IMPLEMENTATION OF AN INTERIOR POINT CUTTING PLANE ALGORITHM

IMPLEMENTATION OF AN INTERIOR POINT CUTTING PLANE ALGORITHM

By

HAMID R. GHAFFARI

Computational Eng. & Sci.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirments

for Degree

MASTER OF APPLIED SCIENCE

McMaster University

MASTER OF APPLIED SCIENCE, (Sept. 2008) McMaster University

(Computational Eng. & Sci.) Hamilton, Ontario

TITLE: IMPLEMENTATION OF AN INTERIOR POINT CUTTING PLANE ALGORITHM

AUTHOR: HAMID R. GHAFFARI

SUPERVISOR: PROFESSOR TAMÁS TERLAKY

NUMBER OF PAGES: viii, 61

Abstract

In this thesis, first we propose a new approach to solve Semi Infinite Linear Optimization Problems (SILP). The new algorithm uses the idea of adding violated cut or cuts at each iteration. Our proposed algorithm distinguishes itself from Luo, Roos, and Terlaky's logarithmic barrier decomposition method, in three aspects: First, the violated cuts are added at their original locations. Second, we extend the analysis to the case where multiple violated cuts are added simultaneously, instead of adding only one constraint at a time. Finally, at each iteration we update the barrier parameter and the feasible set in the same step. In terms of complexity, we also show that a good approximation of an optimal solution will be guaranteed after finite number of iterations.

Our focus in this thesis is mainly on the implementation of our algorithm to approximate an optimal solution of the SILP. Our numerical experiences show that unlike other SILP solvers which are suffering from the lack of accuracy, our algorithm can reach high accuracy in a competitive time.

We discuss the linear algebra involved in efficient implementation and describe the software that was developed. Our test problem set includes large scale second order conic optimization problems.

Acknowledgement

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the School of Computational Engineering and Science, McMaster University for providing excellent research environment for me to work on my project and thesis.

I am deeply indebted to my supervisor Prof. Tamás Terlaky from the School of Computational Engineering and Science whose help, stimulating suggestions and encouragement helped me in all the time in research and writing this thesis.

I thank friend, Dr. Mohammad Oskoorouchi from the Department of Information Systems and Operations Management, College of Business Administration, California State University San Marcos, for all his help, support and valuable hints.

I also want to thank Dr. Yuriy Zinchenko, Dr. Antoein Deza, and Dr. Tim Davidson as my examination committee members for all their help, useful comments and suggestions on the material covered in my thesis.

Especially, I would like to give my special thanks to my wife Behnaz whose patience and love enabled me to complete this work.

Contents

List of Figures

List of Tables

Chapter 1

Preliminaries

1.1 Linear Optimization and Optimality Conditions

A linear optimization problem (LP) deals with minimizing (maximizing) a linear function subject to finite number of linear constraints. In *standard form*, it can be formulated as

$$\min_{x\in\mathbb{R}^n} \left\{ c^T x \ : \ Ax = b, \ x \geq 0 \right\}, \tag{1.1}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and A is an $m \times n$ real matrix. If x satisfies the constraints $Ax = b, \ x \geq 0,$ we call it a *feasible point*, and the set of all feasible points is the *feasible set.*

Associated with any LP there is another linear optimization problem, called the *dual*

problem, which consists of the same data objects. The dual of (1.1) is

$$\min_{y \in \mathbb{R}^m} \left\{ b^T y \ : \ A^T y + s = c, \ s \geq 0 \right\}, \tag{1.2}$$

where s is a vector in $\mathbb{R}^n$. The components of vector y are called the *dual variables* while s is the vector of *dual slacks.*

Problem (1.1) is often called the *primal problem,* to distinguish it from (1.2), and the two problems together are referred to as the *primal-dual pair.*

For further use, let us define

$$\mathcal{F} = \left\{ (x, y, s) \ : \ Ax = b, \ A^T y + s = c, \ x \geq 0, \ s \geq 0 \right\} \tag{1.3}$$

$$\mathcal{F}^\circ = \left\{ (x, y, s) \ : \ Ax = b, \ A^T y + s = c, \ x > 0, \ s > 0 \right\} \tag{1.4}$$

to be the primal-dual feasible set and its relative interior, respectively.

A *duality theory* that explains the relationship between the two problems (1.1) and (1.2) has been developed during the last sixty years. In 1947, Dantzig introduced the simplex method to solve problem (1.1). Appendix (A.1) contains those aspects of the duality theory that have a direct bearing on the design and analysis of our algorithm. We do not attempt to present a complete treatment of this fascinating topic, we refer the reader instead to standard reference texts [22]. Our main focus is the implementation and numerical property of our new algorithm.

Optimality conditions are derived from the fundamental principles of the duality theory of linear optimization problem. They can also be regarded as a special case of the *optimality*

conditions for general constrained linear optimization problem, known as Karush-Kuhn-Tucker conditions (or KKT) [5]. The optimality conditions for primal problem (1.1) are as follows (see Theorems A.1.1 and A.1.2):

The vector $x \in \mathbb{R}^n$ *is a solution of the problem* (1.1) *if and only if there exist vectors* $s \in \mathbb{R}^n$ *and* $y \in \mathbb{R}^m$ *for which the following conditions hold*

$$A^T y + s = c, \tag{1.5a}$$

$$Ax = b, \tag{1.5b}$$

$$x_i s_i = 0, \quad i = 1, 2, \ldots, n, \tag{1.5c}$$

$$(x, s) \geq 0. \tag{1.5d}$$

This is a crucial result that defines the relation between the primal and dual problems. Formally, we state the dual optimality conditions as follows:
The vector $(y, s) \in \mathbb{R}^m \times \mathbb{R}^n$ *is an optimal solution of* (1.2)*, if and only if there exists a vector* $x \in \mathbb{R}^n$ *such that the conditions* (1.5) *hold.*

By examining the conditions (1.5) from both the primal and the dual point of view, we can conclude that vector (x^*, y^*, s^*) solves system (1.5), if and only if x^* solves the primal problem (1.1) and (y^*, s^*) solves the dual problem (1.2). Vectors x^*, (y^*, s^*), and (x^*, y^*, s^*) are called *primal optimal solution, dual optimal solution,* and *primal-dual optimal solution,* respectively. Excluding (1.5d) from the system of equations (1.5) gives us a simple bilinear equation system that can be solved efficiently by Newton's method. However, satisfying the nonnegativity constraint is a challenging problem.

1.2 Interior Point Methods

1.2.1 History

Although interior-point techniques, primarily in the form of barrier methods, were widely used during the 1960s for problems with nonlinear constraints [9, 21], their use for the solution of the fundamental problem of linear optimization was unthinkable because of the dominance of the simplex method. During the 1970s, the interest in barrier methods decreased, nearly to the point of oblivion by emerging and seemingly more efficient alternatives, such as augmented Lagrangian [9, 23] and sequential quadratic optimization methods. By the early 1980s, barrier methods were almost universally regarded as a closed chapter in the history of optimization. This picture changed dramatically in 1984, when Karmarkar [20] announced a fast polynomial-time interior point method for linear optimization problem. In 1985, a formal connection was established between Karmarkar's method and classical barrier methods [10]. Since then, interior point methods have continued to transform both the theory and practice of constrained optimization.

