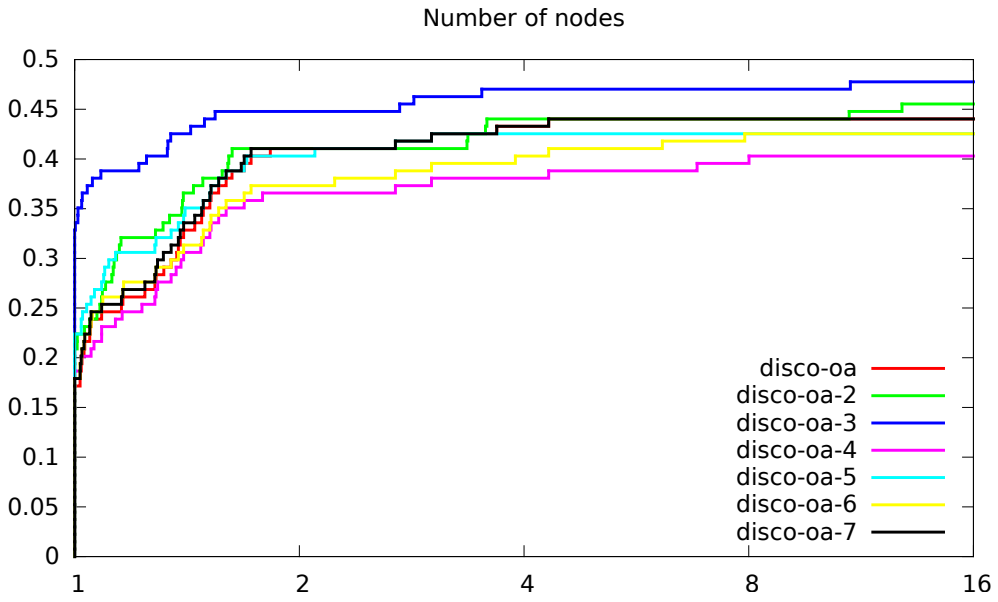


Figure 6.6: bb-lp Algorithm, Performance Profile of Number of Nodes Processed with Different OA Cut Parameter Values

of nodes. These are the strategies that favor the OA cuts least.

Default values of the parameters can be considered as a compromise between cutting and branching. Performance profiles show that run with the default parameter values is the best performer among all parameter settings tested for the majority of the problems.

6.5.4 Branching Strategy for bb-lp Algorithm

Experiments of this section aims to determine the best performing branching strategy for bb-lp algorithm. Two different branching strategies are tested for this purpose, pseudocost and strong branching. CLP solver is used to solve LP problems in each node. Default parameters are used for OA cut management. MILP cuts are enabled by default.

Performance profile for the CPU time spent is given in Figure 6.7. Figure 6.8 gives the performance profile for the number of nodes processed by the two branching strategies. Table A.13, A.14 and A.15 gives the values used for generating the performance profiles.

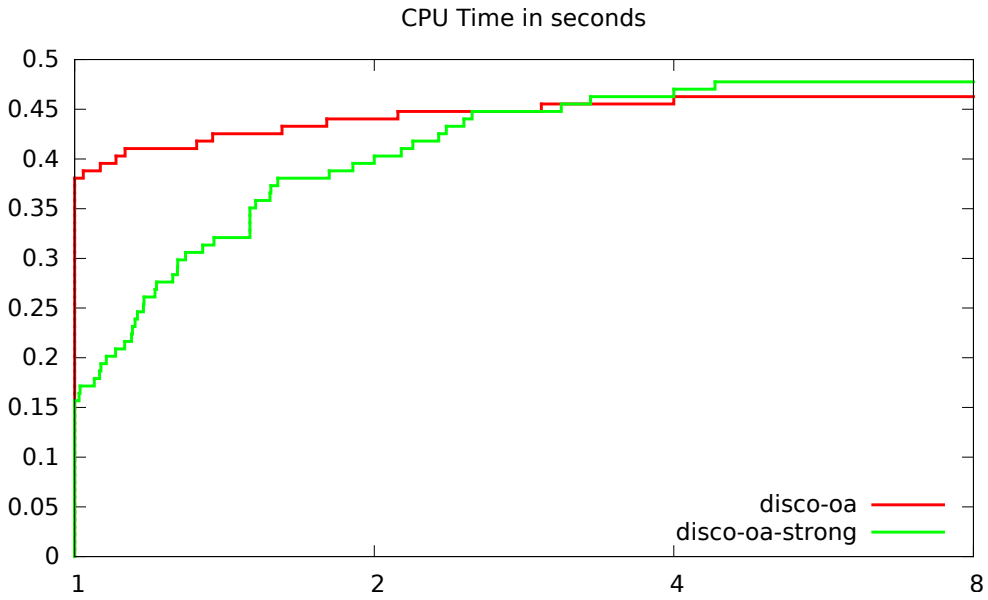


Figure 6.7: bb-lp Algorithm, Performance Profile of CPU Time with Different Branching Strategies

There is a clear gap between the two branching strategies in terms of both CPU time and number of nodes. Strong branching processes less number of nodes as expected. Pseudocost branching performs better in terms of CPU time. Strong branching solves a few more problems than pseudocost branching in the time permitted.

6.5.5 MILP Cuts for bb-lp

MILP cuts are enabled by default in DisCO. All bb-lp experiments conducted in this thesis have MILP cuts enabled. It is explicitly stated when they are disabled. MILP cuts are generated as explained in Section 3.3.8. This section compares default MILP strategy of DisCO to disabling MILP cuts.

Table A.16, A.17 and A.18 presents the results of the experiment. Figure 6.9 displays the performance profile of the two strategies for the CPU time. Figure 6.10 gives the performance profile of the number of nodes processed.

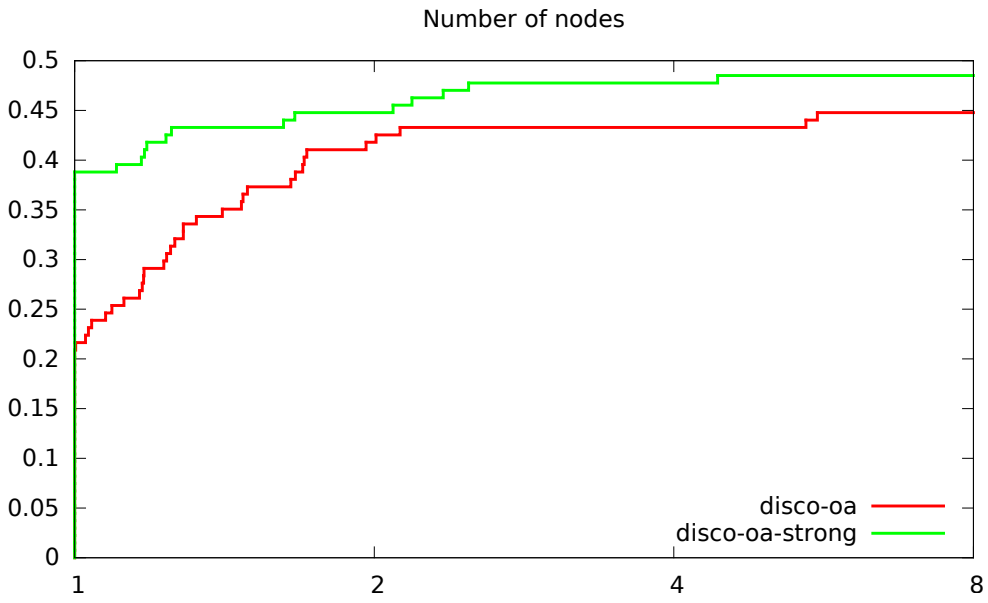


Figure 6.8: bb-lp Algorithm, Performance Profile of Number of Nodes Processed with Different Branching Strategies

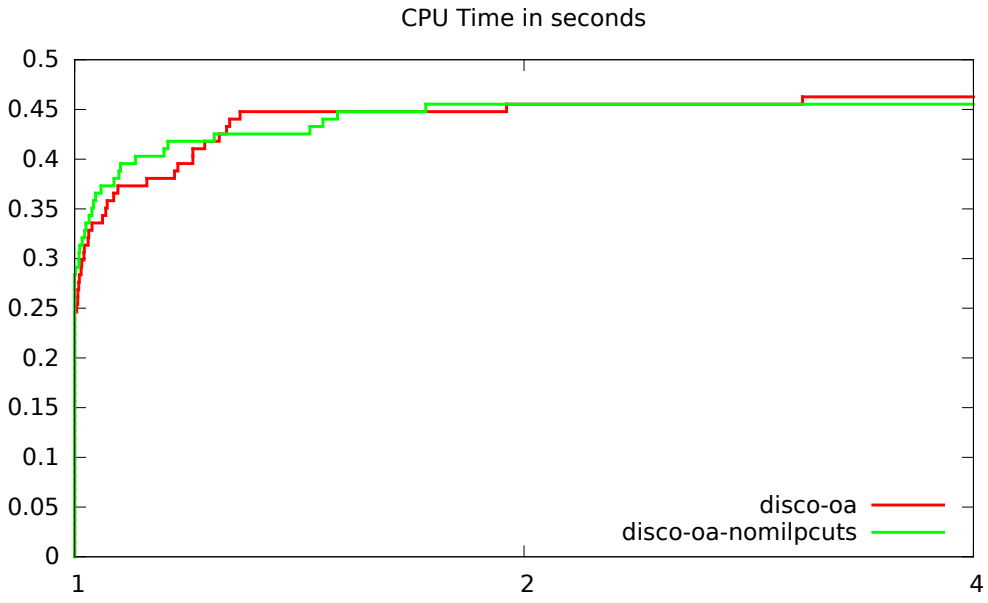


Figure 6.9: bb-lp Algorithm, Performance Profile of CPU Time with and without MILP Cuts

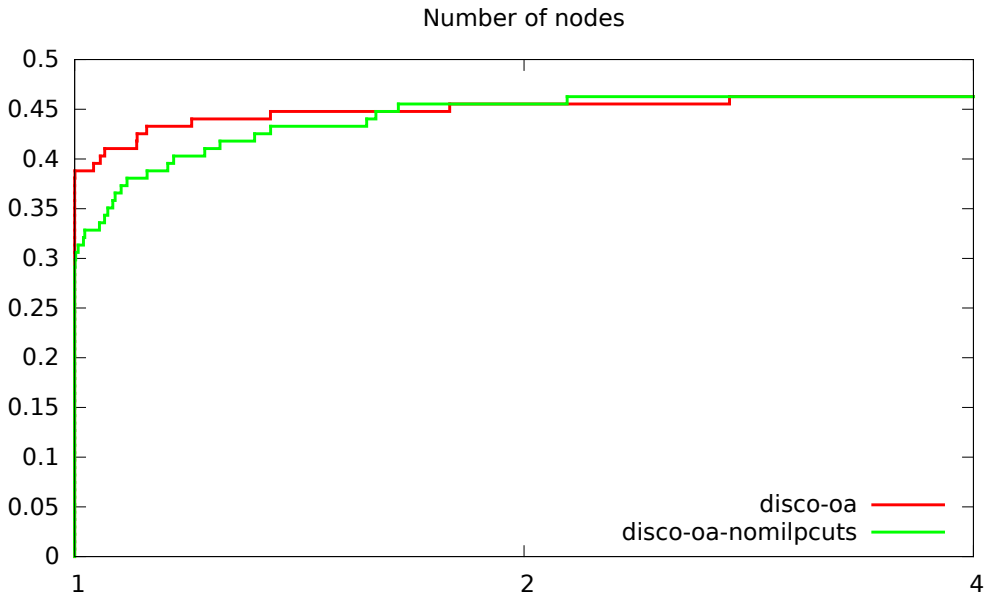


Figure 6.10: bb-lp Algorithm, Performance Profile of Number of Nodes Processed with and without MILP Cuts

Adding MILP cuts does not improve the performance of the bb-lp algorithm in terms of CPU time much. A small improvement is observed in terms of number of nodes processed. bb-lp Algorithm processes less number of nodes when MILP cuts are enabled. Performance improvement on number of nodes is not reflected to CPU time. The results we obtained in this section are similar to the results reported in Abhishek, Leyffer, and Linderoth [ALL10] using FilMINT.

6.5.6 bb-socp with Disjunctive Cuts

This section presents experiments that test the effectiveness of disjunctive cuts for MISOCP using bb-socp algorithm. In this experiment, we compare 3 different bb-socp runs, disco-cplex and disco-cplex-dc-all and disco-cplex-dc-best.

We use disjunctive cut procedure of Belotti et al. [Bel+13] described in Section 3.2.4. Conic cut library is used to generate cuts. In these experiments we generate cuts in the

root node to improve the continuous relaxation. Then we use DisCO with CPLEX to solve the problem.

We test two different strategies, (1) generate all cuts and add the best one (disco-cplex-dc-best), (2) generate and add all possible cuts (disco-cplex-dc-all). In strategy (1), we use the bound improvement measure to decide the best. We generate all cuts, add them to the problem (one at a time) and solve the problem to measure the bound improvement. After detecting the most bound improving, we just add that one to the problem and start branch and bound process.

Randomly generated problem instances are used for disjunctive cut experiments presented in this section and in Section 6.5.7. Input sets to disjunctive cut procedure are ellipsoids for these problem instances. Conic cuts are computed for disjunctions over ellipsoids.

Note that cuts generated are valid for the whole branch-and-bound tree. They are kept in the subproblems in all the lower level nodes of the tree.

CPLEX barrier method suffers from numerics and fails to converge on some instances. We mark these instances as unsolved. disco-cplex fails on some of the random problems after adding cuts due to this. Check Table A.19 for details. Performance profiles are generated for the problems solved successfully.

6.5.7 bb-lp with Disjunctive Cuts

This section presents experiments that test the effectiveness of disjunctive cuts for MISOCP using bb-lp algorithm. In this experiment, we compare 3 different bb-lp runs, disco-oa-nomilpcuts and disco-oa-dc-all and disco-oa-dc-best.

We use disjunctive cut procedure of Belotti et al. [Bel+13] described in Section 3.2.4. Conic cut library is used to generate cuts. In these experiments cuts are generated in the root node to improve the continuous relaxation. Afterwards, DisCO with outer-

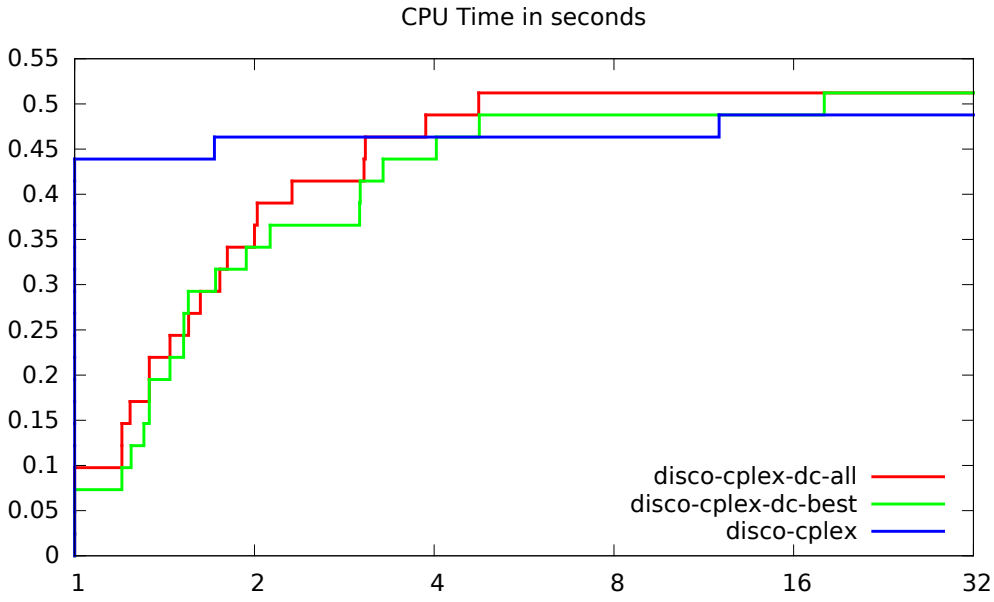


Figure 6.11: Performance Profile of CPU Time using disco-cplex with disjunctive cuts

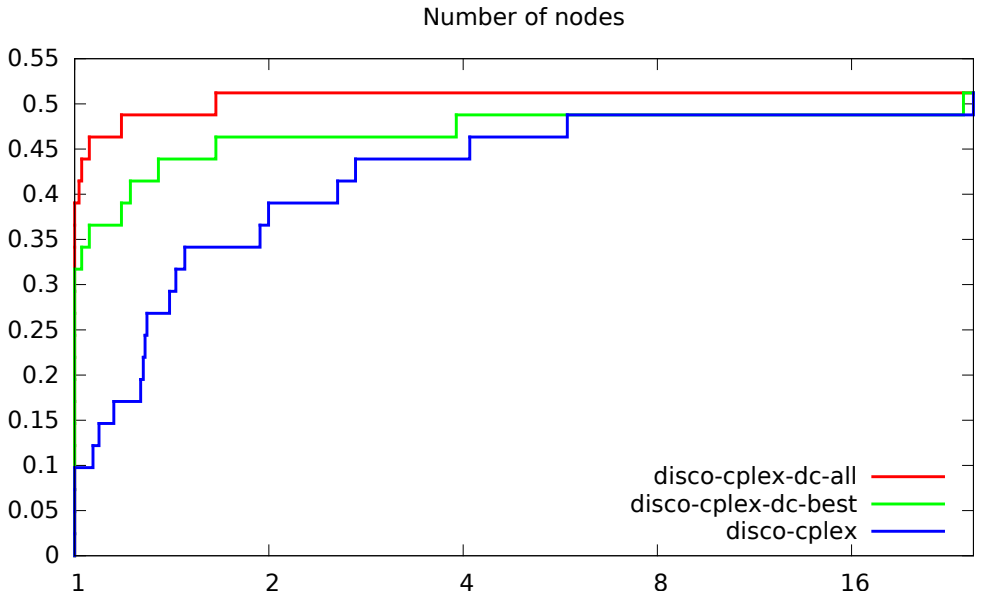


Figure 6.12: Performance Profile of Number of Nodes Processed using disco-cplex with disjunctive cuts

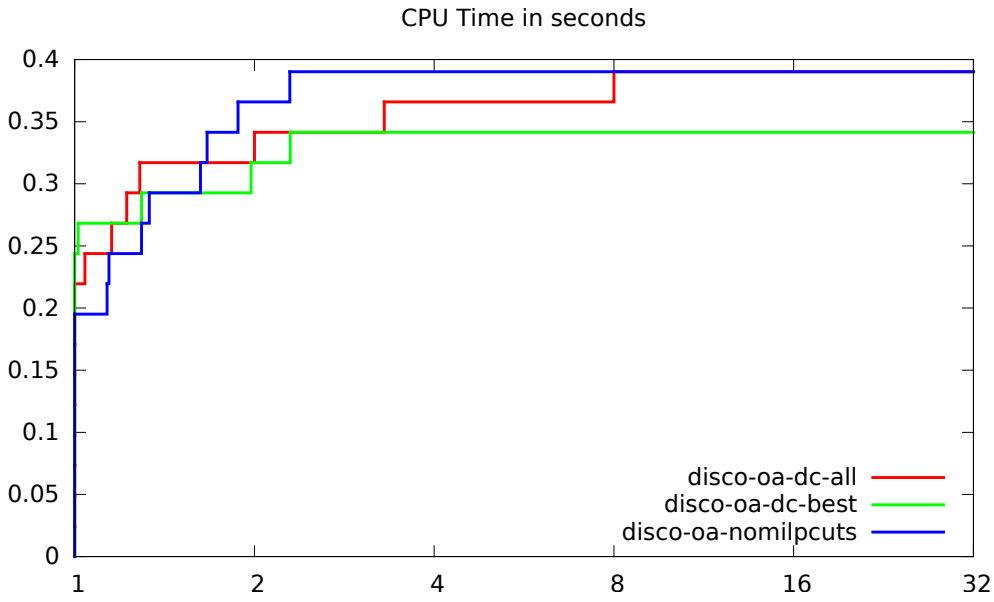


Figure 6.13: Performance Profile of CPU Time using bb-lp with disjunctive cuts

approximation algorithm is used to solve the problem. The experimental setting and the problem instances are same as in Section 6.5.6, except that bb-lp algorithm is used, instead of bb-socp.

Two different cut generation strategies are tested, same as in Section 6.5.6, (1) generate all cuts and add the best one (disco-oa-dc-best), (2) generate and add all possible cuts (disco-oa-dc-all). In strategy (1), bound improvement measure is used to decide the best. Cuts are generated for all possible disjunctions and added to the problem one at a time and bound improvement of the the modified problems is measured. Only the most bound improving cut is kept and branch-and-bound process is started.

Note that cuts generated are valid for the whole branch-and-bound tree. They are kept in the subproblems in all the nodes of the branch-and-bound tree.

CLP solver fails to find optimal solutions in some of the subproblems encountered during bb-lp algorithm. These instances are marked as unsolved and excluded from the

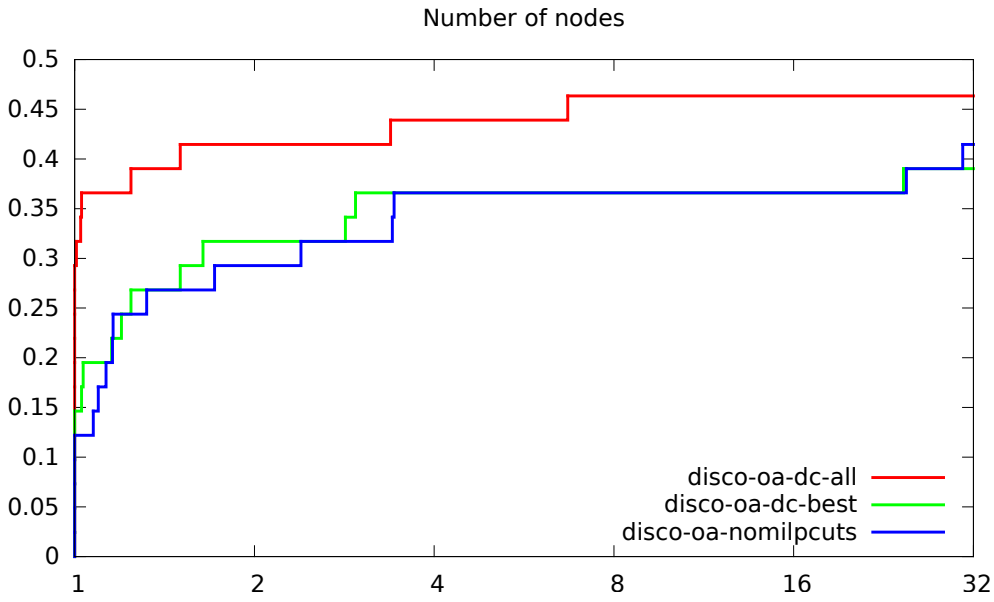


Figure 6.14: Performance Profile of Number of Nodes Processed using bb-lp with disjunctive cuts

performance profiles. Check Table A.20 for details.

6.5.8 Parallelization and Scalibility of bb-socp

In this section we present results of experiments with parallel bb-socp. For these experiments we use ALPS built with OpenMPI. For these experiments we use polyps cluster for COR@L Lab. Each node in polyps cluster has 16 processors. In our experiments we use up to 4 nodes and 15 processor at each node, meaning up to 60 processors. Each node has 32 GB of memory. We limit memory of parallel experiments to 2GB per processor, i.e. for experiments with 15 processors memory limit is 30 GB in total. For parallel experiments in this section CPLEX is used as solver.

Figure 6.15 gives performance profile of CPU time for various number of processors. Figure 6.16 presents performance profile of number of nodes processed. Table A.21, A.22 and A.23 given in appendix presents the results performance profiles are based on.

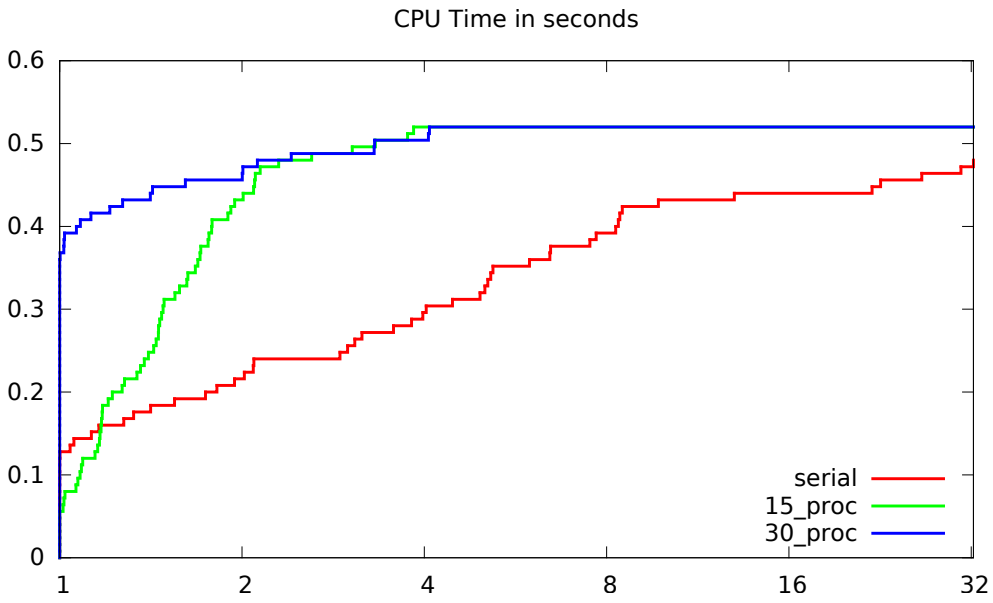


Figure 6.15: Performance Profile of CPU Time for disco-cplex-mpi for various number of processors

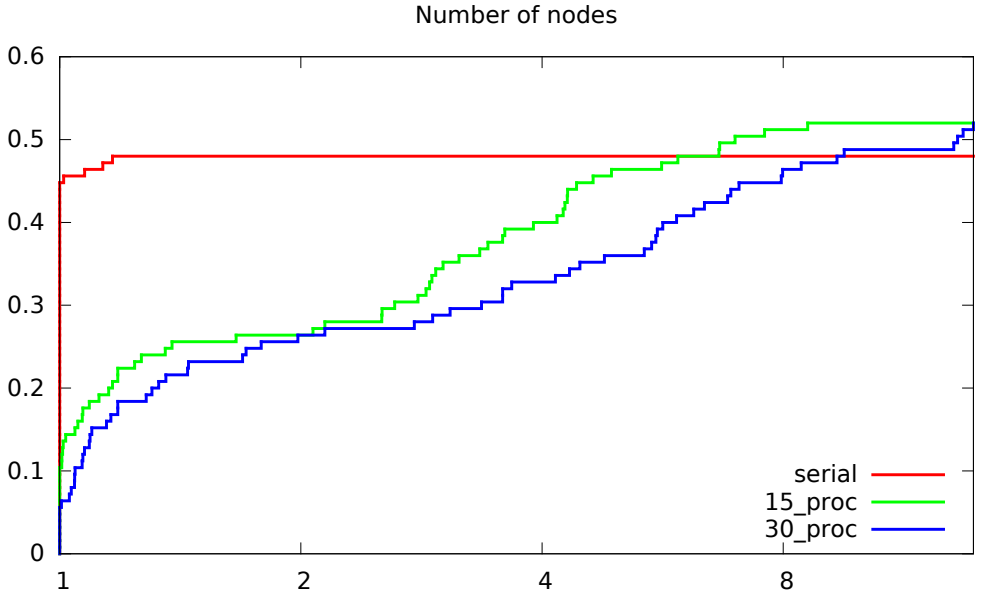


Figure 6.16: Performance Profile of Number of Nodes Processed for disco-cplex-mpi for various number of processors

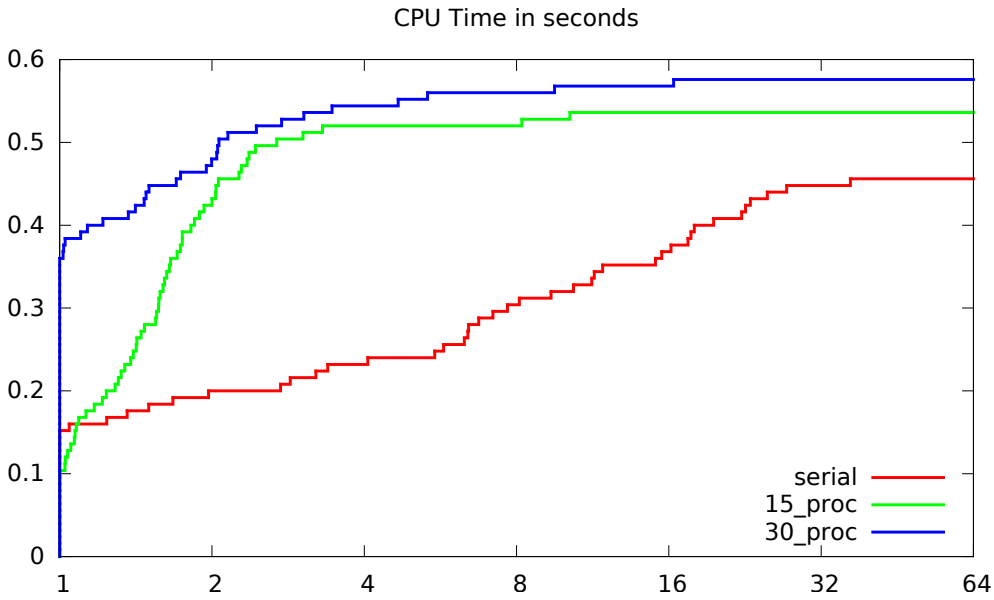


Figure 6.17: Performance Profile of CPU Time for disco-oa-mpi for various number of processors

6.5.9 Parallelization and Scalibility of bb-lp

In this section we test scalibility of parallel bb-lp algorithm. For this, we use DisCO build with OA option and ALPS build with OpenMPI. Memory is limited to 2 GB per process. Experimental setting is same as the one described in Section 6.5.8, except that bb-lp algorithm is used.

Figure 6.17 and 6.18 presents performance profile of parallel experiments with disco-oa-parallel run.

Table A.24, A.25 and A.26 given in appendix presents the results performance profiles are based on.

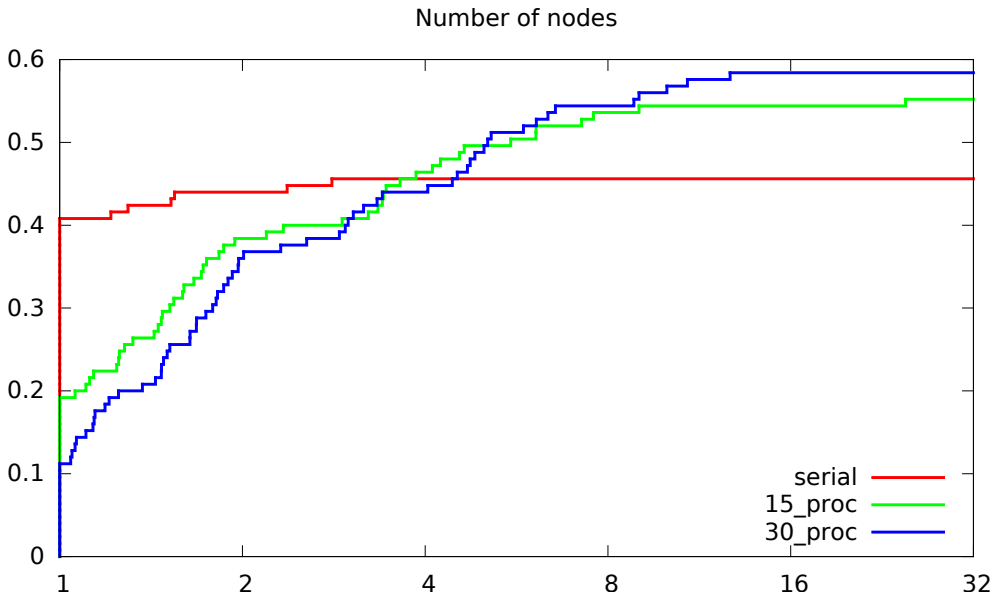


Figure 6.18: Performance Profile of Number of Nodes Processed for disco-oa-mpi for various number of processors

6.5.10 bb-lp versus bb-socp

Performance of bb-lp is compared to bb-socp in terms of CPU time and number of nodes processed. This section answers the question of whether speedup from using simplex in bb-lp can beat the tighter relaxations of bb-socp algorithm. Answering this question is one of the motivations of this study.

Figure 6.19 and 6.20 give performance profiles of CPU time and number of nodes processed for bb-lp and bb-socp algorithms respectively. The results show that bb-lp performs worse than bb-socp in terms of CPU time. bb-socp processes less number of nodes as expected since the subproblems are tighter than bb-lp.

bb-lp performs poorly compared to bb-socp for problems with large cones, i.e. size 50 or larger. Around half of the problems in CBLIB has cones of size 3. Figure 6.21 give performance profile of CPU time generated from the randomly generated instances and

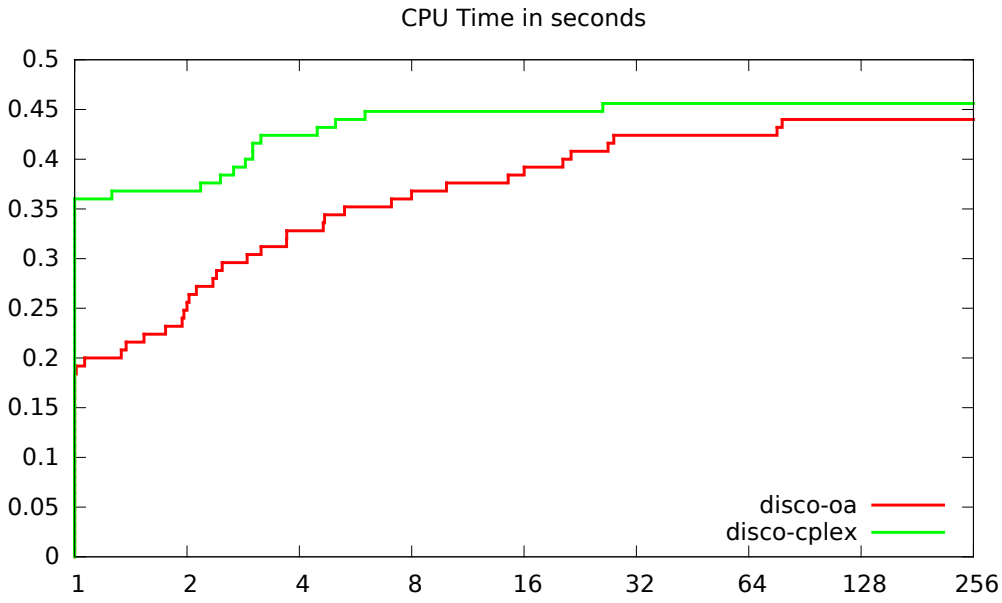


Figure 6.19: Performance Profile of CPU Time, bb-lp versus bb-socp

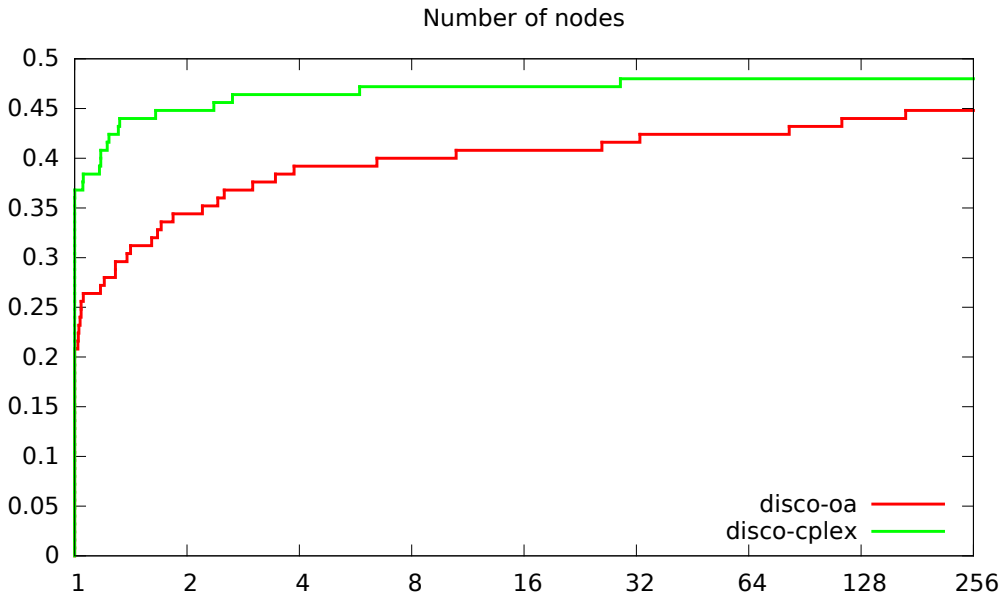


Figure 6.20: Performance Profile of Number of Nodes Processed, bb-lp versus bb-socp

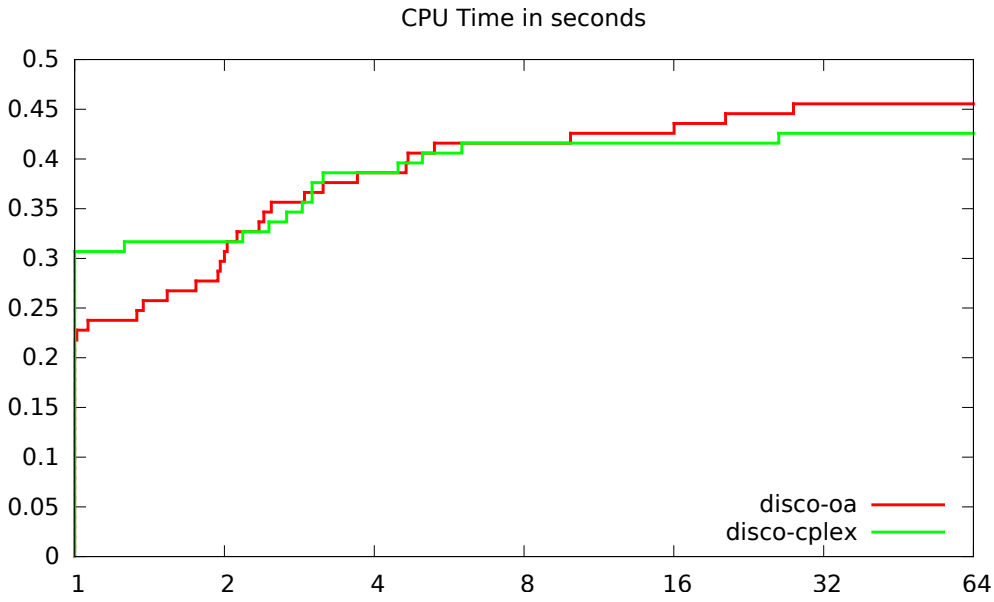


Figure 6.21: Performance Profile of CPU Time, bb-lp versus bb-socp, Problems with Low Dimensional Cones

instances of cone size 3 of CBLIB.