1.2.2 The Central Path

One of the widely used methods to track the solution of KKT systems is to follow a smooth curve that starts somewhere inside the interior of the feasible set and ends at an optimal solution. The best known such curve is the *central path*. The central path C is an strictly feasible curve that maps any positive real number $\mu > 0$, the so-called *barrier parameter,*

to a solution (x_μ, y_μ, s_μ), that is the unique solution (see [28]) of the following system:

$$A^T y + s = c, \tag{1.6a}$$

$$Ax = b, \tag{1.6b}$$

$$x_i s_i = \mu, \quad i = 1, 2, \ldots, n, \tag{1.6c}$$

$$(x, s) > 0. \tag{1.6d}$$

The unique solution of system (1.5) is called the μ*-center* of the primal-dual problem. This system differs from the KKT conditions only in the term of μ on the right hand side of (1.6c). Instead of the complementarity condition (1.5c), we require that the pairwise products $x_i s_i$ have the same value for all indices i. It is well known that system (1.6) has a unique (see [28]) solution for each $\mu > 0$ if $\mathcal{F}^\circ$ is nonempty. As $\mu \downarrow 0$, the sequence of unique solutions (x_μ, y_μ, s_μ) of μ*-centers* converges to a solution (x^*, y^*, s^*) where x^* is an optimal solution of (1.1) and (y^*, s^*) is an optimal solution of (1.2). Hence, the central path leads us to a solution of the primal and dual problems at the same time.

Most interior point algorithms take Newton steps toward the μ-center on the central path, rather than taking pure Newton steps for equations (1.5). Then, subsequently μ is reduced and the process is repeated. Since, in practice, there is no need to find the exact solution of (1.6), a measure is required to make sure the approximate μ-centers are close enough to the exact ones. There are different methods to measure the distance of a given point (x, y, s) to the μ-center [7]. The *proximity measure* used in our implementation is the one given in [27], which is

$$\delta(x) = \frac{1}{2} \left\| v - v^{-1} \right\|, \tag{1.7}$$

where

$$\nu = \sqrt{\frac{xs(x)}{\mu}}. \tag{1.8}$$

Note that the vector product in (1.8) and the vector inversion in (1.7) is component wise.

The advantage of using proximity (1.7) is not only because it tells us how far (x, y, s) is from the current μ-center (x_μ, y_μ, s_μ), it also tells us whether the given point is close to the boundaries of the positive orthant. In the next subsections, we present three well known directions towards the μ-center.

1.2.3 Primal

This method is based on search directions that are variant of Newton's method applied to the equalities in (1.5) and modifying the search directions and step lengths so that the inequalities $x \geq 0$ are satisfied *strictly* at every iterations. Given an $x > 0$ with $Ax = b$, the Newton direction to solve (1.6) can be computed as follows:

$$\begin{aligned} y &= (AX^2A^T)^{-1}(AX^2c - \mu b), \\ s &= c - A^T y, \\ \Delta x &= x - \frac{1}{\mu}X^2 s, \end{aligned} \tag{1.9}$$

where $X = \text{diag}(x)$. Then x will be updated by choosing a *step length* α such that the updated point minimizes the primal barrier function:

$$\varphi_p(x) = \frac{c^T x}{\mu} - \sum_{i=1}^{n} \log(x_i). \tag{1.10}$$

and the new iterate stays inside the positive orthant:

$$\alpha = \underset{t>0}{\operatorname{argmin}} \left\{ \varphi_p(x + t\Delta x) \ : \ x + t\Delta x > 0 \right\}.$$

In Chapter 3, we discuss some computational linear algebra methods that are essential for efficient and numerically stable implementation of this method.

1.2.4 Primal-Dual

This methods find approximate primal-dual optimal solutions by applying a variant of Newton's method to the three equalities in (1.5). The search directions and step lengths are chosen so that the inequalities $(x, s) \geq 0$ are *strictly* valid at each iterations. The full Newton direction to solve (1.6) is given by:

$$\Delta y = (AXS^{-1}A^T)^{-1}(b - \mu As), \tag{1.11}$$

$$\Delta s = -A^T \Delta y, \tag{1.12}$$

$$\Delta x = \mu s^{-1} - x - XS^{-1}\Delta s, \tag{1.13}$$

where $S = \text{diag}(s)$ and $(x, s) > 0$ is a primal-dual strictly feasible starting point. Then (x, s) will be updated by choosing the step length α such that the updated point minimizes the primal-dual barrier function

$$\varphi_{pd}(x, s) = \frac{x^T s}{\mu} - \sum_{i=1}^{n} \log(s_i x_i), \tag{1.14}$$

and the new iterate stays inside the positive orthant:

$$\alpha = \underset{t>0}{\operatorname{argmin}} \{\, \varphi_d(x + t\Delta x, s + t\Delta s) \;:\; x + t\Delta x > 0,\; s + t\Delta s > 0 \,\}.$$

In Chapter 3, we discuss some computational linear algebra methods that are essential for efficient and numerically stable implementation of this method.

1.2.5 Infeasible Primal-Dual

For the primal-dual direction discussed in the previous section, we assume that the given iterate (x, y, s) is strictly feasible. If one chooses $x > 0$ and $s > 0$, usually for such a choice $Ax - b$ and $c - s - A^T y$ are not zero. In this case, the following direction will lead us to find a μ-centering Newton direction. To solve (1.6):

$$\Delta y = (AXS^{-1}A^T)^{-1}(r_b - AS^{-1}Xr_c + Ax - \mu As^{-1}), \tag{1.15}$$

$$\Delta s = r_c - A^T \Delta y, \tag{1.16}$$

$$\Delta x = \mu s^{-1} - x - XS^{-1}\Delta s, \tag{1.17}$$

where

$$r_b = b - Ax, \tag{1.18}$$

$$r_c = c - s - A^T y. \tag{1.19}$$

If x and (y, s) are primal and dual feasible, respectively, we have the primal dual search direction where r_b and r_c are zero.

1.3 Semi-Infinite Linear Optimization

1.3.1 General Form

Semi-infinite linear optimization (or programming) (SILP) deals with an optimization problem with linear objective and linear constraints in which either the number of constraints or the dimension of the variables space, but not both, is allowed to be infinite. The primary propose of the thesis is to develop and study an algorithm to solve SILP, i.e., programs that can be formulated as

$$\max \left\{ b^T y \ : \ a_t^T y \leq c_t, \ t \in T \right\}, \tag{1.20}$$

where $b \in \mathbb{R}^m$, T is an arbitrary (possibly infinite) index set,

$$a_t = a(t) = (a_1(t), \ldots, a_m^T(t)) \tag{1.21}$$

maps T into $\mathbb{R}^m$, and $c_t = c(t)$ is a scalar function on T.[1] Problem (1.20) is said to be the *Dual SILP.*

Let us observe that the *feasible set* of (1.20);

$$\mathscr{F} = \left\{ y \ : \ a_t^T y \leq c_t, \ t \in T \right\} \tag{1.22}$$

[1] Different sources define *semi-infinite linear optimization* in different manner but equivalent to each other. The above definition is borrowed from Goberna and López, [12]. The reader may find more details and results in [1, 2, 8, 11]

is a closed convex set in $\mathbb{R}^m$, since it is the intersection of a family of closed half spaces. The *semi-infinite linear system* (SILS short form) $\left\{ a_t^T y \leq c_t,\ t \in T \right\}$ provides an external representation of the feasible set $\mathscr{F}$. Therefore, to some extent, (1.20) is a convex linear optimization problem. If (1.20) is consistent, i.e., if it has at least one feasible solution, then the *optimal value* v of problem (1.20) can be either a real number or $+\infty$.

There are two major classes of SILPs. Those where the infinite index set T is countable and those where the index set is the continuous subset of the Euclidean space (such as a line segment or rectangle). We shall call them the *countable* and *continuous* semi-infinite linear programs, respectively. Therefore, we can rewrite the countable case of problem (1.20) as follows:

$$\max \left\{ b^T y \ :\ a_i^T y \leq c_i, \quad i = 1, 2, 3, \ldots \right\}. \tag{1.23}$$

The primal semi-infinite linear optimization problem associated with problem (1.24) is defined as follows:

$$\min \left\{ \sum_{i=1}^{\infty} c_i x_i \ :\ \sum_{i=1}^{\infty} x_i a_i = b, \quad x \geq 0 \right\}, \tag{1.24}$$

where x_i is zero for all $i = 1, 2, 3, \ldots$, except for a finite number of indices for which $x_i \geq 0$ (see [1]).

Example 1.3.1 (Example of Karney)**.** Consider the following primal semi-infinite linear

optimization problem:

$$\begin{array}{lrcl} \min & x_1 & & \\ \text{s.t.} & -x_1 - x_3 - x_4 - \cdots & = & -1, \\ & x_2 + \frac{1}{3}x_3 + \frac{1}{4}x_4 + \cdots & = & 0, \\ & x & \geq & 0 \end{array} \tag{1.25}$$

Problem (1.25) meets its optimal value 1 at point $x_1^* = 1,\ x_i^* = 0,\ i = 2, 3, \ldots$.

The dual problem associated with problem (1.25) is given as follows:

$$\begin{array}{lrcl} \max & -y_1 & & \\ \text{s.t.} & -y_1 & \leq & 1, \\ & y_2 & \leq & 0, \\ & -y_1 + \dfrac{y_2}{i} & \leq & 0, \quad i = 3, 4, \ldots . \end{array} \tag{1.26}$$

The optimal solution is $y = (0, 0)$ with the optimal value 0.

In Example of Karney, although both primal and dual are feasible, however, the duality gap is not zero. The duality theories which dealing with this problem are presented in Appendix A.2.

1.4 Analytic Center Cutting Plane Method (ACCPM)

The analytic center cutting plane method (ACCPM) [16, 29] is an efficient algorithm (see [4, 18]). The complexity of related algorithms where given in [3, 4], and subsequently in [17]. Extensions to deep cuts were given in [13] and to very deep cuts in [15] in which the

method studied corresponds to the practical implementation of ACCPM given in [19] with a single cut.

1.4.1 Analytic Center

The analytic center of $\mathcal{F}_D$ is the unique point maximizing the *dual logarithmic barrier function*

$$\Phi_d(s) = \sum_{i=1}^{n} \log(s_i), \tag{1.27}$$

where $s = c - A^T y > 0$. We now introduce the linear optimization problem

$$\max \left\{ \Phi_d(s) \ : \ s = c - A^T y > 0 \right\} \tag{1.28}$$

and the associated first order optimality conditions

$$\begin{aligned} xs &= \mathbf{1}_n, \\ A^T y + s &= c, \quad s > 0, \\ Ax &= 0, \quad x > 0, \end{aligned} \tag{1.29}$$

where $\mathbf{1}_n$ is the *all one vector* in the euclidian space $\mathbb{R}^n$. Recall that the notation xs indicates the *Hadamard,* or componentwise product of the two vectors x and s. The solution of (1.29) gives also a vector x that is the μ-center of the set $\{ x \ : \ Ax = 0, \ x \geq 0 \}$ with $\mu = 1$. The analytic center of $\mathcal{F}$ can alternatively be defined as the optimal dual solution

of

$$\max \left\{ -c^T x + \Phi_p(x) \ : \ Ax = 0, \quad x > 0 \right\}, \tag{1.30}$$

where

$$\Phi_p(x) = \sum_{i=1}^{n} \log(x_i) \tag{1.31}$$

denotes the *primal log-barrier function*. One easily checks that problem (1.31) shares with (1.28) the same first order optimality conditions.

Finally, we define approximate centers by relaxing the condition $xs = \mathbf{1}_n$ in the first order optimality conditions. Formally, any solution (x, s) of

$$\begin{aligned} \|\mathbf{1}_n - xs\| &\leq \theta, \\ A^T y + s &= c, \quad s > 0, \\ Ax &= 0, \quad x > 0, \end{aligned} \tag{1.32}$$

where $\theta < 1$, defines a pair of θ-approximate centers.

1.4.2 Cut

Now let us define what we mean by a cut for the set $\mathcal{F}$.

A *cut* at $\bar{y} \notin \mathcal{F}$ is given in the form

$$a^T y \leq a^T \bar{y} - \bar{\gamma}.$$

If $\bar{\gamma} > 0$, then the cut $(a; \gamma)$ separates $\bar{y}$ from the feasible set, consequently we say that the

cut is *deep*; if $\bar{\gamma} < 0$, the point $\bar{y}$ stays on the feasible side of the cut, thus we say that the cut is *shallow*; if $\bar{\gamma} = 0$, the cut passes through $\bar{y}$, and we will refer to this cut as a *central cut*.

The concept of shallow and deep cuts was first introduced in the context of the analytic center cutting plane method by [15].

1.4.3 Analytic Center Cutting Plane Algorithm

ACCPM can be shortly stated as follows;

Initialization Let $\mathcal{F}_D^0 = \{ y \in \mathbb{R}^m : 0 \leq y \leq \mathbf{1}_m \}$ be the unit cube and $y_0 = \frac{1}{2}\mathbf{1}_m$ be its center. The centering parameter is $0 < \theta < 1$.

Basic Step y^k is a θ-approximation of the analytic center of $\mathcal{F}_D^k$;

$n_k = 2m + \sum_{j=0}^{k-1} p_j$ the total number of hyper-planes describing $\mathcal{F}_D^k$;

1. The oracle returns the cuts $(a_{n_k+j}; \gamma_{n_k+j})$ for $j = 1, \ldots, p_k$, at y^k;
2. Update $\mathcal{F}_D^{k+1} = \mathcal{F}_D^k \cap \left\{ y : a_{n_k+j}^T (y - y^k) \leq -\gamma_{n_k+j}, j = 1, \ldots, p_k \right\}$;
3. compute a θ-approximation of analytic center of $\mathcal{F}_D^{k+1}$.

If only central cuts are employed, then $\gamma_j = 0$ for all generated cuts.

The computation of a new θ-approximate center, after adding new cuts, will be discussed in a later section. Convergence results and complexity analysis can be found in [14].

In the original ACCPM all the cuts added at each iteration are *central*, while in our algorithm, which is described in Chapter 2, the cuts are always deep. An illustration of the ACCPM is given in Figure 1.1.

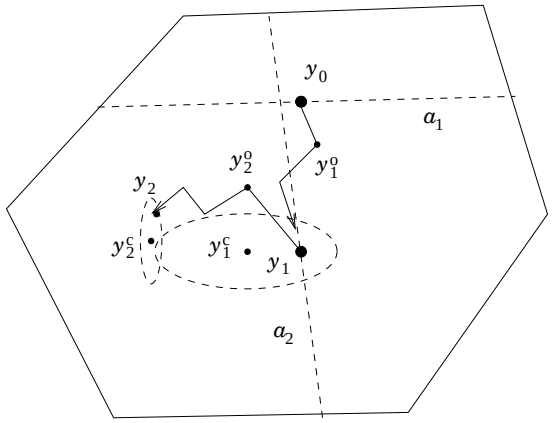

Figure 1.1: The ACCPM algorithm starts at y_0 close to the analytic center of the initial problem. The oracle returns a cut $(a_1; 0)$ passing throw y_0; using *recovery* direction in one step (see [15]), the algorithm gets into an interior point y_1° of the updated feasible set. From there, it takes a few iterations to get inside the θ-neighborhood of the analytic center y_1. Then, the procedure is repeated for the new cut and for the new analytic center y_2.