We observe that performance of bb-lp is close to bb-socp for the problems with smaller cones. CPLEX solver fails to find optimal solution of relaxations in some of the Steiner problems and due to this bb-lp ends up solving more problems. As a general remark we can also claim that combinatorially challenging problems are more suitable for the bb-lp algorithm. Stochastic service system design (SSSD) problem instances are combinatorially challenging. We observe that bb-lp with strong branching can solve instances from this family. We can conclude that bb-socp is winner for problems with large cones and combinatorially easier (less number of nodes are needed to prove optimality). bb-lp performance is comparable to bb-socp for problems with small cones. It has advantage for problems that are harder combinatorially, i.e. many nodes should be processed to prove optimality.

Figure 6.22 gives performance profile of both serial and parallel runs of disco-oa and disco-cplex for all the problem instances in the test set.

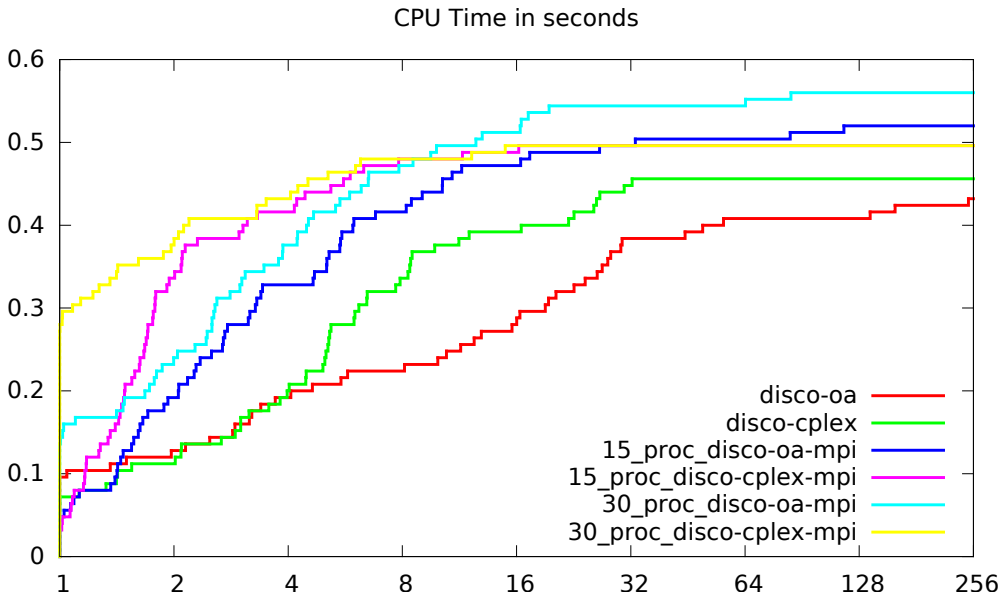


Figure 6.22: Performance Profile of CPU Time, bb-lp (disco-oa) versus bb-socp (disco-cplex)

Figure 6.23 gives performance profile of both serial and parallel runs for the randomly generated instances and instances of cone size 3 of CBLIB. Figure 6.23 shows that disco-oa-mpi with 30 processors solves the highest number of problems with low dimensional cones.

6.6 Conclusion

Pseudocost branching strategy performs better than strong branching for majority of the problems for both bb-lp and bb-socp algorithms. This result is somehow expected for the bb-socp case due to lack of warm start in the IPM implementation used. The difference in CPU time is large for bb-socp. Pseudocost branching strategy performs better than strong branching for bb-lp but the difference is not as dramatic as bb-socp case. Strong branching implementation in case of bb-lp use warm start capability of the CLP solver.

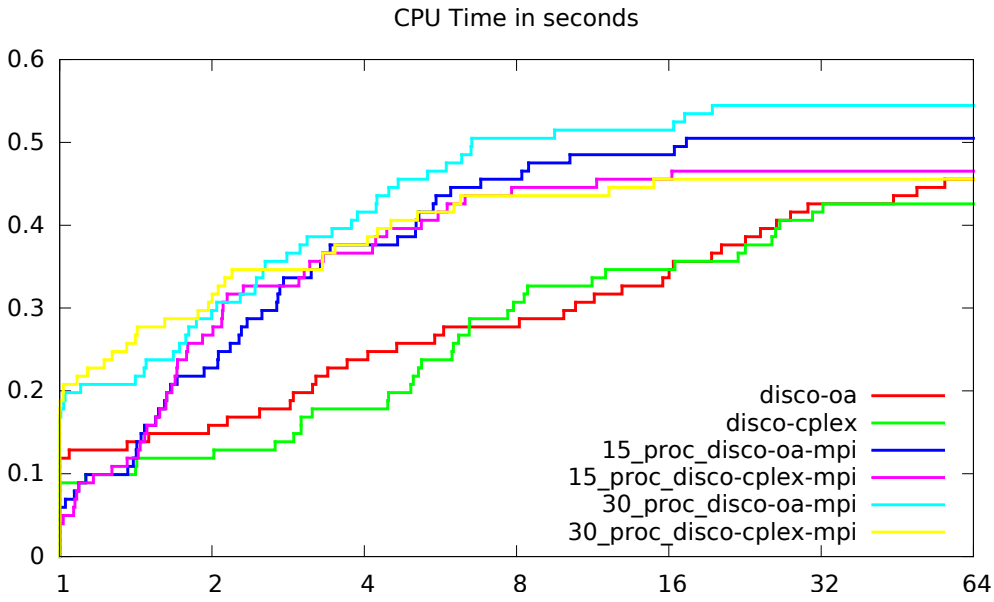


Figure 6.23: Performance Profile of CPU Time, bb-lp (disco-oa) versus bb-socp (disco-cplex), Problems with Low Dimensional Cones

Even though strong branching performs worse than pseudocost for majority, it is the only setting that solves instances from stochastic service system design (SSSD) problem family. This problem family is hard combinatorially. Branch-and-bound trees are large and many nodes are processed to prove optimality. Pseudocost branching of bb-lp can solve some instances from this family only in parallel case when 30 processors are used.

Experiments that validate the default OA cut parameter values are conducted. We showed that a balance between cut generation and branching should be achieved when setting OA cut generation parameters. Aggressive cutting lead less number of nodes and aggressive branching lead more nodes. Both, in both of the extremes CPU time required increases.

MILP cuts does not have much effect on the performance of the bb-lp solver. The performance of the bb-lp algorithm does not change much when they are deactivated. This conclusion is parallel to outcome reported in Abhishek, Leyffer, and Linderoth [ALL10]

using FilMINT. MILP cuts are slightly more effective for the randomly generated problem instances. We believe this is due to fact that these instances have integer variables as Lorentz cone members. CBLIB instances do not have integral variables as Lorentz cone members.

Disjunctive cuts of Belotti et al. [Bel+15] help reducing the number of nodes. We tested two extreme strategies, adding only the best cut in terms of bound reduction and adding all possible cuts. Adding all cuts perform the best in terms of number of nodes. These results are obtained using randomly generated problem instances by Góez [Góe13]. We observed that problem instances in this set are either very easy or very hard. Better results on disjunctive cuts might be obtained from a more refined problem set.

bb-lp algorithm performs better in case of disjunctive cuts. This is due to fact that the cones are not in the formulation of the subproblems. They are used at cut generation. The difficulty of the subproblems increase less compared to the bb-socp case. Effect of disjunctive cuts on number of nodes is similar for both bb-lp and bb-socp.

Experiments in this chapter shows that DisCO scales well for both bb-lp and bb-socp algorithms. We conducted experiments with up to 2 nodes and 30 processors. DisCO scalibility does not vary depending on the underlying algorithm. Its scalibility performance is same for both bb-lp and bb-socp algorithms. Scalibility of branch-and-bound algorithm is not easy to achieve. DisCO can be considered as a pioneer in the field of scalable MISOCP solvers.

During branch-and-bound algorithm, CPLEX, Ipopt and Mosek, when used as a SOCP solver encounters numerical problems on some subproblems. Preconditioning of the subproblem, or starting with a different starting point (in case of Ipopt) might be necessary for those cases which we do not apply.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the work and its conclusions, as well as discuss future research directions. In Chapter 2, we describe a procedure that separates points from Lorentz cones. This procedure generates hyperplanes that support the conic constraint and define the strongest possible valid inequalities. Chapter 2 also introduces a cutting-plane algorithm for solving SOCP based on this procedure.

In Chapter 3, we first survey some of the literature on valid inequalities for MISOCP feasible set. Studies suggest various methods to remove continuous relaxation solutions from MISOCP feasible set. Two lines of work very relevant to this study are Belotti et al. [Bel+13] and Kılınç-Karzan and Yıldız [KY14]. Belotti et al. [Bel+13] give disjunctive conic cuts for general MISOCP and Kılınç-Karzan and Yıldız [KY14] give valid inequalities for the two-term disjunctions on a Lorentz cone. Even though both Kılınç-Karzan and Yıldız [KY14] and Belotti et al. [Bel+13] solve very similar problems, their derivation and the theory behind them are very different. A compelling future research direction is to investigate the relationship of the valid inequalities given by these two different line of work.

Chapter 3 proposes a flexible branch-and-cut framework that supports a variety of

relaxation-based algorithms. It can employ either SOCP relaxations or use the approximating valid inequalities given in Chapter 2. In this algorithm, both integrality and conic constraints may be relaxed. For relaxation solutions, both integer feasibility and conic feasibility are checked and both valid inequalities, as well as branching are used to address infeasibilities. When conic constraints are relaxed, the outer approximation approach proposed for enforcing conic feasibility is different than the existing ones in the literature. The existing methods approximate the problem a priori, whereas the proposed algorithm builds a polyhedral approximation dynamically. Contrary to the methods in literature, approximation procedure is applied to the cones directly without decomposing the cones first. An intriguing research direction is to use cone decomposition before applying the proposed algorithm.

Chapter 4 introduces an extensive framework for solving MISOCP. This framework includes a solver interface, solver implementations, a cut library and a MISOCP solver called DisCO. All software projects presented are open source and will be distributed under COIN-OR Initiative. The software framework implements the disjunctive procedure of Belotti et al. [Bel+13]. Possible future directions include implementing more of the cut procedures reviewed in Chapter 3 and testing their practical performance.

In Chapter 5 we prove the complexity of the inverse MILP problem. The main contribution of this study is to show that a certain decision version of the inverse problem is co-NP-complete. This broad-ranging implications and some of this work may be generalized in order to provide a different view of the equivalence of optimization and separation given by the well-known result of Grötschel–Lovász–Schrijver.

Chapter 6 present the computational experiments conducted. We compare the algorithms introduced and answer the questions raised around these algorithms. We observe that SOCP relaxations perform better that LP based relaxations in case of problems with large dimensional cones. The performance of the two different approach is close to each

other in case of problems with low dimensional cones. We observe that both of the algorithms scale well with the increasing number of processors. A future work in terms of the experiments is trying DisCO with more processors. The framework DisCO depends has been tried with up to 2000 cores. Experimenting DisCO in such a scale might crack the combinatorially challenging problems of CBLIB. DisCO does not implement reliability branching yet. Implementing and testing it is another future research direction.

Bibliography

[ALL10] Kumar Abhishek, Sven Leyffer, and Jeff Linderoth. "FilMINT: an outer approximation-based solver for convex mixed-integer nonlinear programs". In: *INFORMS Journal on Computing* 22.4 (2010), pp. 555–567 (cit. on pp. 70, 71, 193, 204).

[Ach07] Tobias Achterberg. "Constraint Integer Programming". PhD thesis. Technische Universität Berlin, 2007 (cit. on pp. 33, 34, 96).

[AA95] Ilan Adler and Farid Alizadeh. *Primal-dual interior point algorithms for convex quadratically constrained and semidefinite optimization problems*. Tech. rep. 1995 (cit. on p. 48).

[AO01] Ravindra K. Ahuja and James B. Orlin. "Inverse optimization". In: *Operations Research* 49.5 (2001), pp. 771–783 (cit. on pp. 21, 22, 149, 157).

[AAG09] Mehmet Selim Aktürk, Alper Atamtürk, and Sinan Gürel. "A strong conic quadratic reformulation for machine-job assignment with controllable processing times". In: *Operations Research Letters* 37.3 (2009), pp. 187–191 (cit. on p. 9).

[AG03] Farid Alizadeh and Donald Goldfarb. "Second-order cone programming". In: *Mathematical Programming* 95.1 (2003), pp. 3–51 (cit. on p. 9).

BIBLIOGRAPHY

[ART03] Erling D. Andersen, Kees Roos, and Tamás Terlaky. "On implementing a primal-dual interior-point method for conic quadratic optimization". English. In: *Mathematical Programming* 95.2 (2003), pp. 249–277 (cit. on pp. 47, 49).

[AJ13] Kent Andersen and Anders Nedergaard Jensen. "Intersection cuts for mixed integer conic quadratic sets". In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer. 2013, pp. 37–48 (cit. on p. 86).

[App+95] David Applegate, Robert Bixby, Vasek Chvatal, and William J. Cook. *Finding cuts in the TSP (a preliminary report)*. Tech. rep. 1995 (cit. on pp. 33, 95).

[ABS12] Alper Atamtürk, Gemma Berenguer, and Zuo-Jun Shen. "A conic integer programming approach to stochastic joint location-inventory problems". In: *Operations Research* 60.2 (2012), pp. 366–381 (cit. on pp. 9, 65).

[AN10] Alper Atamtürk and Vishnu Narayanan. "Conic mixed-integer rounding cuts". English. In: *Mathematical Programming* 122.1 (2010), pp. 1–20 (cit. on pp. 72, 73, 76, 118).

[Bal75] Egon Balas. "Facets of the knapsack polytope". In: *Mathematical Programming* 8.1 (1975), pp. 146–164 (cit. on p. 37).

[Bal79] Egon Balas. "Disjunctive programming". In: *Annals of Discrete Mathematics* 5 (1979), pp. 3–51 (cit. on pp. 29, 93).

[Bal85] Egon Balas. "Disjunctive programming and a hierarchy of relaxations for discrete optimization problems". In: *SIAM Journal on Algebraic Discrete Methods* 6.3 (1985), pp. 466–486 (cit. on p. 29).

[BCC96] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. "Mixed 0-1 programming by lift-and-project in a branch-and-cut framework". In: *Management Science* 42.9 (1996), pp. 1229–1246 (cit. on p. 38).

BIBLIOGRAPHY

[Bal+96] Egon Balas, Sebastian Ceria, Gérard Cornuéjols, and N. Natraj. "Gomory cuts revisited". In: *Operations Research Letters* 19.1 (1996), pp. 1–9 (cit. on pp. 38, 93).

[Bea] John E. Beasley. *OR library: Euclidean Steiner tree problems.* `http://people.brunel.ac.uk/mastjjb/jeb/orlib/esteininfo.html` (cit. on pp. 171, 172).

[Bel+13] Pietro Belotti, Julio C. Góez, Imre Pólik, Ted K. Ralphs, and Tamás Terlaky. "On families of quadratic surfaces having fixed intersections with two hyperplanes". In: *Discrete Applied Mathematics* 161.16–17 (2013), pp. 2778–2793 (cit. on pp. 19, 73, 82, 84–86, 94, 99, 117, 118, 175, 193, 194, 206, 207).

[Bel+15] Pietro Belotti, Julio C. Góez, Imre Pólik, Ted K. Ralphs, and Tamás Terlaky. "A conic representation of the convex hull of disjunctive sets and conic cuts for integer second order cone optimization". In: *Numerical Analysis and Optimization*. 2015, pp. 1–35 (cit. on pp. 89, 93, 94, 99, 205).

[Bel+09] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. "Branching and bounds tightening techniques for non-convex MINLP". In: *Optimization Methods and Software* 24.4-5 (2009), pp. 597–634 (cit. on p. 72).

[Bén+71] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and O. Vincent. "Experiments in mixed-integer linear programming". In: *Mathematical Programming* 1.1 (1971), pp. 76–94 (cit. on pp. 33, 95, 124).

[Ben98] Harold P. Benson. "An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem". In: *Journal of Global Optimization* 13.1 (1998), pp. 1–24 (cit. on p. 40).

BIBLIOGRAPHY

[BN98] Aharon Ben-Tal and Arkadi Nemirovski. “Robust convex optimization”. In: *Mathematics of Operations Research* 23.4 (1998), pp. 769–805 (cit. on p. 9).

[BN01a] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on Modern Convex Optimization*. Society for Industrial and Applied Mathematics, 2001 (cit. on pp. 13, 48).

[BN01b] Aharon Ben-Tal and Arkadi Nemirovski. “On polyhedral approximations of the second-order cone”. In: *Mathematics of Operations Research* 26.2 (2001), pp. 193–205 (cit. on pp. 40, 48, 51, 61, 67).

[BV97] Ethan Bernstein and Umesh Vazirani. “Quantum complexity theory”. In: *SIAM Journal on Computing* 26.5 (1997), pp. 1411–1473 (cit. on p. 23).

[BFL07] Livio Bertacco, Matteo Fischetti, and Andrea Lodi. “A feasibility pump heuristic for general mixed-integer problems”. In: *Discrete Optimization* 4.1 (2007), pp. 63–76 (cit. on p. 35).

[BS09] Dimitris Bertsimas and Romy Shioda. “Algorithm for cardinality-constrained quadratic optimization”. In: *Computational Optimization and Applications* 43.1 (2009), pp. 1–22 (cit. on pp. 9, 65).

[Bix+00] Robert Bixby, Mary Fenelon, Zonghao Gu, Edward Rothberg, and Roland Wunderling. “MIP: theory and practice–closing the gap”. In: *System Modelling and Optimization*. 2000, pp. 19–49 (cit. on pp. 38, 93).

[Bon+08] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya, et al. “An algorithmic framework for convex mixed integer nonlinear programs”. In: *Discrete Optimization* 5.2 (2008), pp. 186–204 (cit. on pp. 70, 71).

BIBLIOGRAPHY

[Boy94] Andrew Boyd. "Fenchel cutting planes for integer programs". In: *Operations Research* 42.1 (1994), pp. 53–64 (cit. on p. 153).

[BR15] Aykut Bulut and Ted K. Ralphs. *On the complexity of inverse mixed integer linear optimization*. Tech. rep. COR@L Laboratory Report 15T-001-R3, Lehigh University, 2015 (cit. on p. 147).

[BR16a] Aykut Bulut and Ted K. Ralphs. *Conic optimization using linear approximation*. `https://github.com/aykutbulut/COLA`. 2016 (cit. on p. 113).

[BR16b] Aykut Bulut and Ted K. Ralphs. *Conic solver interface for CPLEX*. `https://github.com/aykutbulut/OsiCplex`. 2016 (cit. on p. 112).

[BR16c] Aykut Bulut and Ted K. Ralphs. *Conic solver interface for Ipopt*. `https://github.com/aykutbulut/OsiIpopt`. 2016 (cit. on p. 112).

[BR16d] Aykut Bulut and Ted K. Ralphs. *Conic solver interface for Mosek*. `https://github.com/aykutbulut/OSI-MOSEK`. 2016 (cit. on p. 112).

[BR16e] Aykut Bulut and Ted K. Ralphs. *Cut generation library for conic problems*. `https://github.com/aykutbulut/CGL-CONIC`. 2016 (cit. on p. 117).

[BR16f] Aykut Bulut and Ted K. Ralphs. *Discrete conic optimization solver library*. `https://github.com/aykutbulut/DisCO`. 2016 (cit. on p. 123).

[BR16g] Aykut Bulut and Ted K. Ralphs. *Solver interface for conic problems*. `https://github.com/aykutbulut/OSI-CONIC`. 2016 (cit. on p. 110).

[BR] Aykut Bulut and Ted K. Ralphs. *Graphics for understanding mathematical programming in Python*. `https://github.com/aykutbulut/CHiPPS-BiCePS` (cit. on p. 137).

[CYZ99] M. C. Cai, X. G. Yang, and J. Z. Zhang. "The complexity analysis of the inverse center location problem". In: *Journal of Global Optimization* 15.2 (1999), pp. 213–218 (cit. on p. 148).

BIBLIOGRAPHY

[CG96] Alberto Caprara and Juan J. Salazar González. "A branch-and-cut algorithm for a generalization of the uncapacitated facility location problem". In: *Top* 4.1 (1996), pp. 135–163 (cit. on p. 37).

[ÇPT17] Sertalp Çay, Imre Pólik, and Tamás Terlaky. *Warm-start of interior point methods for second order cone optimization via rounding over optimal Jordan frames*. Tech. rep. COR@L Laboratory Report 17T-006, Lehigh University, 2017 (cit. on p. 63).

[ÇI05] Mehmet T. Çezik and Garud Iyengar. "Cuts for mixed 0-1 conic programming". English. In: *Mathematical Programming* 104.1 (2005), pp. 179–202 (cit. on pp. 72, 73, 77, 93, 118).

[CGH94] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. "The MPI message passing interface standard". In: *Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994*. 1994, pp. 213–218 (cit. on p. 127).

[Cor07] Gérard Cornuéjols. "Revival of the Gomory cuts in the 1990's". In: *Annals of Operations Research* 149.1 (2007), pp. 63–66 (cit. on p. 38).

[Cor08] Gérard Cornuéjols. "Valid inequalities for mixed integer linear programs". In: *Mathematical Programming* 112.1 (2008), pp. 3–44 (cit. on p. 20).

[Cut80] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980 (cit. on p. 23).

[DRP05] Emilie Danna, Edward Rothberg, and Claude Le Pape. "Exploring relaxation induced neighborhoods to improve MIP solutions". In: *Mathematical Programming* 102.1 (2005), pp. 71–90 (cit. on p. 35).

BIBLIOGRAPHY

[DM02] Elizabeth D. Dolan and Jorge J. Moré. "Benchmarking optimization software with performance profiles". In: *Mathematical Programming* 91.2 (2002), pp. 201–213 (cit. on p. 171).

[Dre09] Sarah Drewes. "Mixed Integer Second Order Cone Programming". PhD thesis. Technische Universitat Darmstadt, 2009 (cit. on pp. 9, 77–79, 172).

[DG86] Marco A. Duran and Ignacio E. Grossmann. "An outer-approximation algorithm for a class of mixed-integer nonlinear programs". In: *Mathematical Programming* 36.3 (1986), pp. 307–339 (cit. on pp. 40, 70, 71).

[FM04] Marcia Fampa and Nelson Maculan. "Using a conic formulation for finding Steiner minimal trees". In: *Numerical Algorithms* 35.2-4 (2004), pp. 315–330 (cit. on pp. 9, 65, 172).

[Fis94] Matteo Fischetti. *Odd cut-sets, odd cycles, and 0-1/2 Chvatal-Gomory cuts.* Tech. rep. Univ. of Michigan, Ann Arbor, MI (United States), 1994 (cit. on p. 37).

[FL94] Roger Fletcher and Sven Leyffer. "Solving mixed integer nonlinear programs by outer approximation". English. In: *Mathematical Programming* 66.1-3 (1994), pp. 327–349 (cit. on pp. 40, 70).

[FLT02] Roger Fletcher, Sven Leyffer, and Philippe L. Toint. "On the global convergence of a filter–SQP algorithm". In: *SIAM Journal on Optimization* 13.1 (2002), pp. 44–59 (cit. on p. 71).

[For+a] John Forrest, Stefan Vigerske, Ted K. Ralphs, Pierre Bonami, John P. Fasano, and Yan Xu. *COIN-OR cut generation library.* `https://projects.coin-or.org/Cgl` (cit. on p. 108).

[For+b] John Forrest, Stefan Vigerske, Ted K. Ralphs, John P. Fasano, and Matthew Saltzman. *COIN-OR build tools*. `https://projects.coin-or.org/BuildTools` (cit. on p. 108).

[For+c] John Forrest, Stefan Vigerske, Ted K. Ralphs, John P. Fasano, and Matthew Saltzman. *COIN-OR linear programming solver*. `https://projects.coin-or.org/Clp` (cit. on p. 108).

[For+d] John Forrest, Stefan Vigerske, Ted K. Ralphs, John P. Fasano, and Matthew Saltzman. *COIN-OR utilities*. `https://projects.coin-or.org/CoinUtils` (cit. on p. 108).

[For+e] John Forrest, Stefan Vigerske, Ted K. Ralphs, John P. Fasano, Matthew Saltzman, and Brady Hunsaker. *COIN-OR open solver interface*. `https://projects.coin-or.org/Osi` (cit. on p. 108).

[Fri16] Henrik A. Friberg. "CBLIB 2014: a benchmark library for conic mixed-integer and continuous optimization". In: *Mathematical Programming Computation* 8.2 (2016), pp. 191–214 (cit. on pp. 45, 171).

[FM05a] Armin Fügenschuh and Alexander Martin. "Computational integer programming and cutting planes". In: *Discrete Optimization*. Vol. 12. Supplement C. 2005, pp. 69–121 (cit. on p. 38).

[FM05b] Armin Fügenschuh and Alexander Martin. "Computational integer programming and cutting planes". In: *Handbooks in Operations Research and Management Science* 12 (2005), pp. 69–121 (cit. on p. 124).

[Gab+04] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. "Open MPI: goals, concept, and design

of a next generation MPI implementation". In: *Proceedings, 11th European PVM/MPI Users' Group Meeting*. 2004, pp. 97–104 (cit. on pp. 127, 175).

[GL97] Laurent El Ghaoui and Hervé Lebret. "Robust solutions to least-squares problems with uncertain data". In: *SIAM Journal on Matrix Analysis and Applications* 18.4 (1997), pp. 1035–1064 (cit. on p. 9).

[Gli00] François Glineur. *Computational experiments with a linear approximation of second-order cone optimization*. 2000 (cit. on pp. 53, 62, 63, 67).

[Góe13] Julio C. Góez. "Mixed Integer Second Order Cone Optimization Disjunctive Conic Cuts: Theory and Experiments". PhD thesis. Lehigh University, 2013 (cit. on pp. 45, 171, 173, 205).

[Gom58] Ralph E. Gomory. "Outline of an algorithm for integer solutions to linear programs". In: *Bull. Amer. Math. Soc.* 64.5 (1958), pp. 275–278 (cit. on pp. 37, 93).

[Gom60] Ralph E. Gomory. "Solving linear programming problems in integers". In: *Combinatorial Analysis* 10 (1960), pp. 211–215 (cit. on p. 20).

[Gro02] William Gropp. "MPICH2: A new start for MPI implementations". In: *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. 2002, pp. 7– (cit. on p. 127).

[GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Second corrected edition. Vol. 2. Springer, 1993 (cit. on pp. 21, 22, 36, 150, 157, 158, 169).

[HJP75] Peter L. Hammer, Ellis L. Johnson, and Uri N. Peled. "Facet of regular 0–1 polytopes". In: *Mathematical Programming* 8.1 (1975), pp. 179–206 (cit. on p. 37).

BIBLIOGRAPHY

[Heu04] Clemens Heuberger. "Inverse combinatorial optimization: a survey on problems, methods, and results". In: *Journal of Combinatorial Optimization* 8.3 (2004), pp. 329–361 (cit. on p. 147).

[Hua05] Siming Huang. "Inverse problems of some NP-complete problems". In: *Algorithmic Applications in Management: First International Conference, AAIM 2005, Xian, China, June 22-25, 2005. Proceedings*. 2005, pp. 422–426 (cit. on p. 148).

[KB79] Ravindran Kannan and Achim Bachem. "Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix". In: *SIAM Journal on Computing* 8.4 (1979), pp. 499–507 (cit. on p. 19).

[Kea03] Ralph Baker Kearfott. "GlobSol: history, composition, and advice on use". In: *Global Optimization and Constraint Satisfaction: First International Workshop on Global Constraint Optimization and Constraint Satisfaction, COCOS 2002, Valbonne-Sophia Antipolis, France, October 2002. Revised Selected Papers*. 2003, pp. 17–31 (cit. on p. 72).

[KY14] Fatma Kılınç-Karzan and Sercan Yıldız. "Two-term disjunctions on the second-order cone". In: *Integer Programming and Combinatorial Optimization*. Vol. 8494. 2014, pp. 345–356 (cit. on pp. 73, 86, 88, 89, 93, 118, 206).

[KTD56] Harold William Kuhn, Albert William Tucker, and George Bernard Dantzig. *Linear Inequalities and Related Systems*. 38. Princeton University Press, 1956 (cit. on p. 49).

[LD60] Ailsa H. Land and Alison G. Doig. "An automatic method of solving discrete programming problems". In: *Econometrica* 28.3 (1960), pp. 497–520 (cit. on p. 28).

[Leb09] Yahia Lebbah. "ICOS: a branch and bound based solver for rigorous global optimization". In: *Optimization Methods and Software* 24.4-5 (2009), pp. 709–726 (cit. on p. 72).

[LS99] Jeff T. Linderoth and Martin W. P. Savelsbergh. "A computational study of search strategies for mixed integer programming". In: *INFORMS Journal on Computing* 11.2 (1999), pp. 173–187 (cit. on pp. 95, 124).

[LR05] Jeffrey T. Linderoth and Ted K. Ralphs. "Noncommercial software for mixed-integer linear programming". In: *Integer Programming: Theory and Practice*. 2005, pp. 253–303 (cit. on p. 20).

[Lob+98] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. "Applications of second-order cone programming". In: *Linear Algebra and its Applications* 284.1 (1998), pp. 193–228 (cit. on pp. 9, 11).

[LAL99] Andrea Lodi, Kim Allemand, and Thomas M. Liebling. "An evolutionary heuristic for quadratic 0–1 programming". In: *European Journal of Operational Research* 119.3 (1999), pp. 662–670 (cit. on p. 35).

[Lou03] Robin Lougee-Heimer. "The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community". In: *IBM Journal of Research and Development* 47.1 (2003), pp. 57–66 (cit. on pp. 10, 108).

[Mar+02] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. "Cutting planes in integer and mixed integer programming". In: *Discrete Applied Mathematics* 123.1 (2002), pp. 397–446 (cit. on p. 38).

[MF14] Ruth Misener and Christodoulos A. Floudas. "ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization of Nonlinear Equations". In: *Jour-*

nal of Global Optimization (2014). DOI: 10.1007/s10898-014-0166-2 (cit. on p. 72).

[Mit73] Gautam Mitra. "Investigation of some branch and bound strategies for the solution of mixed integer linear programs". In: *Mathematical Programming* 4.1 (1973), pp. 155–170 (cit. on p. 124).

[MKV15] Sina Modaresi, Mustafa R. Kılınç, and Juan Pablo Vielma. "Split cuts and extended formulations for mixed integer conic quadratic programming". In: *Operations Research Letters* 43.1 (2015), pp. 10–15 (cit. on p. 93).

[MOS15] MOSEK ApS. *The MOSEK release notes, version 7.1.* 2015 (cit. on p. 66).

[MS83] Bruce A. Murtagh and Michael A. Saunders. *MINOS 5.51 user's guide.* Tech. rep. Systems Optimization Laboratory, Stanford University, 1983 (cit. on p. 72).

[NSS94] George L. Nemhauser, Martin W. P. Savelsbergh, and Gabriele C. Sigismondi. "MINTO, a mixed integer optimizer". In: *Operations Research Letters* 15.1 (1994), pp. 47–58 (cit. on p. 71).

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience, 1988 (cit. on pp. 20, 98).

[NW90] George L. Nemhauser and Laurence A. Wolsey. "A recursive procedure to generate all cuts for 0–1 mixed integer programs". In: *Mathematical Programming* 46.1 (1990), pp. 379–390 (cit. on pp. 37, 73).

[NS96] Arkadii Nemirovskii and Katya Scheinberg. "Extension of Karmarkar's algorithm onto convex quadratically constrained quadratic problems". In: *Mathematical Programming* 72.3 (1996), pp. 273–289 (cit. on p. 48).

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, 1994 (cit. on p. 11).

BIBLIOGRAPHY

[Neu+05] Arnold Neumaier, Oleg Shcherbina, Waltraud Huyer, and Tamás Vinkó. "A comparison of complete global optimization solvers". In: *Mathematical Programming* 103.2 (2005), pp. 335–356 (cit. on p. 72).

[OHR07] Osman Y. Ozaltin, Brady Hunsaker, and Ted K. Ralphs. *Visualizing branch-and-bound algorithms*. Tech. rep. COR@L Laboratory, Lehigh Univesity, 2007 (cit. on p. 137).

[Pad75] Manfred W. Padberg. "A note on zero-one programming". In: *Operations Research* 23.4 (1975), pp. 833–837 (cit. on p. 37).

[PRW85] Manfred W. Padberg, Tony J. Van Roy, and Laurence A. Wolsey. "Valid linear inequalities for fixed charge problems". In: *Operations Research* 33.4 (1985), pp. 842–861 (cit. on p. 37).

[Pap84] Christos H. Papadimitriou. "On the complexity of unique solutions". In: *Journal of the ACM* 31.2 (1984), pp. 392–400 (cit. on p. 27).

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994 (cit. on p. 26).

[PY82] Christos H. Papadimitriou and Mihalis Yannakakis. "The complexity of facets (and some facets of complexity)". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. 1982, pp. 255–260 (cit. on pp. 25, 27, 156, 157).

[Pin98] János D. Pintér. "A model development system for global optimization". In: *High Performance Algorithms and Software in Nonlinear Optimization*. 1998, pp. 301–314 (cit. on p. 72).

[Pin13] János D. Pintér. *Global Optimization in Action: Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications*. Vol. 6. Springer Science & Business Media, 2013 (cit. on p. 39).

BIBLIOGRAPHY

[QG92] Ignacio Quesada and Ignacio E. Grossmann. "An LP/NLP based branch and bound algorithm for convex MINLP optimization problems". In: *Computers & Chemical Engineering* 16.10 (1992), pp. 937–947 (cit. on p. 70).

[Qui+98] Arie J. Quist, Etienne de Klerk, Cornelis Roos, and Tamás Terlaky. "Copositive realxation for general quadratic programming". In: *Optimization Methods and Software* 9.1-3 (1998), pp. 185–208 (cit. on p. 11).

[RLS04] Ted K. Ralphs, Laszlo Ladanyi, and Matthew J. Saltzman. "A library hierarchy for implementing scalable parallel search algorithms". In: *The Journal of Supercomputing* 28.2 (2004), pp. 215–234 (cit. on pp. 30, 109, 125).

[RW86] Tony J. Van Roy and Laurence A. Wolsey. "Valid inequalities for mixed 0–1 programs". In: *Discrete Applied Mathematics* 14.2 (1986), pp. 199–213 (cit. on p. 37).

[Sah17] Nikolaos V. Sahinidis. *BARON 17.8.9: global optimization of mixed-integer nonlinear programs,* user's manual. 2017 (cit. on p. 72).

[Sav94] Martin W. P. Savelsbergh. "Preprocessing and probing techniques for mixed integer programming problems". In: *ORSA Journal on Computing* 6.4 (1994), pp. 445–454 (cit. on p. 37).

[Sch09] Andrew J. Schaefer. "Inverse integer programming". In: *Optimization Letters* 3.4 (2009), pp. 483–489 (cit. on p. 148).

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., 1986 (cit. on p. 145).

[SS63] John C. Shepherdson and Howard E. Sturgis. "Computability of recursive functions". In: *Journal of the ACM* 10.2 (1963), pp. 217–255 (cit. on p. 23).

BIBLIOGRAPHY

[SAY13] Anders Skajaa, Erling D. Andersen, and Yinyu Ye. "Warmstarting the homogeneous and self-dual interior point method for linear and conic quadratic problems". In: *Mathematical Programming Computation* 5.1 (2013), pp. 1–25 (cit. on pp. 63, 92).

[SM99] Robert A. Stubbs and Sanjay Mehrotra. "A branch-and-cut method for 0-1 mixed convex programming". English. In: *Mathematical Programming* 86.3 (1999), pp. 515–532 (cit. on pp. 72, 73, 79, 93, 118).

[TS05] Mohit Tawarmalani and Nikolaos V. Sahinidis. "A polyhedral branch-and-cut approach to global optimization". In: *Mathematical Programming* 103 (2 2005), pp. 225–249 (cit. on p. 72).

[Tur37] Alan M. Turing. "On computable numbers, with an application to the entscheidungsproblem". In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265 (cit. on p. 23).

[VAN08] Juan Pablo Vielma, Shabbir Ahmed, and George L. Nemhauser. "A lifted linear programming branch-and-bound algorithm for mixed-integer conic quadratic programs". In: *INFORMS Journal on Computing* 20.3 (2008), pp. 438–450 (cit. on pp. 63, 66–69, 105, 106).