1.4.4 A Logarithmic Barrier Decomposition Method

Luo, Roos, and Terlaky [22] proposed another *column generation* interior point based approach to solve (1.20). Their algorithm is based on adding only shallow cuts, too. The algorithm can be briefly described as follows:

Algorithm (LBDM):

Initialization Let $\mathcal{F}_D^0 = \{ y \in \mathbb{R}^m : 0 \le y \le \mathbf{1}_m \}$ be the unit cube and $y_0 = \frac{1}{2}\mathbf{1}_m$ be its center. The barrier parameter updating factor is $0 < \eta < 1$, the centering parameter is $0 < \theta < 1$, barrier parameter $\mu = \mu_0$, and $n_0 = 2m$;

Basic Step y^k is a θ-approximate μ_k-center of $\mathcal{F}_D^k$;

$n_k = 2m + k - 1$ is the total number of hyperplanes describing $\mathcal{F}_D^k$;

1. The oracle returns the cut a_{n_k}, at y^k; and let $\gamma_k = 0.25\sqrt{a_{n_k}^T \nabla^2 \varphi_d(y^k)}\, a_{n_k}$;

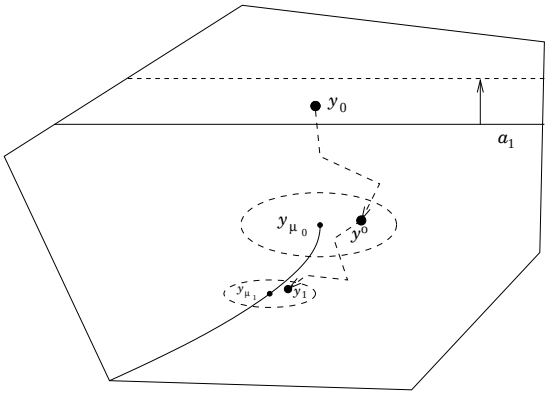

Figure 1.2: The ACCPM method explained in Section 1.4.4 starts at a point y_0 close to the μ_0-center of the initial problem. The oracle returns a violated (deep) cut (a_1, γ_1); the algorithm *shifts* this cut to make y_0 feasible (make it *shallow*). After updating the feasible set by adding the shifted cut, Newton steps are employed to get close to the new μ_0-center to get y°. Again, Newton steps are needed to get close to the μ_1-center to get y_1.

2. Update $\mathcal{F}_D^{k+1} = \mathcal{F}_D^k \cap \left\{ y \ : \ a_{n_k}^T (y - y^k) \le \gamma_k \right\}$;
3. Compute a θ-approximate μ_k-center of $\mathcal{F}_D^{k+1}$.
4. Reduce $\mu_{k+1} = (1 - \eta)\mu_k$;
5. Compute a θ-approximate μ_{k+1}-center of $\mathcal{F}_D^{k+1}$.

Recall that a θ-approximate μ-center is a point in $\mathcal{F}_D$ that is contained in the θ-neighborhood of the μ-center on the central path, that is;

$$A^T \bar{y} + \bar{s} = c, \quad A\bar{x} = b, \quad \left\| \frac{\overline{xs}}{\mu} - e \right\| \le \theta < 1. \tag{1.33}$$

Note: In case that no cut is returned by the oracle, then steps 1, 2, 3 are ignored.

In the next section we present our new approach to solve (1.20) and compare it with the ones discussed in Sections 1.4.3 and 1.4.4.

1.4.5 Convex Optimization Problems and SILP

Consider the following *convex* linear optimization problem:

$$\max_{y \in \mathcal{D}} \left\{ b^T y \ : \ g(y) \leq 0 \right\} \tag{1.34}$$

where $\mathcal{D}$ is a *closed* and *bounded* subset of $\mathbb{R}^m$, and $g : \mathbb{R}^m \to \mathbb{R}$ is a *convex function*. Problem (1.34) is equivalent to the following SILP

$$\max_{y \in \mathbb{R}^m} \left\{ b^T y \ : \ g(\bar{y}) + v(\bar{y})^T (y - \bar{y}) \leq 0, \ \bar{y} \in \mathcal{D} \right\}, \tag{1.35}$$

where the vector $v(\bar{y}) \in \mathbb{R}^m$ is a *sub-gradient* of g at $\bar{y}$, i.e., it satisfies the following inequality:

$$g(y) - g(\bar{y}) \geq v(\bar{y})^T (y - \bar{y}). \tag{1.36}$$

If g is differentiable at $\bar{y}$, then the only sub-gradient of g at $\bar{y}$ is the gradient $\nabla g(\bar{y})$ (See Figure 1.3 for an illustration of sub-gradient cuts).

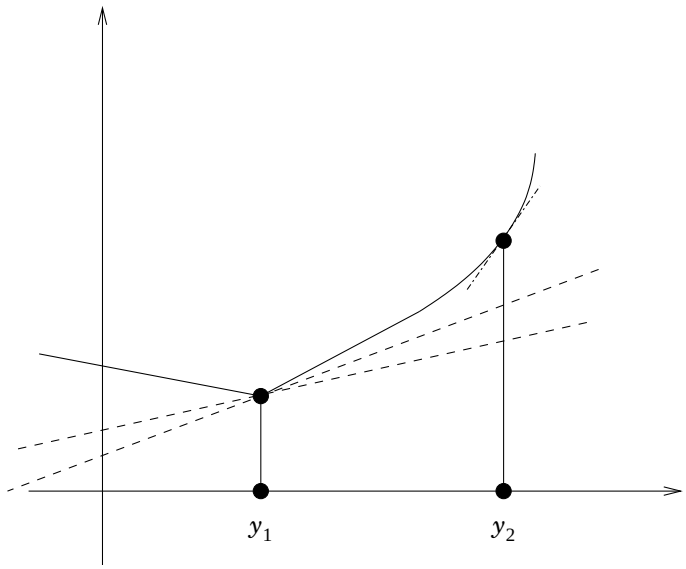

Figure 1.3: The dashed lines indicate two different sub-gradient cuts of the function g at y_1, while the only sub-gradient cut at y_2 is the tangent line.

Chapter 2

A Cutting Plane Method for Semi-Infinite Linear Optimization

2.1 Constraint Generation Algorithm

In this section we present our constraint generation algorithm for solving SILP. The material of this section is selected from paper [26]. Let us consider the SILP (1.20).

We make the following assumptions:

Assumption 1. The index set T is compact and the mappings $t \to a_t$ and $t \to c_t$ are continuous in t.

Assumption 2. The feasible set $\mathscr{F}$ contains a δ-radius full dimensional ball.

Assumption 3. $\mathscr{F}$ is contained in the unit cube $[0,1]^m$ and all m-vectors b and a_t are normalized.

Assumption 1 is made to ensure that the optimal solution of the constraint generation algorithm coincides with that of problem (1.20), see Lemma 2.1.2. Assumption 2 is needed to establish a bound on the number of constraints, and Assumption 3 is a scaling assumption that helps to keep the complexity bound simple.