[VG16] Stefan Vigerske and Ambros Gleixner. *SCIP: global optimization of mixed-integer nonlinear programs in a branch-and-cut framework*. eng. Tech. rep. 16-24. ZIB, 2016 (cit. on p. 72).

[WB06] Andreas Wächter and Lorenz T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical Programming* 106.1 (2006), pp. 25–57 (cit. on p. 108).

BIBLIOGRAPHY

[Wan09] Lizhi Wang. "Cutting plane algorithms for the inverse mixed integer linear programming problem". In: *Operations Research Letters* 37.2 (2009), pp. 114–116 (cit. on pp. 148, 150).

[Xu07] Yan Xu. "Scalable Algorithms for Parallel Tree Search". PhD thesis. Lehigh University, 2007 (cit. on pp. 35, 97).

[Xu+05] Yan Xu, Ted K. Ralphs, Laszlo Ladanyi, and Matthew J. Saltzman. "ALPS: a framework for implementing parallel tree search algorithms". In: *The Next Wave in Computing, Optimization, and Decision Technologies*. 2005, pp. 319–334 (cit. on pp. 30, 97, 109, 126).

[Xu+09] Yan Xu, Ted K. Ralphs, Laszlo Ladanyi, and Matthew J. Saltzman. "Computational experience with a software framework for parallel integer programming". In: *INFORMS Journal on Computing* 21.3 (2009), pp. 383–397 (cit. on p. 109).

[XRV] Yan Xu, Ted K. Ralphs, and Stefan Vigerske. *COIN-OR the BiCePS linear integer solver*. `https://projects.coin-or.org/CHiPPS` (cit. on p. 130).

[Xu+a] Yan Xu, Ted K. Ralphs, Stefan Vigerske, and Aykut Bulut. *COIN-OR abstract library for parallel search*. `https://github.com/aykutbulut/CHiPPS-ALPS` (cit. on p. 109).

[Xu+b] Yan Xu, Ted K. Ralphs, Stefan Vigerske, and Aykut Bulut. *COIN-OR branch, constrain and price software library*. `https://github.com/aykutbulut/CHiPPS-BiCePS` (cit. on pp. 109, 127).

Appendix A

Details of Computational Results

Table A.1: bb-socp CPU Time and Number of Nodes with Various Solvers Part 1

	disco-cola		disco-cplex		disco-ipopt		disco-mosek	
problem	CPU time	node	CPU time	node	CPU time	node	CPU time	node
classical_200_1	TL	TL	TL	TL	TL	TL	SF	SF
classical_200_2	TL	TL	TL	TL	TL	TL	SF	SF
classical_200_3	TL	TL	SF	SF	TL	TL	SF	SF
classical_50_1	TL	TL	24.04	1713	439.97	2635	49.48	5899
classical_50_2	TL	TL	60.35	3945	1025.18	6693	83.33	10897
classical_50_3	TL	TL	149.14	10259	2973.74	18085	SF	SF
estein4_A	0.14	31	SF	SF	1.35	31	0.07	31
estein4_B	0.12	31	SF	SF	1.52	31	0.07	31
estein4_C	0.1	31	SF	SF	1.87	31	0.07	31
estein4_nr22	0.05	31	SF	SF	1.7	31	0.06	31
estein5_A	35.33	785	1.96	785	SF	SF	2.25	785
estein5_B	14.64	445	1.18	445	44.36	445	1.2	445
estein5_C	14.96	635	1.54	635	SF	SF	1.85	635
estein5_nr1	36.0	653	1.67	649	SF	SF	1.82	649
estein5_nr21	43.77	785	1.96	785	65.44	785	1.98	785
estein6_0	5204.34	27647	91.61	27789	SF	SF	84.58	27891
estein6_1	3070.1	14503	SF	SF	SF	SF	SF	SF
estein6_2	2952.9	19043	66.18	19531	SF	SF	SF	SF
estein7_0	TL	TL	5313.37	990533	SF	SF	SF	SF
estein7_1	TL	TL	1824.74	340935	SF	SF	SF	SF
estein7_2	TL	TL	3098.51	598215	SF	SF	SF	SF
pp-n10-d10	1.16	213	0.63	295	37.61	73	0.39	307
pp-n10-d10000	SF	SF	1.99	2049	SF	SF	SF	SF
pp-n100-d10	TL	TL	TL	TL	SF	SF	TL	TL
pp-n100-d10000	ML	ML	TL	TL	SF	SF	SF	SF
pp-n1000-d10	TL	TL	TL	TL	SF	SF	TL	TL
pp-n1000-d10000	TL	TL	TL	TL	SF	SF	ML	ML
pp-n100000-d10	TL	TL	SF	SF	SF	SF	TL	TL
pp-n100000-d10000	TL	TL	SF	SF	ML	ML	TL	TL
robust_100_1	TL	TL	1168.5	6109	ML	ML	1411.27	17939
robust_100_2	TL	TL	398.04	2141	3210.36	6077	SF	SF
robust_100_3	TL	TL	167.43	801	3068.68	5387	578.94	7379
robust_200_1	TL	TL	TL	TL	TL	TL	TL	TL
robust_200_2	TL	TL	6795.73	4883	TL	TL	TL	TL
robust_200_3	TL	TL	TL	TL	TL	TL	TL	TL
robust_50_1	TL	TL	2.34	59	402.17	1611	48.75	2723
robust_50_2	TL	TL	2.64	67	639.48	2573	46.48	2451
robust_50_3	934.93	153	5.45	143	330.34	1355	25.55	1315

Table A.2: bb-socp CPU Time and Number of Nodes with Various Solvers Part 2

	disco-cola		disco-cplex		disco-ipopt		disco-mosek	
problem	CPU time	node	CPU time	node	CPU time	node	CPU time	node
r12c15k5i10	0.01	63	0.03	63	SF	SF	0.08	63
r12c15k5i15	0.01	29	0.02	29	SF	SF	0.04	29
r14c18k3i12	0.3	489	0.45	501	SF	SF	SF	SF
r14c18k3i15	0.17	247	0.19	183	SF	SF	SF	SF
r14c18k3i18	0.06	149	0.16	149	SF	SF	SF	SF
r14c18k3i9	3.07	3013	2.22	3133	SF	SF	SF	SF
r17c20k5i15	0.02	33	0.05	33	SF	SF	0.05	33
r17c20k5i20	0.01	33	0.03	33	SF	SF	0.06	33
r17c30k3i12	2.8	51	0.1	79	SF	SF	0.13	73
r17c30k3i15	91.05	617	1.6	1145	SF	SF	1.16	747
r17c30k3i18	TL	TL	SF	SF	SF	SF	SF	SF
r17c30k3i21	TL	TL	SF	SF	SF	SF	SF	SF
r17c30k3i24	TL	TL	SF	SF	SF	SF	SF	SF
r17c30k3i27	TL	TL	SF	SF	SF	SF	SF	SF
r22c30k10i20	0.88	3295	2.31	3295	SF	SF	SF	SF
r22c40k10i20	0.07	19	0.03	23	SF	SF	0.06	23
r22c40k10i30	3.47	1341	1.8	1333	SF	SF	SF	SF
r22c40k10i40	11.18	12141	20.14	16037	SF	SF	SF	SF
r23c45k3i21	416.1	447	1.08	529	SF	SF	1.15	567
r23c45k3i24	882.8	673	2.13	1003	SF	SF	SF	SF
r23c45k3i27	TL	TL	SF	SF	SF	SF	SF	SF
r23c45k3i30	TL	TL	SF	SF	SF	SF	SF	SF
r23c45k3i33	TL	TL	SF	SF	SF	SF	SF	SF
r23c45k3i36	TL	TL	SF	SF	SF	SF	SF	SF
r27c50k5i25	84.78	459	1.48	691	SF	SF	SF	SF
r27c50k5i30	TL	TL	TL	TL	SF	SF	SF	SF
r27c50k5i35	TL	TL	SF	SF	SF	SF	SF	SF
r27c50k5i40	TL	TL	SF	SF	SF	SF	SF	SF
r27c50k5i45	TL	TL	SF	SF	SF	SF	SF	SF
r27c50k5i50	TL	TL	SF	SF	SF	SF	SF	SF
r32c45k15i30	0.06	173	0.24	173	SF	SF	0.29	173
r32c45k15i45	0.23	635	0.67	635	SF	SF	0.74	631
r32c60k15i30	0.68	125	0.24	127	SF	SF	0.39	193
r32c60k15i45	3951.94	428731	1186.22	576477	SF	SF	SF	SF
r32c60k15i60	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i35	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i40	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i45	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i50	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i60	TL	TL	SF	SF	SF	SF	SF	SF
r52c75k5i65	TL	TL	SF	SF	SF	SF	SF	SF

Table A.3: bb-socp CPU Time and Number of Nodes with Various Solvers Part 3

	disco-cola		disco-cplex		disco-ipopt		disco-mosek	
problem	CPU time	node	CPU time	node	CPU time	node	CPU time	node
shortfall_100_1	TL	TL	1717.76	9221	ML	ML	TL	TL
shortfall_100_2	TL	TL	TL	TL	ML	ML	TL	TL
shortfall_100_3	TL	TL	5440.13	22915	ML	ML	2869.3	36959
shortfall_200_1	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_50_1	TL	TL	24.55	685	531.18	2199	70.66	3579
shortfall_50_2	TL	TL	77.44	2245	1530.77	6381	154.82	8093
shortfall_50_3	TL	TL	SF	SF	3150.97	11075	186.0	10687
sssd-strong-15-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-strong-15-8	TL	TL	TL	TL	SF	SF	TL	TL
sssd-strong-20-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-strong-20-8	TL	TL	TL	TL	SF	SF	SF	SF
sssd-strong-25-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-strong-25-8	TL	TL	TL	TL	SF	SF	SF	SF
sssd-strong-30-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-strong-30-8	TL	TL	TL	TL	SF	SF	TL	TL
sssd-weak-15-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-weak-15-8	TL	TL	TL	TL	SF	SF	SF	SF
sssd-weak-20-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-weak-20-8	TL	TL	TL	TL	SF	SF	SF	SF
sssd-weak-25-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-weak-25-8	TL	TL	SF	SF	SF	SF	SF	SF
sssd-weak-30-4	TL	TL	SF	SF	SF	SF	SF	SF
sssd-weak-30-8	TL	TL	TL	TL	SF	SF	SF	SF
turbine07GF	0.02	17	0.1	45	SF	SF	0.13	39
turbine07_aniso	0.01	1	0.01	1	0.18	1	0.02	1
turbine07_lowb	2.99	241	SF	SF	SF	SF	SF	SF
turbine07_lowb_aniso	2.98	569	SF	SF	SF	SF	SF	SF
turbine54GF	0.07	13	0.52	29	SF	SF	TL	TL
uflquad-nopsc-10-100	1801.52	209	35.86	209	SF	SF	23.25	209
uflquad-nopsc-10-150	2409.0	185	53.83	185	ML	ML	38.65	185
uflquad-nopsc-20-100	TL	TL	SF	SF	SF	SF	1696.46	10609
uflquad-nopsc-20-150	TL	TL	TL	TL	SF	SF	4229.4	14825
uflquad-nopsc-30-100	TL	TL	SF	SF	ML	ML	1971.31	8603
uflquad-nopsc-30-150	TL	TL	TL	TL	SF	SF	ML	ML
uflquad-nopsc-30-200	TL	TL	TL	TL	ML	ML	ML	ML
uflquad-nopsc-30-300	TL	TL	TL	TL	ML	ML	ML	ML
uflquad-psc-10-100	15.72	9	1.63	7	304.52	1	2.07	13
uflquad-psc-10-150	35.28	7	2.98	7	SF	SF	SF	SF
uflquad-psc-20-100	49.83	15	8.05	15	SF	SF	SF	SF
uflquad-psc-20-150	109.48	11	10.4	11	SF	SF	11.3	25
uflquad-psc-30-100	138.7	23	17.28	21	SF	SF	22.25	35
uflquad-psc-30-150	100.82	3	1.55	1	SF	SF	26.35	29
uflquad-psc-30-200	319.62	7	15.73	7	SF	SF	SF	SF
uflquad-psc-30-300	1667.77	19	53.31	17	SF	SF	SF	SF

Table A.4: bb-socp CPU Time and Number of Nodes with Different Branching Strategies Part 1

	disco-cplex		disco-cplex-strong	
problem	CPU time	node	CPU time	node
classical_200_1	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL
classical_200_3	SF	SF	TL	TL
classical_50_1	24.04	1713	117.42	741
classical_50_2	60.35	3945	123.67	807
classical_50_3	149.14	10259	578.5	3391
estein4_A	SF	SF	SF	SF
estein4_B	SF	SF	SF	SF
estein4_C	SF	SF	SF	SF
estein4_nr22	SF	SF	SF	SF
estein5_A	1.96	785	10.4	785
estein5_B	1.18	445	7.41	373
estein5_C	1.54	635	8.28	387
estein5_nr1	1.67	649	8.67	407
estein5_nr21	1.96	785	10.14	785
estein6_0	91.61	27789	419.63	17399
estein6_1	SF	SF	SF	SF
estein6_2	66.18	19531	300.85	8621
estein7_0	5313.37	990533	TL	TL
estein7_1	1824.74	340935	TL	TL
estein7_2	3098.51	598215	SF	SF
pp-n10-d10	0.63	295	0.76	47
pp-n10-d10000	1.99	2049	21.06	2789
pp-n100-d10	TL	TL	TL	TL
pp-n100-d10000	TL	TL	TL	TL
pp-n1000-d10	TL	TL	TL	TL
pp-n1000-d10000	TL	TL	TL	TL
pp-n100000-d10	SF	SF	TL	TL
pp-n100000-d10000	SF	SF	TL	TL
robust_100_1	1168.5	6109	2232.6	1239
robust_100_2	398.04	2141	690.42	307
robust_100_3	167.43	801	488.73	239
robust_200_1	TL	TL	TL	TL
robust_200_2	6795.73	4883	TL	TL
robust_200_3	TL	TL	TL	TL
robust_50_1	2.34	59	17.03	25
robust_50_2	2.64	67	17.51	25
robust_50_3	5.45	143	23.97	41

Table A.5: bb-socp CPU Time and Number of Nodes with Different Branching Strategies Part 2

	disco-cplex		disco-cplex-strong	
problem	CPU time	node	CPU time	node
r12c15k5i10	0.03	63	0.07	57
r12c15k5i15	0.02	29	0.08	19
r14c18k3i12	0.45	501	0.54	139
r14c18k3i15	0.19	183	0.89	119
r14c18k3i18	0.16	149	1.02	109
r14c18k3i9	2.22	3133	TL	TL
r17c20k5i15	0.05	33	0.2	39
r17c20k5i20	0.03	33	0.36	39
r17c30k3i12	0.1	79	0.5	41
r17c30k3i15	1.6	1145	2.11	121
r17c30k3i18	SF	SF	SF	SF
r17c30k3i21	SF	SF	TL	TL
r17c30k3i24	SF	SF	TL	TL
r17c30k3i27	SF	SF	TL	TL
r22c30k10i20	2.31	3295	4.4	3153
r22c40k10i20	0.03	23	0.14	17
r22c40k10i30	1.8	1333	4.5	385
r22c40k10i40	20.14	16037	100.14	7151
r23c45k3i21	1.08	529	TL	TL
r23c45k3i24	2.13	1003	SF	SF
r23c45k3i27	SF	SF	SF	SF
r23c45k3i30	SF	SF	TL	TL
r23c45k3i33	SF	SF	SF	SF
r23c45k3i36	SF	SF	TL	TL
r27c50k5i25	1.48	691	4.93	133
r27c50k5i30	TL	TL	TL	TL
r27c50k5i35	SF	SF	TL	TL
r27c50k5i40	SF	SF	TL	TL
r27c50k5i45	SF	SF	TL	TL
r27c50k5i50	SF	SF	TL	TL
r32c45k15i30	0.24	173	0.2	29
r32c45k15i45	0.67	635	1.48	141
r32c60k15i30	0.24	127	1.76	73
r32c60k15i45	1186.22	576477	1404.82	209167
r32c60k15i60	SF	SF	4318.31	206719
r52c75k5i35	SF	SF	TL	TL
r52c75k5i40	SF	SF	TL	TL
r52c75k5i45	SF	SF	TL	TL
r52c75k5i50	SF	SF	TL	TL
r52c75k5i60	SF	SF	TL	TL
r52c75k5i65	SF	SF	TL	TL

Table A.6: bb-socp CPU Time and Number of Nodes with Different Branching Strategies Part 3

	disco-cplex		disco-cplex-strong	
problem	CPU time	node	CPU time	node
shortfall_100_1	1717.76	9221	TL	TL
shortfall_100_2	TL	TL	TL	TL
shortfall_100_3	5440.13	22915	TL	TL
shortfall_200_1	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL
shortfall_50_1	24.55	685	90.35	207
shortfall_50_2	77.44	2245	144.6	457
shortfall_50_3	SF	SF	853.94	2291
sssd-strong-15-4	SF	SF	TL	TL
sssd-strong-15-8	TL	TL	TL	TL
sssd-strong-20-4	SF	SF	1249.84	10441
sssd-strong-20-8	TL	TL	TL	TL
sssd-strong-25-4	SF	SF	282.6	1109
sssd-strong-25-8	TL	TL	TL	TL
sssd-strong-30-4	SF	SF	TL	TL
sssd-strong-30-8	TL	TL	TL	TL
sssd-weak-15-4	SF	SF	TL	TL
sssd-weak-15-8	TL	TL	TL	TL
sssd-weak-20-4	SF	SF	1253.12	8415
sssd-weak-20-8	TL	TL	TL	TL
sssd-weak-25-4	SF	SF	304.27	511
sssd-weak-25-8	SF	SF	TL	TL
sssd-weak-30-4	SF	SF	3739.48	26955
sssd-weak-30-8	TL	TL	TL	TL
turbine07GF	0.1	45	0.22	17
turbine07_aniso	0.01	1	0.0	1
turbine07_lowb	SF	SF	88.0	91
turbine07_lowb_aniso	SF	SF	SF	SF
turbine54GF	0.52	29	1.24	15
uflquad-nopsc-10-100	35.86	209	266.02	215
uflquad-nopsc-10-150	53.83	185	365.15	191
uflquad-nopsc-20-100	SF	SF	TL	TL
uflquad-nopsc-20-150	TL	TL	TL	TL
uflquad-nopsc-30-100	SF	SF	TL	TL
uflquad-nopsc-30-150	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL
uflquad-psc-10-100	1.63	7	3.96	7
uflquad-psc-10-150	2.98	7	9.88	5
uflquad-psc-20-100	8.05	15	68.17	13
uflquad-psc-20-150	10.4	11	46.82	11
uflquad-psc-30-100	17.28	21	68.65	21
uflquad-psc-30-150	1.55	1	1.63	1
uflquad-psc-30-200	15.73	7	105.65	7
uflquad-psc-30-300	53.31	17	262.13	15

Table A.7: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 1

	disco-oa		disco-oa-2		disco-oa-3		disco-oa-4	
	exper 0	exper 2	exper 3	exper 4				
problem	CPU t	node	CPU t	node	CPU t	node	CPU t	node
classical_200_1	TL	TL	TL	TL	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL	TL	TL	TL	TL
classical_200_3	TL	TL	TL	TL	TL	TL	TL	TL
classical_50_1	6778.7	288767	TL	TL	TL	TL	TL	TL
classical_50_2	1622.01	25469	1705.56	24605	1670.31	24319	TL	TL
classical_50_3	TL	TL	TL	TL	TL	TL	TL	TL
estein4_A	0.07	31	0.06	31	0.11	31	0.07	31
estein4_B	0.05	31	0.08	31	0.09	31	0.05	31
estein4_C	0.06	31	0.08	31	0.08	31	0.06	31
estein4_nr22	0.07	31	0.1	31	0.09	31	0.07	31
estein5_A	3.97	801	3.86	785	3.51	785	3.6	801
estein5_B	2.29	457	2.41	499	2.24	443	2.29	513
estein5_C	2.7	485	2.62	481	2.83	635	2.33	477
estein5_nr1	3.34	617	2.95	661	3.07	669	3.21	655
estein5_nr21	4.6	803	3.71	791	3.76	785	4.03	799
estein6_0	179.76	29307	199.16	27945	163.66	28557	167.79	30373
estein6_1	89.24	16101	106.97	18253	89.3	15821	87.86	17941
estein6_2	90.9	16621	95.13	16173	103.76	19093	75.2	14321
estein7_0	ML	ML	TL	TL	TL	TL	TL	TL
estein7_1	3867.0	481167	3982.39	389783	3415.1	347427	2954.29	427631
estein7_2	4753.69	701587	TL	TL	TL	TL	4981.7	784365
pp-n10-d10	0.67	307	0.51	241	0.6	223	0.62	305
pp-n10-d10000	2.01	2047	1.96	2047	1.97	2047	1.98	2047
pp-n100-d10	ML	ML	ML	ML	ML	ML	ML	ML
pp-n100-d10000	ML	ML	ML	ML	ML	ML	TL	TL
pp-n1000-d10	TL	TL	TL	TL	TL	TL	TL	TL
pp-n1000-d10000	ML	ML	TL	TL	TL	TL	TL	TL
pp-n100000-d10	TL	TL	TL	TL	TL	TL	TL	TL
pp-n100000-d10000	TL	TL	SF	SF	SF	SF	TL	TL
robust_100_1	TL	TL	TL	TL	TL	TL	TL	TL
robust_100_2	1471.85	7391	1914.4	8097	2927.49	15985	6169.13	38287
robust_100_3	TL	TL	3187.08	8603	TL	TL	TL	TL
robust_200_1	TL	TL	TL	TL	TL	TL	TL	TL
robust_200_2	TL	TL	TL	TL	TL	TL	TL	TL
robust_200_3	TL	TL	TL	TL	TL	TL	TL	TL
robust_50_1	50.05	1929	14.8	487	158.1	4893	50.6	1929
robust_50_2	21.1	705	68.43	2401	TL	TL	825.31	19335
robust_50_3	415.24	16251	111.92	3427	14.24	267	415.69	15739

Table A.8: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 2

	disco-oa		disco-oa-2		disco-oa-3		disco-oa-4	
problem	CPU time	node	CPU time	node	CPU time	node	CPU time	node
r12c15k5i10	0.01	51	0.01	51	0.01	51	0.02	51
r12c15k5i15	0.0	1	0.0	1	0.0	1	0.0	1
r14c18k3i12	SF	SF	0.14	375	0.16	379	SF	SF
r14c18k3i15	SF	SF	0.1	237	0.16	339	SF	SF
r14c18k3i18	0.06	155	0.04	121	0.05	151	0.06	155
r14c18k3i9	1.02	2969	SF	SF	1.17	2999	0.98	2969
r17c20k5i15	0.0	25	0.0	25	0.01	25	0.01	25
r17c20k5i20	0.01	27	0.01	27	SF	SF	0.01	27
r17c30k3i12	0.24	191	0.08	71	0.18	75	0.22	191
r17c30k3i15	1967.06	1605185	70.14	44933	3.55	1653	2063.04	1605185
r17c30k3i18	ML	ML	ML	ML	635.2	343397	ML	ML
r17c30k3i21	ML	ML	ML	ML	ML	ML	ML	ML
r17c30k3i24	ML	ML	ML	ML	ML	ML	ML	ML
r17c30k3i27	ML	ML	1045.97	892249	860.47	554505	ML	ML
r22c30k10i20	0.94	3407	0.93	3239	0.97	3367	0.94	3407
r22c40k10i20	0.61	595	0.3	229	0.04	21	0.63	595
r22c40k10i30	1.43	2143	1.34	1585	1.54	1441	1.53	2143
r22c40k10i40	6.38	13765	7.82	16393	4.92	10219	6.31	13765
r23c45k3i21	84.99	43457	4103.34	1470823	54.67	14457	88.25	43457
r23c45k3i24	ML	ML	ML	ML	560.62	143871	ML	ML
r23c45k3i27	ML	ML	ML	ML	TL	TL	ML	ML
r23c45k3i30	ML	ML	ML	ML	ML	ML	ML	ML
r23c45k3i33	ML	ML	ML	ML	ML	ML	ML	ML
r23c45k3i36	ML	ML	ML	ML	ML	ML	ML	ML
r27c50k5i25	ML	ML	560.05	255093	16.19	4663	ML	ML
r27c50k5i30	ML	ML	ML	ML	ML	ML	ML	ML
r27c50k5i35	ML	ML	ML	ML	ML	ML	ML	ML
r27c50k5i40	ML	ML	TL	TL	ML	ML	TL	TL
r27c50k5i45	ML	ML	ML	ML	ML	ML	ML	ML
r27c50k5i50	ML	ML	ML	ML	ML	ML	TL	TL
r32c45k15i30	0.04	105	0.05	105	0.04	105	0.06	105
r32c45k15i45	0.15	269	0.48	905	0.4	733	0.14	269
r32c60k15i30	0.32	233	0.37	201	0.44	153	0.32	233
r32c60k15i45	413.76	490851	386.19	450013	420.7	350665	423.81	490851
r32c60k15i60	ML	ML	ML	ML	ML	ML	SF	SF
r52c75k5i35	TL	TL	TL	TL	TL	TL	TL	TL
r52c75k5i40	ML	ML	TL	TL	TL	TL	TL	TL
r52c75k5i45	SF	SF	TL	TL	TL	TL	TL	TL
r52c75k5i50	ML	ML	TL	TL	TL	TL	ML	ML
r52c75k5i60	ML	ML	TL	TL	ML	ML	ML	ML
r52c75k5i65	TL	TL	TL	TL	TL	TL	TL	TL

Table A.9: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 3

	disco-oa		disco-oa-2		disco-oa-3		disco-oa-4	
problem	CPU time	node	CPU time	node	CPU time	node	CPU time	node
shortfall_100_1	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_100_2	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_100_3	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_200_1	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL	TL	TL	TL	TL
shortfall_50_1	356.38	2653	331.7	2277	332.57	2519	1457.88	27015
shortfall_50_2	547.0	5651	526.12	4143	415.21	3091	1286.0	24745
shortfall_50_3	5658.02	28637	6204.34	33979	5109.38	25475	TL	TL
sssd-strong-15-4	ML	ML	ML	ML	ML	ML	ML	ML
sssd-strong-15-8	TL	TL	TL	TL	TL	TL	TL	TL
sssd-strong-20-4	ML	ML	TL	TL	ML	ML	ML	ML
sssd-strong-20-8	TL	TL	TL	TL	TL	TL	TL	TL
sssd-strong-25-4	ML	ML	ML	ML	ML	ML	TL	TL
sssd-strong-25-8	TL	TL	TL	TL	TL	TL	TL	TL
sssd-strong-30-4	ML	ML	TL	TL	ML	ML	ML	ML
sssd-strong-30-8	TL	TL	TL	TL	TL	TL	TL	TL
sssd-weak-15-4	ML	ML	ML	ML	TL	TL	ML	ML
sssd-weak-15-8	ML	ML	ML	ML	TL	TL	ML	ML
sssd-weak-20-4	ML	ML	ML	ML	ML	ML	ML	ML
sssd-weak-20-8	ML	ML	TL	TL	ML	ML	TL	TL
sssd-weak-25-4	ML	ML	ML	ML	ML	ML	ML	ML
sssd-weak-25-8	ML	ML	ML	ML	TL	TL	ML	ML
sssd-weak-30-4	ML	ML	ML	ML	ML	ML	ML	ML
sssd-weak-30-8	ML	ML	TL	TL	TL	TL	ML	ML
turbine07GF	0.02	17	0.02	17	0.03	17	0.02	17
turbine07_aniso	0.0	1	0.0	1	0.0	1	0.0	1
turbine07_lowb	1.5	329	1.74	295	1.87	211	1.48	329
turbine07_lowb_aniso	2.24	657	1.92	433	2.74	433	2.26	657
turbine54GF	0.02	5	0.02	5	0.03	5	0.02	5
uflquad-nopsc-10-100	189.49	251	200.79	263	176.7	257	193.36	251
uflquad-nopsc-10-150	251.69	407	287.7	355	278.96	255	262.94	407
uflquad-nopsc-20-100	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-nopsc-20-150	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-100	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-150	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL	TL	TL	TL	TL
uflquad-psc-10-100	4.05	9	4.56	9	6.11	9	3.98	9
uflquad-psc-10-150	9.41	9	9.63	7	10.76	7	9.62	9
uflquad-psc-20-100	37.32	25	46.47	25	45.97	23	37.34	25
uflquad-psc-20-150	30.13	11	36.4	11	47.01	11	32.87	11
uflquad-psc-30-100	63.96	29	91.48	33	157.56	39	66.18	29
uflquad-psc-30-150	43.15	3	49.39	3	54.24	3	47.58	3
uflquad-psc-30-200	156.0	7	206.59	7	165.39	7	166.36	7
uflquad-psc-30-300	853.88	29	1089.7	29	1335.16	29	813.86	29

Table A.10: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 4

	disco-oa-5		disco-oa-6		disco-oa-7	
problem	CPU time	node	CPU time	node	CPU time	node
classical_200_1	TL	TL	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL	TL	TL
classical_200_3	TL	TL	TL	TL	TL	TL
classical_50_1	5849.27	232547	TL	TL	6587.66	288879
classical_50_2	897.27	6927	2975.46	42449	1624.42	25469
classical_50_3	3641.37	35863	TL	TL	TL	TL
estein4_A	0.07	31	0.07	31	0.07	31
estein4_B	0.05	31	0.04	31	0.04	31
estein4_C	0.06	31	0.07	31	0.06	31
estein4_nr22	0.07	31	0.07	31	0.07	31
estein5_A	4.41	801	4.06	801	3.98	801
estein5_B	2.66	471	2.34	457	2.33	457
estein5_C	3.01	495	2.74	485	2.71	485
estein5_nr1	4.12	677	3.33	617	3.28	617
estein5_nr21	4.92	801	4.63	803	4.69	803
estein6_0	198.09	28639	181.55	29287	180.48	29307
estein6_1	105.11	17579	91.75	16091	90.8	16101
estein6_2	99.39	15665	93.77	16621	95.71	16621
estein7_0	ML	ML	TL	TL	ML	ML
estein7_1	TL	TL	3224.52	404169	3933.17	481167
estein7_2	TL	TL	3958.62	607279	4801.18	701587
pp-n10-d10	0.71	307	0.69	307	0.66	307
pp-n10-d10000	2.03	2047	2.03	2047	2.01	2047
pp-n100-d10	ML	ML	ML	ML	ML	ML
pp-n100-d10000	ML	ML	ML	ML	ML	ML
pp-n1000-d10	TL	TL	TL	TL	TL	TL
pp-n1000-d10000	TL	TL	ML	ML	TL	TL
pp-n100000-d10	TL	TL	TL	TL	TL	TL
pp-n100000-d10000	TL	TL	SF	SF	SF	SF
robust_100_1	TL	TL	TL	TL	TL	TL
robust_100_2	1485.09	5613	2969.23	12519	1513.98	7381
robust_100_3	TL	TL	TL	TL	TL	TL
robust_200_1	TL	TL	TL	TL	TL	TL
robust_200_2	TL	TL	TL	TL	TL	TL
robust_200_3	TL	TL	TL	TL	TL	TL
robust_50_1	16.7	447	51.41	1929	49.12	1929
robust_50_2	19.92	673	21.86	711	20.66	705
robust_50_3	263.8	9469	423.59	16251	419.18	16251

Table A.11: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 5

	disco-oa-5		disco-oa-6		disco-oa-7	
problem	CPU time	node	CPU time	node	CPU time	node
r12c15k5i10	0.01	51	0.01	51	0.02	51
r12c15k5i15	0.0	1	0.0	1	0.0	1
r14c18k3i12	SF	SF	SF	SF	SF	SF
r14c18k3i15	SF	SF	SF	SF	SF	SF
r14c18k3i18	0.05	155	0.05	155	0.06	155
r14c18k3i9	0.99	2969	0.98	2969	0.94	2969
r17c20k5i15	0.01	25	0.01	25	0.01	25
r17c20k5i20	0.02	27	0.01	27	0.01	27
r17c30k3i12	0.22	191	0.23	191	0.22	191
r17c30k3i15	1994.86	1605185	2016.69	1605187	1981.56	1605185
r17c30k3i18	ML	ML	ML	ML	ML	ML
r17c30k3i21	ML	ML	SF	SF	ML	ML
r17c30k3i24	ML	ML	ML	ML	ML	ML
r17c30k3i27	ML	ML	ML	ML	ML	ML
r22c30k10i20	0.94	3407	0.96	3407	0.97	3407
r22c40k10i20	0.62	595	0.59	595	0.61	595
r22c40k10i30	1.43	2143	1.44	2143	1.42	2143
r22c40k10i40	6.33	13765	6.34	13765	6.22	13765
r23c45k3i21	85.77	43457	83.98	43457	84.44	43457
r23c45k3i24	ML	ML	ML	ML	ML	ML
r23c45k3i27	ML	ML	ML	ML	ML	ML
r23c45k3i30	TL	TL	TL	TL	ML	ML
r23c45k3i33	ML	ML	ML	ML	ML	ML
r23c45k3i36	ML	ML	ML	ML	ML	ML
r27c50k5i25	ML	ML	ML	ML	ML	ML
r27c50k5i30	ML	ML	ML	ML	ML	ML
r27c50k5i35	ML	ML	ML	ML	ML	ML
r27c50k5i40	ML	ML	ML	ML	ML	ML
r27c50k5i45	ML	ML	ML	ML	ML	ML
r27c50k5i50	ML	ML	TL	TL	ML	ML
r32c45k15i30	0.04	105	0.05	105	0.05	105
r32c45k15i45	0.16	269	0.17	269	0.14	269
r32c60k15i30	0.32	233	0.32	233	0.3	233
r32c60k15i45	423.8	490851	412.4	490851	423.12	490851
r32c60k15i60	ML	ML	SF	SF	ML	ML
r52c75k5i35	TL	TL	TL	TL	TL	TL
r52c75k5i40	ML	ML	TL	TL	ML	ML
r52c75k5i45	TL	TL	TL	TL	TL	TL
r52c75k5i50	TL	TL	ML	ML	ML	ML
r52c75k5i60	ML	ML	TL	TL	ML	ML
r52c75k5i65	TL	TL	TL	TL	TL	TL

Table A.12: bb-lp CPU Time and Number of Nodes with Different bb-lp OA Cut Parameter Values, Part 6

	disco-oa-5		disco-oa-6		disco-oa-7	
problem	CPU time	node	CPU time	node	CPU time	node
shortfall_100_1	TL	TL	TL	TL	TL	TL
shortfall_100_2	TL	TL	TL	TL	TL	TL
shortfall_100_3	TL	TL	TL	TL	TL	TL
shortfall_200_1	TL	TL	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL	TL	TL
shortfall_50_1	321.76	1633	602.64	6361	265.46	2109
shortfall_50_2	821.7	6485	1866.77	24425	563.48	5167
shortfall_50_3	TL	TL	TL	TL	5056.01	20899
sssd-strong-15-4	ML	ML	ML	ML	ML	ML
sssd-strong-15-8	TL	TL	TL	TL	TL	TL
sssd-strong-20-4	TL	TL	ML	ML	ML	ML
sssd-strong-20-8	TL	TL	TL	TL	TL	TL
sssd-strong-25-4	ML	ML	ML	ML	ML	ML
sssd-strong-25-8	TL	TL	TL	TL	TL	TL
sssd-strong-30-4	TL	TL	TL	TL	ML	ML
sssd-strong-30-8	TL	TL	TL	TL	TL	TL
sssd-weak-15-4	ML	ML	ML	ML	ML	ML
sssd-weak-15-8	ML	ML	TL	TL	ML	ML
sssd-weak-20-4	ML	ML	ML	ML	ML	ML
sssd-weak-20-8	TL	TL	ML	ML	TL	TL
sssd-weak-25-4	ML	ML	ML	ML	ML	ML
sssd-weak-25-8	ML	ML	TL	TL	TL	TL
sssd-weak-30-4	TL	TL	ML	ML	ML	ML
sssd-weak-30-8	ML	ML	TL	TL	ML	ML
turbine07GF	0.02	17	0.02	17	0.02	17
turbine07_aniso	0.0	1	0.0	1	0.0	1
turbine07_lowb	1.54	329	1.52	329	1.48	329
turbine07_lowb_aniso	2.29	657	2.2	657	2.25	657
turbine54GF	0.02	5	0.02	5	0.03	5
uflquad-nopsc-10-100	191.64	251	189.74	251	200.86	251
uflquad-nopsc-10-150	255.22	407	248.96	407	244.4	407
uflquad-nopsc-20-100	TL	TL	TL	TL	TL	TL
uflquad-nopsc-20-150	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-100	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-150	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL	TL	TL
uflquad-psc-10-100	4.16	9	4.03	9	3.93	9
uflquad-psc-10-150	8.94	9	8.89	9	8.99	9
uflquad-psc-20-100	37.82	25	37.95	25	36.67	25
uflquad-psc-20-150	30.25	11	31.59	11	32.47	11
uflquad-psc-30-100	65.12	29	63.97	29	62.16	29
uflquad-psc-30-150	51.68	3	48.75	3	53.84	3
uflquad-psc-30-200	165.91	7	142.2	7	161.36	7
uflquad-psc-30-300	967.69	29	985.68	29	918.44	29