We now describe the algorithm. Let $\bar{y}$ be a point in the vicinity of the central path of the current LP sub-problem and $\hat{a}_j^T y \leq \hat{c}_j$, for $j = 1, \ldots, p$ be p constraints in $\mathscr{F}$ such that $\hat{c}_j < \hat{a}_i^T \bar{y}$. The updated feasible set therefore reads

$$\mathcal{F}_d^+ = \left\{ s \in \mathbb{R}_+^n, r \in \mathbb{R}_+^p \ : \ A^T y + s = c, \hat{A}^T y + r = \hat{c} \right\},$$

where $\hat{A} \in \mathbb{R}^{m \times p}$ is composed of the p column vectors $\hat{a}_i$, $i = 1, \ldots, p$, and $\hat{c} = (\hat{c}_1; \ldots; \hat{c}_p)^T$. Let $\mu^+ = (1 - \eta)\mu$ be the updated barrier parameter for a later-specified value $0 < \eta < 1$. The task is now to find a point in the vicinity of the central path of the updated discretization, close to the μ^+-center of $\mathcal{F}_d^+$. However, since $\hat{c} < \hat{A}^T \bar{y}$, then $\hat{A}^T y \leq \hat{c}$ represent deep cut constraints for $\mathcal{F}_d$, and thus the current point $\bar{y}$ is not a feasible point of $\mathcal{F}_d^+$. Therefore we first need to derive a strictly feasible point for $\mathcal{F}_d^+$. Let

$$\hat{u} = \arg\min \left\{ \frac{p}{2} u^T V u - \sum_{i=1}^{p} \log u_i \right\}, \tag{2.1}$$

where $V = \hat{A}^T (A \bar{X}^2 A^T)^{-1} \hat{A}$, and $X = \mathrm{diag}(x)$. Define

$$\hat{d} = p(\hat{A}^T \bar{y} - \hat{c})\hat{u}. \tag{2.2}$$

Notice that since $\hat{A}^T \bar{y} - \hat{c} > 0$, and $\hat{u} > 0$, then $\hat{d} > 0$. Let $\alpha < 1 - \theta$ be fixed. We

consider two cases:

1. *Moderately deep cuts*: $\hat{d} < \alpha \mathbf{1}_n$. In this case we show that all violated cuts cross through the Dikin ellipsoid[1] around $\bar{y}$ and the dual feasibility can be recovered using the current point $\bar{y}$.

2. *Very deep cuts*: There exists a constraint for which $\hat{d}_i \geq \alpha$. In this case dual feasibility cannot be recovered. We show that one can recover feasibility in the primal space

$$\mathcal{F}_p^+ = \left\{ x \in \mathbb{R}^n_+, u \in \mathbb{R}^p_+ \ : \ Ax + \hat{A}u = b \right\},$$

and obtain a θ-approximation of the new μ^+-center using the primal barrier function.

Lemma 2.1.1. *Let $\mathcal{F}_P$ and $\mathcal{F}_D$ be the primal and dual feasible regions of the current discretization problem, respectively. Let μ be the barrier parameter, and $(\bar{x}, \bar{s})$ be a point in the vicinity of the central path that satisfies* (1.33)*. Let p violated cuts $\hat{A}^T y \leq \hat{c}$ be added to $\mathcal{F}_d$. Then, for $\hat{d}_i < \alpha < 1 - \theta$ and $\Delta x = -\bar{X}^2 A^T (A\bar{X}^2 A^T)^{-1} \hat{A}\hat{u}$ the vector*

$$x^+ = (\bar{x} + \alpha \Delta x; \alpha \hat{u}),$$

is strictly feasible for $\mathcal{F}_p^+$, and for $\Delta s = A^T (A\bar{X}^2 A^T)^{-1} \hat{A}\hat{u}$ and $\hat{r} = \frac{1}{p}(\alpha \mathbf{1}_n - \hat{d})\hat{u}^{-1}$, the vector

$$s^+ = (\bar{s} + \alpha \Delta s; \hat{r})$$

is strictly feasible for $\mathcal{F}_d^+$. Recall that $\hat{u}^{-1} \in \mathbb{R}^p$ is the component-wise inverse of vector $\hat{u}$.

[1]The Dikin ellipsoid around a vector $s \in \mathcal{F}_D^\circ$ is the set $\left\{ \Delta s \ : \ \Delta s = -A^t \Delta y, \ \left\| S^{-1} \Delta s \leq 1 \right\| \right\}$, where $S = \text{diag}(s)$, see [14] for more details.

Proof. An analogous lemma is presented in [14] for a cutting plane algorithm with multiple cuts where $\mu = 1$ and $\hat{d} = 0$. The directions Δx and Δs defined in this lemma are similar to those of [14]. Therefore the proof in [14], to some extend, remains valid here. In particular $A(\Delta x) + \hat{A}\hat{u} = 0$ is obtained by construction. Furthermore, strict feasibility of the updated vectors $\bar{x} + \alpha\Delta x > 0$ and $\bar{s} + \alpha\Delta s > 0$ are obtained when $\alpha < 1 - \theta$ (see [26], Lemma 7).

We prove that $\hat{A}^T y^+ + \hat{r} = \hat{c}$ and $\hat{r} > 0$. Notice that $A^T(\bar{y} + \Delta y) + \bar{s} + \Delta s = c$. Thus $A^T\Delta y = -\Delta s$ and $\Delta y = -(A\bar{X}^2A^T)^{-1}\hat{A}\hat{u}$. Therefore,

$$\hat{A}^T y^+ + \hat{r} = \hat{A}^T\bar{y} + \alpha\hat{A}^T\Delta y + \hat{r} = \hat{A}^T\bar{y} - \alpha V\hat{u} + \hat{r},$$

and from the KKT optimality conditions of problem (2.1) we have

$$\begin{aligned} \hat{A}^T y^+ + \hat{r} &= \hat{A}^T\bar{y} - \frac{\alpha}{p}\hat{u}^{-1} + \frac{1}{p}(\alpha\mathbf{1}_n - \hat{d})\hat{u}^{-1} \\ &= \hat{A}^T\bar{y} - \frac{1}{p}\hat{d}\hat{u}^{-1} \\ &= \hat{c}. \end{aligned}$$

On the other hand since $\hat{d} < \alpha\mathbf{1}_n$, we have $\hat{r} > 0$. □

Lemma 2.1.1 shows that if the violated cuts are moderately deep then Newton's method can be initiated from x^+ and s^+ to obtain a point in the vicinity of the new central path. In the next section we derive a bound for the number of Newton steps required to update the μ^+-center.

In case 2 (see page 21), when there is at least one very deep cut, dual feasibility cannot be recovered because it is not clear how far away the constraint is from the Dikin ellipsoid.

In this situation one can still recover primal feasibility using x^+, and Newton's method in primal space, to update the μ^+-center. This procedure is repeated until the barrier parameter μ falls within the desired accuracy.

The next lemma, proved in [22], shows that the constraint generation algorithm produces an approximate optimal solution of problem (1.20).

Lemma 2.1.2. *Let $\varepsilon > 0$ be given. Under Assumption 1, if $\bar{y} \in \mathscr{F}$ is in the vicinity of a μ-center with $\mu < \frac{\varepsilon}{n+\sqrt{n}}$, then $\bar{y}$ is an ε-maximizer of problem* (1.20).

We now formally present our algorithm.

Algorithm (CIPM):

Initialization. Initiate $\mathcal{F}_d^0 = [0,1]^m$, $\mu_0 = 1$, $y^0 = \frac{1}{2}\mathbf{1}_m$, $s^0 = \frac{1}{2}\mathbf{1}_n$, $n_0 = 2m$, and $\eta_0 = \frac{1}{9\sqrt{2m}}$, $\theta = 0.25$ and $k = 1$.