Table A.13: bb-lp CPU Time and Number of Nodes with Different Branching Strategies Part 1

	disco-oa		disco-oa-strong	
problem	CPU time	node	CPU time	node
classical_200_1	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL
classical_200_3	TL	TL	TL	TL
classical_50_1	6778.7	288767	TL	TL
classical_50_2	1622.01	25469	3832.99	63363
classical_50_3	TL	TL	TL	TL
estein4_A	0.07	31	0.11	31
estein4_B	0.05	31	0.08	31
estein4_C	0.06	31	0.09	31
estein4_nr22	0.07	31	0.08	21
estein5_A	3.97	801	5.48	799
estein5_B	2.29	457	2.96	457
estein5_C	2.7	485	2.86	445
estein5_nr1	3.34	617	3.81	531
estein5_nr21	4.6	803	5.84	783
estein6_0	179.76	29307	193.45	23471
estein6_1	89.24	16101	113.21	13017
estein6_2	90.9	16621	105.1	13521
estein7_0	ML	ML	TL	TL
estein7_1	3867.0	481167	3644.47	281201
estein7_2	4753.69	701587	TL	TL
pp-n10-d10	0.67	307	2.21	1359
pp-n10-d10000	2.01	2047	2.36	1043
pp-n100-d10	ML	ML	TL	TL
pp-n100-d10000	ML	ML	TL	TL
pp-n1000-d10	TL	TL	TL	TL
pp-n1000-d10000	ML	ML	TL	TL
pp-n100000-d10	TL	TL	TL	TL
pp-n100000-d10000	TL	TL	TL	TL
robust_100_1	TL	TL	TL	TL
robust_100_2	1471.85	7391	3218.65	12299
robust_100_3	TL	TL	TL	TL
robust_200_1	TL	TL	TL	TL
robust_200_2	TL	TL	TL	TL
robust_200_3	TL	TL	TL	TL
robust_50_1	50.05	1929	17.0	175
robust_50_2	21.1	705	22.08	351
robust_50_3	415.24	16251	486.88	11549

Table A.14: bb-lp CPU Time and Number of Nodes with Different Branching Strategies Part 2

	disco-oa		disco-oa-strong	
problem	CPU time	node	CPU time	node
r12c15k5i10	0.01	51	0.02	51
r12c15k5i15	0.0	1	0.0	1
r14c18k3i12	SF	SF	SF	SF
r14c18k3i15	SF	SF	SF	SF
r14c18k3i18	0.06	155	0.09	73
r14c18k3i9	1.02	2969	1.28	3271
r17c20k5i15	0.0	25	0.02	17
r17c20k5i20	0.01	27	0.04	23
r17c30k3i12	0.24	191	0.74	399
r17c30k3i15	1967.06	1605185	1928.39	1211199
r17c30k3i18	ML	ML	TL	TL
r17c30k3i21	ML	ML	414.13	233757
r17c30k3i24	ML	ML	382.74	142451
r17c30k3i27	ML	ML	130.1	46233
r22c30k10i20	0.94	3407	1.48	3275
r22c40k10i20	0.61	595	0.46	351
r22c40k10i30	1.43	2143	3.32	2681
r22c40k10i40	6.38	13765	15.7	11737
r23c45k3i21	84.99	43457	114.31	29139
r23c45k3i24	ML	ML	ML	ML
r23c45k3i27	ML	ML	TL	TL
r23c45k3i30	ML	ML	TL	TL
r23c45k3i33	ML	ML	SF	SF
r23c45k3i36	ML	ML	TL	TL
r27c50k5i25	ML	ML	ML	ML
r27c50k5i30	ML	ML	TL	TL
r27c50k5i35	ML	ML	TL	TL
r27c50k5i40	ML	ML	TL	TL
r27c50k5i45	ML	ML	TL	TL
r27c50k5i50	ML	ML	TL	TL
r32c45k15i30	0.04	105	0.01	11
r32c45k15i45	0.15	269	0.66	631
r32c60k15i30	0.32	233	0.34	137
r32c60k15i45	413.76	490851	SF	SF
r32c60k15i60	ML	ML	ML	ML
r52c75k5i35	TL	TL	TL	TL
r52c75k5i40	ML	ML	TL	TL
r52c75k5i45	SF	SF	TL	TL
r52c75k5i50	ML	ML	TL	TL
r52c75k5i60	ML	ML	565.23	55621
r52c75k5i65	TL	TL	552.44	41281

Table A.15: bb-lp CPU Time and Number of Nodes with Different Branching Strategies Part 3

	disco-oa		disco-oa-strong	
problem	CPU time	node	CPU time	node
shortfall_100_1	TL	TL	TL	TL
shortfall_100_2	TL	TL	TL	TL
shortfall_100_3	TL	TL	TL	TL
shortfall_200_1	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL
shortfall_50_1	356.38	2653	258.92	2367
shortfall_50_2	547.0	5651	1041.34	12335
shortfall_50_3	5658.02	28637	TL	TL
sssd-strong-15-4	ML	ML	1572.73	205523
sssd-strong-15-8	TL	TL	TL	TL
sssd-strong-20-4	ML	ML	1382.6	186161
sssd-strong-20-8	TL	TL	TL	TL
sssd-strong-25-4	ML	ML	TL	TL
sssd-strong-25-8	TL	TL	TL	TL
sssd-strong-30-4	ML	ML	TL	TL
sssd-strong-30-8	TL	TL	TL	TL
sssd-weak-15-4	ML	ML	SF	SF
sssd-weak-15-8	ML	ML	TL	TL
sssd-weak-20-4	ML	ML	ML	ML
sssd-weak-20-8	ML	ML	TL	TL
sssd-weak-25-4	ML	ML	TL	TL
sssd-weak-25-8	ML	ML	TL	TL
sssd-weak-30-4	ML	ML	TL	TL
sssd-weak-30-8	ML	ML	TL	TL
turbine07GF	0.02	17	0.03	21
turbine07_aniso	0.0	1	0.0	1
turbine07_lowb	1.5	329	0.71	59
turbine07_lowb_aniso	2.24	657	1.25	121
turbine54GF	0.02	5	0.03	5
uflquad-nopsc-10-100	189.49	251	288.08	295
uflquad-nopsc-10-150	251.69	407	535.94	475
uflquad-nopsc-20-100	TL	TL	TL	TL
uflquad-nopsc-20-150	TL	TL	TL	TL
uflquad-nopsc-30-100	TL	TL	TL	TL
uflquad-nopsc-30-150	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL
uflquad-psc-10-100	4.05	9	3.68	7
uflquad-psc-10-150	9.41	9	9.53	7
uflquad-psc-20-100	37.32	25	23.1	15
uflquad-psc-20-150	30.13	11	36.41	13
uflquad-psc-30-100	63.96	29	56.92	23
uflquad-psc-30-150	43.15	3	48.45	3
uflquad-psc-30-200	156.0	7	157.62	7
uflquad-psc-30-300	853.88	29	938.56	27

Table A.16: bb-lp CPU Time and Number of Nodes with Different MILP Cutting Strategies Part 1

	disco-oa		disco-oanomilpcuts	
problem	CPU time	node	CPU time	node
classical_200_1	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL
classical_200_3	TL	TL	TL	TL
classical_50_1	6778.7	288767	TL	TL
classical_50_2	1622.01	25469	1610.43	25509
classical_50_3	TL	TL	TL	TL
estein4_A	0.07	31	0.07	31
estein4_B	0.05	31	0.04	31
estein4_C	0.06	31	0.05	31
estein4_nr22	0.07	31	0.06	31
estein5_A	3.97	801	4.06	843
estein5_B	2.29	457	2.84	511
estein5_C	2.7	485	2.66	521
estein5_nr1	3.34	617	3.45	641
estein5_nr21	4.6	803	5.28	841
estein6_0	179.76	29307	171.36	27981
estein6_1	89.24	16101	90.81	16189
estein6_2	90.9	16621	81.32	14875
estein7_0	ML	ML	ML	ML
estein7_1	3867.0	481167	4150.41	512219
estein7_2	4753.69	701587	5489.78	817289
pp-n10-d10	0.67	307	0.53	227
pp-n10-d10000	2.01	2047	2.0	2047
pp-n100-d10	ML	ML	ML	ML
pp-n100-d10000	ML	ML	ML	ML
pp-n1000-d10	TL	TL	TL	TL
pp-n1000-d10000	ML	ML	ML	ML
pp-n100000-d10	TL	TL	TL	TL
pp-n100000-d10000	TL	TL	TL	TL
robust_100_1	TL	TL	TL	TL
robust_100_2	1471.85	7391	1511.56	7391
robust_100_3	TL	TL	TL	TL
robust_200_1	TL	TL	TL	TL
robust_200_2	TL	TL	TL	TL
robust_200_3	TL	TL	TL	TL
robust_50_1	50.05	1929	49.55	1929
robust_50_2	21.1	705	20.8	705
robust_50_3	415.24	16251	410.72	16251

Table A.17: bb-lp CPU Time and Number of Nodes with Different MILP Cutting Strategies Part 2

	disco-oa		disco-oanomilpcuts	
problem	CPU time	node	CPU time	node
r12c15k5i10	0.01	51	0.01	69
r12c15k5i15	0.0	1	0.01	29
r14c18k3i12	SF	SF	0.12	333
r14c18k3i15	SF	SF	0.07	191
r14c18k3i18	0.06	155	0.05	149
r14c18k3i9	1.02	2969	0.97	3149
r17c20k5i15	0.0	25	0.01	33
r17c20k5i20	0.01	27	0.01	33
r17c30k3i12	0.24	191	0.25	239
r17c30k3i15	1967.06	1605185	640.05	584635
r17c30k3i18	ML	ML	ML	ML
r17c30k3i21	ML	ML	ML	ML
r17c30k3i24	ML	ML	ML	ML
r17c30k3i27	ML	ML	ML	ML
r22c30k10i20	0.94	3407	0.74	3309
r22c40k10i20	0.61	595	0.67	645
r22c40k10i30	1.43	2143	1.17	1789
r22c40k10i40	6.38	13765	9.17	21595
r23c45k3i21	84.99	43457	43.65	24371
r23c45k3i24	ML	ML	ML	ML
r23c45k3i27	ML	ML	ML	ML
r23c45k3i30	ML	ML	ML	ML
r23c45k3i33	ML	ML	ML	ML
r23c45k3i36	ML	ML	ML	ML
r27c50k5i25	ML	ML	ML	ML
r27c50k5i30	ML	ML	ML	ML
r27c50k5i35	ML	ML	ML	ML
r27c50k5i40	ML	ML	ML	ML
r27c50k5i45	ML	ML	ML	ML
r27c50k5i50	ML	ML	TL	TL
r32c45k15i30	0.04	105	0.06	173
r32c45k15i45	0.15	269	0.22	575
r32c60k15i30	0.32	233	0.34	269
r32c60k15i45	413.76	490851	711.14	781131
r32c60k15i60	ML	ML	ML	ML
r52c75k5i35	TL	TL	TL	TL
r52c75k5i40	ML	ML	TL	TL
r52c75k5i45	SF	SF	TL	TL
r52c75k5i50	ML	ML	ML	ML
r52c75k5i60	ML	ML	ML	ML
r52c75k5i65	TL	TL	TL	TL

Table A.18: bb-lp CPU Time and Number of Nodes with Different MILP Cutting Strategies Part 3

	disco-oa		disco-oanomilpcuts	
problem	CPU time	node	CPU time	node
shortfall_100_1	TL	TL	TL	TL
shortfall_100_2	TL	TL	TL	TL
shortfall_100_3	TL	TL	TL	TL
shortfall_200_1	TL	TL	TL	TL
shortfall_200_2	TL	TL	TL	TL
shortfall_200_3	TL	TL	TL	TL
shortfall_50_1	356.38	2653	360.52	2655
shortfall_50_2	547.0	5651	532.54	5649
shortfall_50_3	5658.02	28637	5694.26	29085
sssd-strong-15-4	ML	ML	ML	ML
sssd-strong-15-8	TL	TL	TL	TL
sssd-strong-20-4	ML	ML	ML	ML
sssd-strong-20-8	TL	TL	TL	TL
sssd-strong-25-4	ML	ML	ML	ML
sssd-strong-25-8	TL	TL	TL	TL
sssd-strong-30-4	ML	ML	ML	ML
sssd-strong-30-8	TL	TL	TL	TL
sssd-weak-15-4	ML	ML	ML	ML
sssd-weak-15-8	ML	ML	ML	ML
sssd-weak-20-4	ML	ML	ML	ML
sssd-weak-20-8	ML	ML	ML	ML
sssd-weak-25-4	ML	ML	ML	ML
sssd-weak-25-8	ML	ML	TL	TL
sssd-weak-30-4	ML	ML	ML	ML
sssd-weak-30-8	ML	ML	ML	ML
turbine07GF	0.02	17	0.02	17
turbine07_aniso	0.0	1	0.0	1
turbine07_lowb	1.5	329	1.5	329
turbine07_lowb_aniso	2.24	657	1.91	597
turbine54GF	0.02	5	0.02	5
uflquad-nopsc-10-100	189.49	251	190.8	251
uflquad-nopsc-10-150	251.69	407	250.91	407
uflquad-nopsc-20-100	TL	TL	TL	TL
uflquad-nopsc-20-150	TL	TL	TL	TL
uflquad-nopsc-30-100	TL	TL	TL	TL
uflquad-nopsc-30-150	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL
uflquad-psc-10-100	4.05	9	3.88	9
uflquad-psc-10-150	9.41	9	8.86	9
uflquad-psc-20-100	37.32	25	37.36	25
uflquad-psc-20-150	30.13	11	29.51	11
uflquad-psc-30-100	63.96	29	64.45	29
uflquad-psc-30-150	43.15	3	43.82	3
uflquad-psc-30-200	156.0	7	167.04	7
uflquad-psc-30-300	853.88	29	798.82	29

Table A.19: bb-socp with Disjunctive Conic Cuts

	disco-cplex		disco-cplex-dc-all		disco-cplex-dc-best	
problem	CPU t	node	CPU t	node	CPU t	node
r12c15k5i10	0.03	63	0.04	49	0.04	49
r12c15k5i15	0.02	29	0.0	5	0.0	5
r14c18k3i12	0.45	501	0.65	387	0.65	387
r14c18k3i15	0.19	183	0.58	303	0.57	303
r14c18k3i18	0.16	149	0.37	157	0.34	157
r14c18k3i9	2.22	3133	4.0	2981	2.9	2935
r17c20k5i15	0.05	33	0.06	39	0.05	39
r17c20k5i20	0.03	33	0.04	23	0.04	23
r17c30k3i12	0.1	79	0.12	29	0.12	29
r17c30k3i15	1.6	1145	3.2	591	2.48	797
r17c30k3i18	SF	SF	TL	TL	TL	TL
r17c30k3i21	SF	SF	SF	SF	SF	SF
r17c30k3i24	SF	SF	SF	SF	SF	SF
r17c30k3i27	SF	SF	SF	SF	SF	SF
r22c30k10i20	2.31	3295	3.75	3379	3.52	3379
r22c40k10i20	0.03	23	0.03	9	0.03	9
r22c40k10i30	1.8	1333	3.64	1053	3.1	1285
r22c40k10i40	20.14	16037	24.94	8027	25.04	8027
r23c45k3i21	1.08	529	5.13	485	5.14	485
r23c45k3i24	2.13	1003	8.25	715	8.59	715
r23c45k3i27	SF	SF	SF	SF	SF	SF
r23c45k3i30	SF	SF	SF	SF	SF	SF
r23c45k3i33	SF	SF	SF	SF	SF	SF
r23c45k3i36	SF	SF	SF	SF	SF	SF
r27c50k5i25	1.48	691	4.54	601	4.45	601
r27c50k5i30	TL	TL	TL	TL	SF	SF
r27c50k5i35	SF	SF	SF	SF	SF	SF
r27c50k5i40	SF	SF	TL	TL	TL	TL
r27c50k5i45	SF	SF	SF	SF	SF	SF
r27c50k5i50	SF	SF	SF	SF	TL	TL
r32c45k15i30	0.24	173	0.02	7	0.36	167
r32c45k15i45	0.67	635	1.04	497	1.02	497
r32c60k15i30	0.24	127	0.14	31	0.46	121
r32c60k15i45	1186.22	576477	2076.15	389185	2299.53	389185
r32c60k15i60	SF	SF	SF	SF	SF	SF
r52c75k5i35	SF	SF	SF	SF	SF	SF
r52c75k5i40	SF	SF	SF	SF	TL	TL
r52c75k5i45	SF	SF	SF	SF	SF	SF
r52c75k5i50	SF	SF	TL	TL	TL	TL
r52c75k5i60	SF	SF	SF	SF	SF	SF
r52c75k5i65	SF	SF	SF	SF	SF	SF

Table A.20: bb-lp with Disjunctive Conic Cuts

	disco-oa-dc-all		disco-oa-dc-best		disco-oanomilpcuts	
problem	CPU t	node	CPU t	node	CPU t	node
r12c15k5i10	0.01	63	0.01	63	0.01	69
r12c15k5i15	0.0	25	0.0	25	0.01	29
r14c18k3i12	0.11	297	0.09	295	0.12	333
r14c18k3i15	0.09	287	0.07	287	0.07	191
r14c18k3i18	0.03	153	0.03	153	0.05	149
r14c18k3i9	0.98	2931	0.85	3029	0.97	3149
r17c20k5i15	0.02	41	0.01	41	0.01	33
r17c20k5i20	0.01	25	0.01	41	0.01	33
r17c30k3i12	TL	TL	SF	SF	0.25	239
r17c30k3i15	TL	TL	SF	SF	640.05	584635
r17c30k3i18	TL	TL	ML	ML	ML	ML
r17c30k3i21	ML	ML	ML	ML	ML	ML
r17c30k3i24	ML	ML	ML	ML	ML	ML
r17c30k3i27	ML	ML	ML	ML	ML	ML
r22c30k10i20	0.77	3387	SF	SF	0.74	3309
r22c40k10i20	0.16	71	0.02	21	0.67	645
r22c40k10i30	0.72	1043	0.73	1249	1.17	1789
r22c40k10i40	4.0	6301	7.9	17897	9.17	21595
r23c45k3i21	SF	SF	SF	SF	43.65	24371
r23c45k3i24	TL	TL	TL	TL	ML	ML
r23c45k3i27	TL	TL	ML	ML	ML	ML
r23c45k3i30	SF	SF	SF	SF	ML	ML
r23c45k3i33	ML	ML	SF	SF	ML	ML
r23c45k3i36	ML	ML	ML	ML	ML	ML
r27c50k5i25	SF	SF	SF	SF	ML	ML
r27c50k5i30	SF	SF	ML	ML	ML	ML
r27c50k5i35	TL	TL	ML	ML	ML	ML
r27c50k5i40	ML	ML	ML	ML	ML	ML
r27c50k5i45	ML	ML	ML	ML	ML	ML
r27c50k5i50	ML	ML	ML	ML	TL	TL
r32c45k15i30	0.0	7	0.05	171	0.06	173
r32c45k15i45	0.17	497	0.22	573	0.22	575
r32c60k15i30	0.99	529	0.3	79	0.34	269
r32c60k15i45	378.78	326433	869.92	964223	711.14	781131
r32c60k15i60	SF	SF	SF	SF	ML	ML
r52c75k5i35	SF	SF	SF	SF	TL	TL
r52c75k5i40	TL	TL	TL	TL	TL	TL
r52c75k5i45	TL	TL	TL	TL	TL	TL
r52c75k5i50	ML	ML	ML	ML	ML	ML
r52c75k5i60	TL	TL	ML	ML	ML	ML
r52c75k5i65	TL	TL	TL	TL	TL	TL