While $(n_k + \sqrt{n_k})\mu_k \geq \varepsilon$ **do**

Step 1. Identify p_k deep cuts $(\hat{A}^k)^T y \leq \hat{c}^k$, for $\mathscr{F}$ such that $\hat{c}^k > (\hat{A}^k)^T y^k$.

Step 2. Update $n_k = n_{k-1} + p_k$, $\eta_k = \frac{1}{9\sqrt{n_k}}$, $\mu_k = (1-\eta_k)\mu_{k-1}$

$$A^k = [A^{k-1} \;\; \hat{A}^k], \text{ and } c^k = (c^{k-1}; \hat{c}^k).$$

Step 3a. Compute $\hat{u}$ from (2.1) and $\hat{d}$ from (2.2). If $\hat{d} < \alpha\mathbf{1}_n$, then use s^+ to start a dual Newton procedure to obtain s^k and $x^k := x(s^k)$ in the vicinity of the μ_k-center of $\mathcal{F}_d^k$.

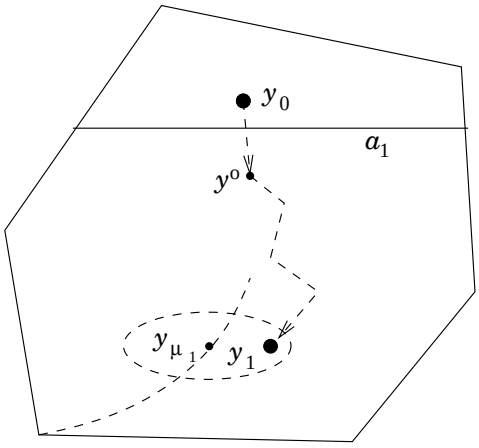

Figure 2.1: The method explained in Section 2.1 starts at a point y_0 close to the μ_0-center of the initial problem. The oracle returns a violated cut $(a_1; \gamma_0)$ assuming $(a_1; \gamma_0)$ is a *moderately deep* cut, one *recovery* step leads us to an interior point y°. After updating the feasible set by adding the cut $(a_1; \gamma_1)$ and updating μ_0 at the same time to μ_1, Newton's method is employed to get close to the μ_1-center to get y_1. If $(a_1; \gamma_1)$ is a *very deep* cut, then we recover infeasibility by employing Newton steps for the primal problem.

Step 3b. Otherwise use x^+ to start a primal Newton procedure to obtain x^k and $s^k := s(x^k)$ in the vicinity of the μ_k-center of $\mathcal{F}_p^k$.

Step 4. $k = k + 1$.

End

2.2 Comparison

The Algorithm CIPM, described in Section 2.1, is a variant of the Algorithm LBDM, presented in Subsection 1.4.4, with major differences both from the theoretical and implementation viewpoints. There are three main theoretical enhancements:

First: our algorithm adds violated cuts with no changes to the right hand side. In Algorithm LBDM, when a violated constraint is identified, it is relaxed by changing its right hand side to make the current μ-center strictly feasible, which of course results in loss of information. We keep the violated constraints as deep as they are.

Second: we extend the analysis to the case where multiple violated cuts are added simultaneously instead of adding only one constraint at a time.

Third: at each iteration we update the barrier parameter together with updating the feasible region in the same step.

Chapter 3

Implementation

Our major concern in this thesis is the , Algorithm CIPM, implementation of an algorithm that can solve SILP problems, including any LP with $n \lll m$. A successful piece of optimization software requires more than just a good algorithm. Having a *good* code requires good knowledge of the algorithm, linear algebra and linear optimization problem, of course.

The implementation, as well as the algorithm, can be divided into two tasks, *oracle* and *computations*. Although an oracle is highly case dependent, and it generally differs from one problem to another, we consider the oracle as a main part of the implementation because it has significant impact on the efficiency of the software.

3.1 Oracles

Recall that $\mathscr{F}$ is the dual feasible set of the semi-infinite linear problem (1.20). By an *Oracle* we mean a machinery to find either that a given point $\bar{y}$ is in $\mathscr{F}$, or it returns a finite non-empty subset T^k of T such that

$$a_t^T y > c_t,\ t \in T^k.$$

Each specific SILP might need its own oracle according to the constraint generator functions. Here we discuss two main strategies that we use for two types of test problem sets that are presented in the next chapter.

3.1.1 Direct Random Search

The first search method that we use is the so called *Direct Random Search* (DRS). In this method, we randomly check all the possible indices in the parameter set until we find as many violated constraints as we need. The the identified violated constraints are added to the current localization as cuts, and the indices of these constraints are removed from the set od constraints to be checked.

The direct random search method has advantages and disadvantages, too. The important advantage is that this method works for all types of SILP problems and, in fact, sometimes it is the only possible choice (see Examples 4.1.1 and 4.1.5). However, it gets too slow as the algorithm gets closer to the optimal solution, i.e., when we need to check all the constraints to find violated constraints, when in fact there is none.

3.1.2 Sub-Gradient Method

Consider Problem (1.34). To find out if there is any violated cut at $\bar{y}$ among the cuts in (1.35) one can easily check the value of $g(\bar{y})$. There are two cases:

1. *if* $g(\bar{y}) > 0$, *then for any sub-gradient* v *the cut*

$$g(\bar{y}) + v^T(y - \bar{y}) \leq 0$$

   is violated by $\bar{y}$, *then the oracle returns one or more of these cuts;*

2. *if* $g(\bar{y}) \leq 0$, *then* $\bar{y}$ *is feasible for all the constraints and the oracle returns the empty set;*

The complexity of finding v strongly depends on g. For instance, if g is differentiable, simply, the gradient ∇g at $\bar{y}$ is the only candidate for v. Examples 4.2 and 4.5 are of this kind.

3.2 Implementation

The implementation of our algorithm presented in Section 2.1 is done in a MATLAB code to solve linear optimization problem (1.20). We assume that the feasible set of (1.20) is bounded (contained in the unit cube), however, in general the implemented code dose not require this assumption.

3.2.1 The Initial Problem

After initialization of a specific problem and the related parameters, we use an infeasible primal-dual technic, see Section 1.2.5, to get close to the μ-center with $\mu = 1$ of

$$\max \left\{ b^T y \ : \ lb \leq y \leq ub \right\}.$$

The vector bounds lb and ub are considered as the dynamic boundary conditions on y. We use these boundaries to make sure we always have a bounded feasible set. The algorithm dynamically increases the magnitude of lb and ub as any component of y gets close to the corresponding component of lb and ub. The advantage of this strategy is to handle problems with unbounded feasible sets by monitoring the increment of each components of the boundary vectors. Thus, the algorithm starts with the following matrix A and right hand side vector c;

$$A = \begin{pmatrix} 1 & \cdots & 0 & -1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & -1 \end{pmatrix}_{m \times 2m}, \quad c = \begin{pmatrix} ub \\ -lb \end{pmatrix}_{2m \times 1}.$$

By default, we set $ub = \mathbf{1}_m$ to be the all one vector in $\mathbb{R}^m$ and $lb = \mathbf{0}_m$ to be the zero vector in the same space. The implementation is capable of accepting user input of an initial set of linear constraints as well. We then simply add the corresponding coefficient matrix to A and its right-hand side to vector c.

The first step is to get close to the μ-center of the initial problem. The reason that we perform this centering instead of taking the middle point is that, in general, we do have

some initial linear constraints that would be added to the block constraints at initialization. For this purpose, an *infeasible primal-dual* method is employed (see Section 1.2.5).

3.2.2 Recovering Feasibility

The main loop of our algorithm starts by checking the current value of μ given in Lemma 2.1.2 as the stopping criterion. Then the specific oracle is called to return a violated cut, if there are any. Clearly, by adding deep cuts to the feasible set, the current iterate $\bar{y}$ becomes infeasible. However, by using Lemma 2.1.1, we can easily recover feasibility when the added cut is *moderately deep.* We use the same method, no mater if the added cut is very deep or moderately deep. The reason is that after this update we have primal feasibility anyway, and the updated $\bar{x}$ is a good warm start point for Newton's method in the primal centering process, (see [15] for more details). As a result, we can recover dual feasibility either by a simple dual steps, or if it is needed, after primal centering steps.

3.2.3 Primal Centering

In primal centering steps, we use *damped Newton* steps to get inside the θ-neighborhood of the μ-center of the primal problem

$$\min \left\{ c^T x \ : \ Ax = b, \quad x \geq 0 \right\}, \tag{3.1}$$

which is equivalent to solving the KKT system (1.5). We update x by using the Newton's direction which is the solution of (1.9). This iterative Newton procedure leads to a θ-approximate solution of (1.5).

To find Δx from (1.9) we first decompose the coefficient matrix AX^2A^T into its Cholesky factors:

$$AX^2A^T = LL^T.$$

Then we solve two triangular linear systems $Lz = AX^2c - \mu b$ and $L^Ty = z$, using forward and backward solving. We implemented subroutines for this purpose that are faster than the analogous built in MATLAB functions. Therefore, this way, we have more control on these problems. One of the challenges in solving the KKT system is when x becomes close to the boundary. In this case, the diagonal matrix X^2 has components close to zero, and consequently the matrix AX^2A^T becomes *ill-conditioned.* In our implementation, we use the Modified Cholesky Decomposition technic proposed by Wright [28]. Due to numerical errors, the search direction Δx computed by (1.9) not always satisfy $A\Delta x = 0$. To overcome this problem we project the computed Δx into the null space of A as follows:

$$\Delta x \longleftarrow \left(I - A^T(AA^T)^{-1}A\right)\Delta x. \tag{3.2}$$

To fulfill the non-negativity constraints in (1.6d), we choose a step length α that ensures

$$x + \alpha\Delta x > 0. \tag{3.3}$$

The maximum value of α, when at least one of the components of the updated point hits the boundary of the positive orthant, is calculated as follows:

$$\alpha_{\max} = \max\left\{ -\frac{x_i}{\Delta x_i} \ : \ \Delta x_i < 0 \right\}. \tag{3.4}$$

On the other hand, we want to optimize the primal barrier function φ_p given in (1.10). So, we choose $\hat{\alpha}$ to be

$$\hat{\alpha} = \text{argmin}\left\{ \varphi_p(x + \lambda\Delta x) \ : \ 0 \le \lambda \le \alpha_{\max} \right\}.$$

After such a primal centering step we keep iterating until x reaches the θ-neighborhood of x_μ, that provides us a θ-approximate μ-center in $\mathcal{F}_P$.

Chapter 4

Computational Results

In this section we test our algorithm on two sets of problems. First we study the convergence behavior of the algorithm on some classical SILP selected from [6], and then we show the capability of our algorithm to solve a class of large scale SOCO problems. The set of SOCO test problems is randomly generated. We set the accuracy of all examples to be 10^{-8}, unless otherwise is stated.

All the test problems were solved on a desktop computer using Intel(R) Core(TM)2 Quad CPU 2.66 GHz processor with 4 GB RAM.

4.1 Examples

4.1.1 Examples Using Direct Random Search in the Oracle

In the following examples, Examples 4.1.1 through 4.1.5, we solve an linear optimization problem of the form

$$\min_{y\in\mathbb{R}^m}\{\ f(y)\ :\ g(y,t)\le 0,\ \forall t\in T\ \}, \tag{4.1}$$

where $g(y,t)$ is a linear function of y for a given t in the compact set T.

To solve these problems of this form by our constraint generation algorithm, we need to convert problem (4.1) into the form of problem (1.20). To do this, at each iteration an oracle is used to discretize T and identify multiple violated constraints using a random search. The violated constraints are then added to the relaxation problem as new constraints, and the μ-center is updated at the same time. At each iteration of the algorithm, therefore, we deal with a relaxation problem in the form of (1.1), and its corresponding primal problem (1.2), which is a restricted form of the primal of the original problem.

Example 4.1.1. [6], In this example dual dimension $m = 3$, and the one dimensional parameter set $T = [0,1]$ is uniformly partitioned into 1000 pieces. We have

$$f(y) = -\sum_{i=1}^{m}\frac{y_i}{i}, \qquad g(y,t) = \tan(t) - \sum_{i=1}^{m} y_i t^{i-1}, \qquad t\in[0,1].$$

The optimal solution of this problem is $y^* = (0.089073; 0.423147; 1.0450756)$, with the optimal value $b^T y^* = -0.6490412$.

Figure 4.1 shows the convergence behavior of our algorithm for Example 4.1.1. In this

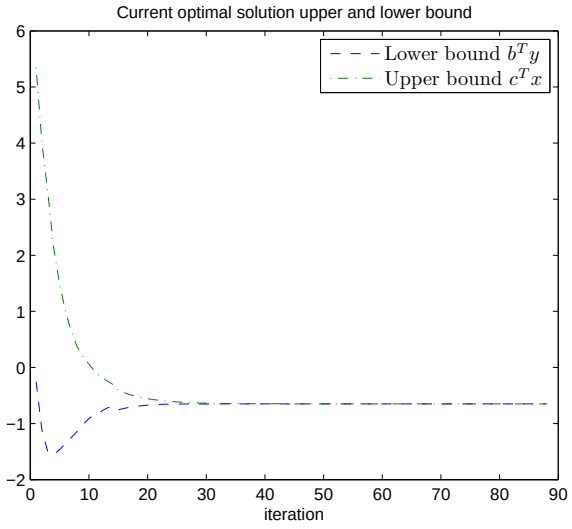

Figure 4.1: The primal and the dual objective values when solving Example 4.1.1. The horizontal axis represents the iteration count.

figure we plot the objective values of the relaxed dual and restricted primal problems at the current approximate μ-center of each iteration. The upper (lower) curve gives the value of $c^T x$ ($b^T y$), the objective function of the restricted primal (relaxed dual) problem, at the current approximate μ-center.

This figure illustrates that our algorithm quickly approaches the optimal value with a reasonably small duality gap. Observe that a good (10^{-4} accuracy) approximation of the optimal solution is achieved in less than 40 iterations. However, to get a high precision (10^{-8}) solution, we let the algorithm run for about 90 iterations.

Notice that since the primal feasible region of the discritized problem is also feasible for the primal of the original problem, therefore $c^T x$ at the μ-center always gives an upper bound on the optimal objective value, i.e., the upper curve never crosses the optimal value line. However, this is not true for the lower curve. This curve is obtained by evaluating

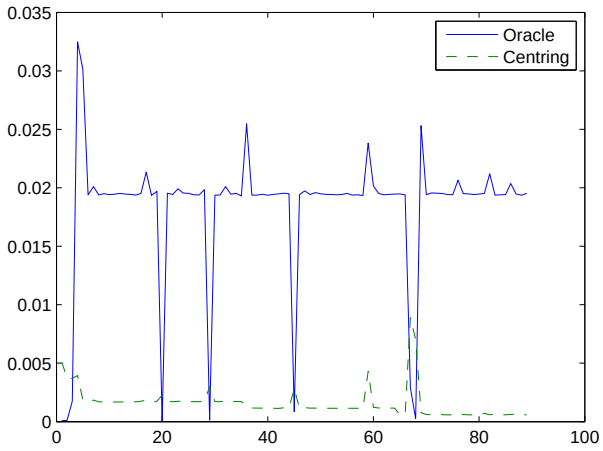

Figure 4.2: CPU time used by the primal centering and the oracle process in Example 4.1.1. Considering the fact that calling the oracle and the centering procedures are the most expensive part of the algorithm, one can imply that most of the time used by our algorithm is dedicated to the oracle. This is resulted by the nature of the direct random search oracle.