Table A.21: Parallel bb-socp with various number of processors Part 1

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
classical_200_1	TL	TL	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL	TL	TL
classical_200_3	SF	SF	TL	TL	TL	TL
classical_50_1	24.04	1713	8.36406	7369	5.9678	10597
classical_50_2	60.35	3945	17.5662	15397	11.8537	21171
classical_50_3	149.14	10259	30.8728	29699	17.562	34483
estein4_A	SF	SF	SF	SF	SF	SF
estein4_B	SF	SF	SF	SF	SF	SF
estein4_C	SF	SF	SF	SF	SF	SF
estein4_nr22	SF	SF	SF	SF	SF	SF
estein5_A	1.96	785	0.254984	793	0.323918	807
estein5_B	1.18	445	0.228411	465	0.140266	509
estein5_C	1.54	635	0.262343	669	0.258142	693
estein5_nr1	1.67	649	0.279194	653	0.258386	671
estein5_nr21	1.96	785	0.258395	785	0.236696	789
estein6_0	91.61	27789	7.74397	28279	4.04088	28997
estein6_1	SF	SF	SF	SF	SF	SF
estein6_2	66.18	19531	5.3745	19695	3.01614	20835
estein7_0	5313.37	990533	362.464	985099	172.722	979473
estein7_1	1824.74	340935	147.481	381689	68.8174	366165
estein7_2	3098.51	598215	192.929	514005	96.0458	537205
pp-n10-d10	0.63	295	0.312297	373	0.317596	585
pp-n10-d10000	1.99	2049	1.11679	13627	1.09452	27491
pp-n100-d10	TL	TL	TL	TL	TL	TL
pp-n100-d10000	TL	TL	TL	TL	TL	TL
pp-n1000-d10	TL	TL	TL	TL	TL	TL
pp-n1000-d10000	TL	TL	TL	TL	TL	TL
pp-n100000-d10	SF	SF	ML	ML	ML	ML
pp-n100000-d10000	SF	SF	ML	ML	ML	ML
robust_100_1	1168.5	6109	414.192	29829	232.021	34039
robust_100_2	398.04	2141	98.8693	6445	77.2885	10247
robust_100_3	167.43	801	48.6272	2879	42.0858	4717
robust_200_1	TL	TL	TL	TL	TL	TL
robust_200_2	6795.73	4883	1579.93	15389	812.302	14261
robust_200_3	TL	TL	TL	TL	TL	TL
robust_50_1	2.34	59	2.0758	411	1.76622	779
robust_50_2	2.64	67	1.9843	575	1.70583	875
robust_50_3	5.45	143	3.50122	953	2.61012	1363

Table A.22: Parallel bb-socp with various number of processors Part 2

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
r12c15k5i10	0.03	63	0.115183	87	0.121856	91
r12c15k5i15	0.02	29	0.045979	27	0.048228	27
r14c18k3i12	0.45	501	0.217421	583	0.150607	501
r14c18k3i15	0.19	183	0.239508	227	0.163703	265
r14c18k3i18	0.16	149	0.18724	159	SF	SF
r14c18k3i9	2.22	3133	SF	SF	SF	SF
r17c20k5i15	0.05	33	0.0727952	39	0.0605199	39
r17c20k5i20	0.03	33	0.078202	39	0.060199	39
r17c30k3i12	0.1	79	0.102213	349	0.0960951	631
r17c30k3i15	1.6	1145	0.248185	1551	0.248617	3173
r17c30k3i18	SF	SF	SF	SF	SF	SF
r17c30k3i21	SF	SF	SF	SF	SF	SF
r17c30k3i24	SF	SF	SF	SF	SF	SF
r17c30k3i27	SF	SF	SF	SF	SF	SF
r22c30k10i20	2.31	3295	1.63873	15267	1.18816	18645
r22c40k10i20	0.03	23	0.091264	99	0.099143	183
r22c40k10i30	1.8	1333	0.586128	4565	0.659862	7397
r22c40k10i40	20.14	16037	1.81618	14165	1.5489	23957
r23c45k3i21	1.08	529	0.366664	1559	0.342166	3375
r23c45k3i24	2.13	1003	0.704857	3355	0.478512	4473
r23c45k3i27	SF	SF	SF	SF	TL	TL
r23c45k3i30	SF	SF	SF	SF	SF	SF
r23c45k3i33	SF	SF	SF	SF	SF	SF
r23c45k3i36	SF	SF	SF	SF	SF	SF
r27c50k5i25	1.48	691	0.646688	1935	0.51002	3787
r27c50k5i30	TL	TL	TL	TL	TL	TL
r27c50k5i35	SF	SF	SF	SF	SF	SF
r27c50k5i40	SF	SF	SF	SF	SF	SF
r27c50k5i45	SF	SF	SF	SF	SF	SF
r27c50k5i50	SF	SF	SF	SF	SF	SF
r32c45k15i30	0.24	173	0.207459	453	0.169828	719
r32c45k15i45	0.67	635	0.631845	1315	0.525229	1085
r32c60k15i30	0.24	127	0.227306	963	0.242274	1755
r32c60k15i45	1186.22	576477	299.439	1680193	157.991	2495093
r32c60k15i60	SF	SF	SF	SF	SF	SF
r52c75k5i35	SF	SF	SF	SF	SF	SF
r52c75k5i40	SF	SF	SF	SF	SF	SF
r52c75k5i45	SF	SF	TL	TL	SF	SF
r52c75k5i50	SF	SF	SF	SF	TL	TL
r52c75k5i60	SF	SF	SF	SF	SF	SF
r52c75k5i65	SF	SF	SF	SF	SF	SF

Table A.23: Parallel bb-socp with various number of processors Part 3

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
shortfall_100_1	1717.76	9221	702.593	54501	482.779	77703
shortfall_100_2	TL	TL	TL	TL	6000.39	968993
shortfall_100_3	5440.13	22915	935.643	57877	558.6	70385
shortfall_200_1	TL	TL	SF	SF	SF	SF
shortfall_200_2	TL	TL	SF	SF	SF	SF
shortfall_200_3	TL	TL	SF	SF	SF	SF
shortfall_50_1	24.55	685	11.8795	3863	11.731	6397
shortfall_50_2	77.44	2245	23.9174	9541	20.3286	15819
shortfall_50_3	SF	SF	34.4716	14057	19.5376	16287
sssd-strong-15-4	SF	SF	SF	SF	SF	SF
sssd-strong-15-8	TL	TL	TL	TL	SF	SF
sssd-strong-20-4	SF	SF	SF	SF	SF	SF
sssd-strong-20-8	TL	TL	SF	SF	SF	SF
sssd-strong-25-4	SF	SF	SF	SF	SF	SF
sssd-strong-25-8	TL	TL	SF	SF	SF	SF
sssd-strong-30-4	SF	SF	SF	SF	SF	SF
sssd-strong-30-8	TL	TL	TL	TL	TL	TL
sssd-weak-15-4	SF	SF	SF	SF	SF	SF
sssd-weak-15-8	TL	TL	SF	SF	SF	SF
sssd-weak-20-4	SF	SF	SF	SF	SF	SF
sssd-weak-20-8	TL	TL	SF	SF	SF	SF
sssd-weak-25-4	SF	SF	SF	SF	SF	SF
sssd-weak-25-8	SF	SF	TL	TL	SF	SF
sssd-weak-30-4	SF	SF	SF	SF	SF	SF
sssd-weak-30-8	TL	TL	TL	TL	TL	TL
turbine07GF	0.1	45	0.088629	49	0.0902851	49
turbine07_aniso	0.01	1	0.037539	1	0.0407889	1
turbine07_lowb	SF	SF	SF	SF	SF	SF
turbine07_lowb_aniso	SF	SF	4.57462	3591	3.99859	4681
turbine54GF	0.52	29	0.32437	31	0.298755	31
uflquad-nopsc-10-100	35.86	209	11.4417	347	7.25559	373
uflquad-nopsc-10-150	53.83	185	16.8124	213	10.3699	251
uflquad-nopsc-20-100	SF	SF	691.442	13695	404.462	15019
uflquad-nopsc-20-150	TL	TL	2253.59	21221	1071.17	22153
uflquad-nopsc-30-100	SF	SF	1770.16	16981	1142.1	21777
uflquad-nopsc-30-150	TL	TL	4244.77	22499	2853.88	29907
uflquad-nopsc-30-200	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	ML	ML	ML	ML
uflquad-psc-10-100	1.63	7	2.33068	25	2.31981	25
uflquad-psc-10-150	2.98	7	2.97705	15	3.02352	15
uflquad-psc-20-100	8.05	15	9.383	43	8.70948	55
uflquad-psc-20-150	10.4	11	17.7316	47	22.0583	75
uflquad-psc-30-100	17.28	21	23.4968	53	24.3925	75
uflquad-psc-30-150	1.55	1	5.14526	1	5.13061	1
uflquad-psc-30-200	15.73	7	26.5933	7	25.3736	7
uflquad-psc-30-300	53.31	17	111.439	71	106.714	117

Table A.24: Parallel bb-lp with various number of processors Part 1

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
classical_200_1	TL	TL	TL	TL	TL	TL
classical_200_2	TL	TL	TL	TL	TL	TL
classical_200_3	TL	TL	TL	TL	TL	TL
classical_50_1	6778.7	288767	696.049	445175	383.035	569193
classical_50_2	1622.01	25469	128.259	28537	92.8773	48237
classical_50_3	TL	TL	577.746	172461	289.023	179763
estein4_A	0.07	31	0.0838239	31	0.051486	31
estein4_B	0.05	31	0.078388	31	0.0550032	31
estein4_C	0.06	31	0.0832169	31	0.0574532	31
estein4_nr22	0.07	31	0.07481	31	0.665677	31
estein5_A	3.97	801	0.688404	849	0.256333	849
estein5_B	2.29	457	0.459796	573	0.357334	657
estein5_C	2.7	485	0.351837	777	0.481023	795
estein5_nr1	3.34	617	0.498583	789	0.464987	771
estein5_nr21	4.6	803	0.64483	913	0.408265	887
estein6_0	179.76	29307	13.8427	36655	7.17565	33517
estein6_1	89.24	16101	8.1024	23403	3.93412	19119
estein6_2	90.9	16621	8.10321	26491	5.06061	29691
estein7_0	ML	ML	957.714	1993875	392.73	2088976
estein7_1	3867.0	481167	141.35	396251	212.217	1320543
estein7_2	4753.69	701587	447.287	1035369	242.193	1276824
pp-n10-d10	0.67	307	0.784296	7593	0.760043	10981
pp-n10-d10000	2.01	2047	0.292129	2047	0.17641	2047
pp-n100-d10	ML	ML	ML	ML	ML	ML
pp-n100-d10000	ML	ML	ML	ML	ML	ML
pp-n1000-d10	TL	TL	TL	TL	TL	TL
pp-n1000-d10000	ML	ML	ML	ML	ML	ML
robust_100_1	TL	TL	1383.8	92355	584.413	83609
robust_100_2	1471.85	7391	362.816	34277	233.36	45063
robust_100_3	TL	TL	379.859	21553	230.537	22967
robust_200_1	TL	TL	SF	SF	SF	SF
robust_200_2	TL	TL	SF	SF	SF	SF
robust_200_3	TL	TL	SF	SF	SF	SF
robust_50_1	50.05	1929	3.32524	687	6.82417	2175
robust_50_2	21.1	705	9.37669	2725	7.72007	2147
robust_50_3	415.24	16251	26.6433	10653	25.6924	12087

Table A.25: Parallel bb-lp with various number of processors Part 2

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
r12c15k5i10	0.01	51	0.081964	95	0.163528	95
r12c15k5i15	0.0	1	0.03563	1	0.038234	1
r14c18k3i12	SF	SF	0.073209	465	0.0745451	561
r14c18k3i15	SF	SF	0.137293	163	0.133723	245
r14c18k3i18	0.06	155	0.084986	267	0.0881279	305
r14c18k3i9	1.02	2969	0.298147	4939	0.185035	4987
r17c20k5i15	0.0	25	0.067816	33	0.0487061	41
r17c20k5i20	0.01	27	0.101983	41	0.0466778	41
r17c30k3i12	0.24	191	0.0708351	659	0.37801	1217
r17c30k3i15	1967.06	1605185	84.8209	1238853	232.898	5815389
r17c30k3i18	ML	ML	ML	ML	2353.27	71957303
r17c30k3i21	ML	ML	195.615	3621779	114.587	4128099
r17c30k3i24	ML	ML	96.7615	1647945	62.5374	2255757
r17c30k3i27	ML	ML	56.257	991395	83.3365	2927977
r22c30k10i20	0.94	3407	0.32995	5945	0.29279	8695
r22c40k10i20	0.61	595	0.065192	385	0.127106	697
r22c40k10i30	1.43	2143	0.351644	5005	0.49624	10167
r22c40k10i40	6.38	13765	1.10094	17087	0.787226	20411
r23c45k3i21	84.99	43457	2.32555	18341	ML	ML
r23c45k3i24	ML	ML	ML	ML	ML	ML
r23c45k3i27	ML	ML	ML	ML	ML	ML
r23c45k3i30	ML	ML	ML	ML	ML	ML
r23c45k3i33	ML	ML	ML	ML	ML	ML
r23c45k3i36	ML	ML	ML	ML	ML	ML
r27c50k5i25	ML	ML	2209.75	21109205	2049.39	40622287
r27c50k5i30	ML	ML	TL	TL	1859.72	38122021
r27c50k5i35	ML	ML	2119.29	21133587	ML	ML
r27c50k5i40	ML	ML	ML	ML	ML	ML
r27c50k5i45	ML	ML	ML	ML	TL	TL
r27c50k5i50	ML	ML	ML	ML	ML	ML
r32c45k15i30	0.04	105	0.0904739	445	0.079901	927
r32c45k15i45	0.15	269	0.351967	1947	0.306642	1299
r32c60k15i30	0.32	233	0.232655	1289	0.213347	1181
r32c60k15i45	413.76	490851	105.999	1785177	72.0858	2849527
r32c60k15i60	ML	ML	ML	ML	ML	ML
r52c75k5i35	TL	TL	TL	TL	TL	TL
r52c75k5i40	ML	ML	TL	TL	TL	TL
r52c75k5i45	SF	SF	TL	TL	TL	TL
r52c75k5i50	ML	ML	TL	TL	TL	TL
r52c75k5i60	ML	ML	ML	ML	TL	TL
r52c75k5i65	TL	TL	TL	TL	TL	TL

Table A.26: Parallel bb-lp with various number of processors Part 3

	serial		15 proc		30 proc	
problem	CPU t	node	CPU t	node	CPU t	node
shortfall_100_1	TL	TL	5538.58	241837	4119.09	421221
shortfall_100_2	TL	TL	TL	TL	TL	TL
shortfall_100_3	TL	TL	TL	TL	6985.63	1632095
shortfall_200_1	TL	TL	SF	SF	SF	SF
shortfall_200_2	TL	TL	SF	SF	TL	TL
shortfall_200_3	TL	TL	SF	SF	SF	SF
shortfall_50_1	356.38	2653	37.2202	3795	30.1006	5329
shortfall_50_2	547.0	5651	68.9334	10341	52.7635	16875
shortfall_50_3	5658.02	28637	517.159	49009	253.867	42125
sssd-strong-15-4	ML	ML	TL	TL	TL	TL
sssd-strong-15-8	TL	TL	TL	TL	TL	TL
sssd-strong-20-4	ML	ML	ML	ML	ML	ML
sssd-strong-20-8	TL	TL	TL	TL	TL	TL
sssd-strong-25-4	ML	ML	ML	ML	ML	ML
sssd-strong-25-8	TL	TL	TL	TL	ML	ML
sssd-strong-30-4	ML	ML	ML	ML	ML	ML
sssd-strong-30-8	TL	TL	TL	TL	TL	TL
sssd-weak-15-4	ML	ML	ML	ML	987.71	10520307
sssd-weak-15-8	ML	ML	ML	ML	ML	ML
sssd-weak-20-4	ML	ML	ML	ML	1943.54	19938422
sssd-weak-20-8	ML	ML	ML	ML	ML	ML
sssd-weak-25-4	ML	ML	ML	ML	ML	ML
sssd-weak-25-8	ML	ML	ML	ML	ML	ML
sssd-weak-30-4	ML	ML	ML	ML	ML	ML
sssd-weak-30-8	ML	ML	ML	ML	ML	ML
turbine07GF	0.02	17	0.0457449	25	0.0489621	25
turbine07_lowb	1.5	329	1.08244	2003	0.761915	3291
turbine07_lowb_aniso	2.24	657	0.784265	1439	0.803671	2913
turbine54GF	0.02	5	0.0661461	17	0.069093	17
uflquad-nopsc-10-100	189.49	251	37.2164	809	28.1307	725
uflquad-nopsc-10-150	251.69	407	61.4617	791	39.1307	941
uflquad-nopsc-20-100	TL	TL	3417.26	40073	1808.69	67343
uflquad-nopsc-20-150	TL	TL	TL	TL	6967.33	105039
uflquad-nopsc-30-100	TL	TL	TL	TL	6635.91	96475
uflquad-nopsc-30-150	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-200	TL	TL	TL	TL	TL	TL
uflquad-nopsc-30-300	TL	TL	TL	TL	TL	TL
uflquad-psc-10-100	4.05	9	8.23333	37	6.88482	45
uflquad-psc-10-150	9.41	9	16.2988	41	19.4288	59
uflquad-psc-20-100	37.32	25	22.2936	73	22.6454	101
uflquad-psc-20-150	30.13	11	52.6627	67	64.8137	119
uflquad-psc-30-100	63.96	29	54.2788	99	51.6089	131
uflquad-psc-30-150	43.15	3	130.57	27	131.095	27
uflquad-psc-30-200	156.0	7	272.556	53	270.346	89
uflquad-psc-30-300	853.88	29	874.409	97	1039.55	149

Biography

Aykut Bulut was born in Adiyaman, Turkey on July 4, 1986. His father's name is Mustafa and his mother's name is Fatime. He received Bachelor and Master of Science degrees in Industrial Engineering from the Industrial Engineering Department of Middle East Technical University, Ankara, Turkey on May 2009 and July 2011 respectively. He joined the Industrial and Systems Engineering Department of Lehigh University to pursue a doctoral degree in 2011. He served as system administrator in Computational Optimization Research At Lehigh (CORAL) laboratory between 2012 and 2016. He was awarded with Rossin Doctoral Fellowship in 2011 Fall and Spring semesters, Gotshall Fellowship in 2013 Fall and Spring semesters, SAS Institute OR Fellowship in 2013 summer and ISE Department Ph.D. student of the year in 2014. He joined MathWorks as a software engineer on January 2017.