$b^T y$, the objective value of the dual problem at a feasible point of the relaxation. Therefore this point is not necessarily feasible for the original dual problem. This is the reason why the lower curve may cross the optimal value line at the early iterates.

Another point to be noticed is the CPU time used by the oracle and the primal-centering process when solving the problem of Example 4.1.1. Figure 4.2 compares the time used by calling the oracle and centering procedures at each iteration. In this example, the oracle (solid line) takes the most proportion of time used by the algorithm. This is resulted by the nature of the direct random search technic used in the oracle.

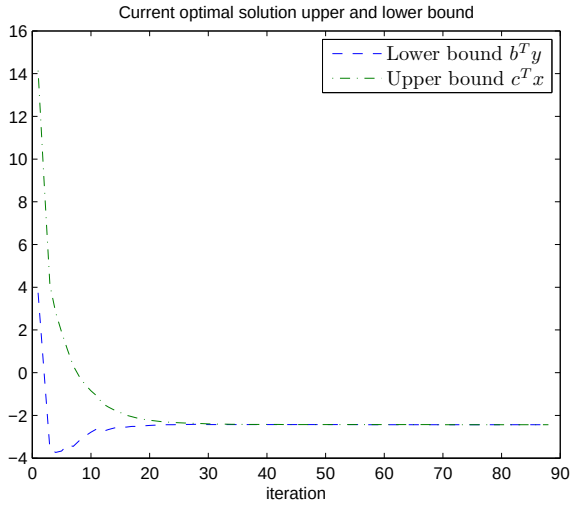

Figure 4.3: The primal and the dual objective values of Example 4.1.2 along the iterations.

Example 4.1.2. [6] Let $y \in \mathbb{R}^5$, $T = [0,1] \times [0,1]$, and

$$f(y) = -y_1 - \frac{1}{2}y_2 - \frac{1}{2}y_3 - \frac{1}{3}y_4 - \frac{1}{4}y_5 - \frac{1}{3}y_6,$$
$$g(y,t) = e^{t_1^2+t_2^2} - \left(y_1 + t_1 y_2 + t_2 y_3 + t_1^2 y_4 + t_1 t_2 y_5 + t_2^2 y_6\right), \qquad t \in [0,1] \times [0,1].$$

The optimal solution of this problem is

$$y^* = (2.5782999, -4.106585, -4.0981235, 4.2450596, 4.5222404, 4.2370932)^T,$$

and the optimal objective value is $b^T y^* = -2.4338899$. The convergence behavior of this problem is shown in Figure 4.3.

The oracle and centering time usage for Examples 4.1.2 through 4.1.5 is mostly the same as the one shown in Figure 4.1, therefore we do not repeat these figures here.

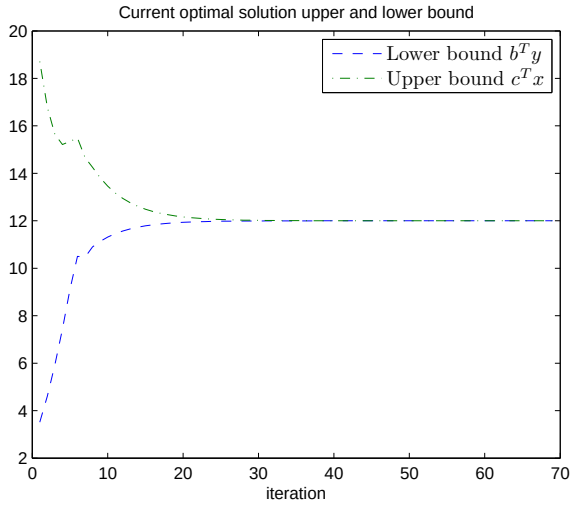

Figure 4.4: The primal and the dual objective values of Example 4.1.3 along the iterations.

Example 4.1.3. [6] Let $y \in \mathbb{R}^6$, $T = [-1,1] \times [-1,1]$, and

$$f = -\left(4y_1 - \frac{2}{3}(y_4 + y_6)\right)$$

$$g = y_1 + t_1 y_2 + t_2 y_3 + t_1^2 y_4 + t_1 t_2 y_5 + t_2^2 y_6 - 3 - (t_1^2 + t_2^2) \qquad t \in [-1,1] \times [1-,1].$$

The optimal solution of this problem is $y^* = (3,0,0,0,0,0)^T$, and the optimal objective value is $f^* = -12$. See Figure 4.4 to see the behavior of the upper and the lower bounds of the approximate optimal solutions.

Example 4.1.4. [6] Let $y \in \mathbb{R}^3$, $T = [-1,4] \times [-1,4]$, and

$$f = -(2y_1 + 4y_2 + y_3), \qquad g = \sum_{i=1}^{3}(1 - y_i)\gamma_i(t_1,t_2) - \frac{1}{2}, \qquad t \in [-1,4] \times [1-,4],$$

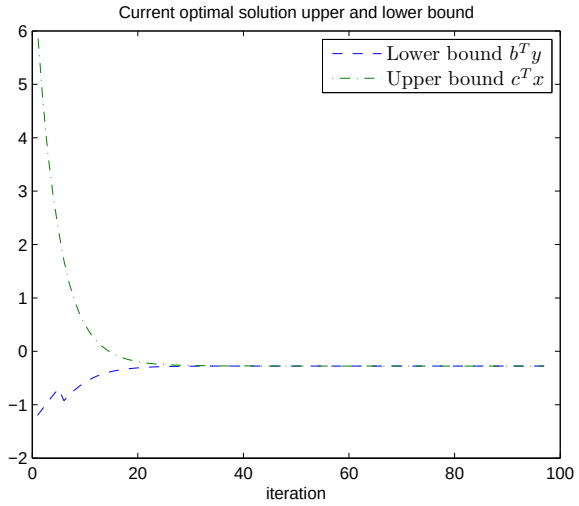

Figure 4.5: The primal and the dual objective values, upper and lower curve, of Example 4.1.4 along the iterations.

where

$$\gamma_1(t_1, t_2) = (1/t_1)(exp((-1/t_1)(1 + (t_2 - 1)^2)) \quad \text{if} \quad t_1 > 0,$$

$$\gamma_2(t_1, t_2) = (1/t_1)(exp((-1/t_1)(2 + t_2^2/4)) \quad \text{if} \quad t_1 > 0,$$

$$\gamma_3(t_1, t_2) = (1/(t_1 - 2)(exp((-1/(t_1 - 2))(1 + (t_2 + 1)^2)) \quad \text{if} \quad t_1 > 2,$$

$$\gamma_{1,2,3}(t_1, t_2) = 0 \quad \text{elsewhere.}$$

This problem has also bounds $0 \leq y_i \leq 1,\ i = 1, 2, 3,$ on the variables.

The optimal solution is $y^* = (0, 0, 0.2752207)^T$, with optimal objective value $f^* = -0.275209$.

The objective function behavior is shown in Figure 4.5.

Example 4.1.5. [6] Consider Example 4.1.4 but with no bound on the variable y. Then, the approximate optimal solution is $y^* = (1.5425, -2.1014821, 0.9345579)^T$, which gives

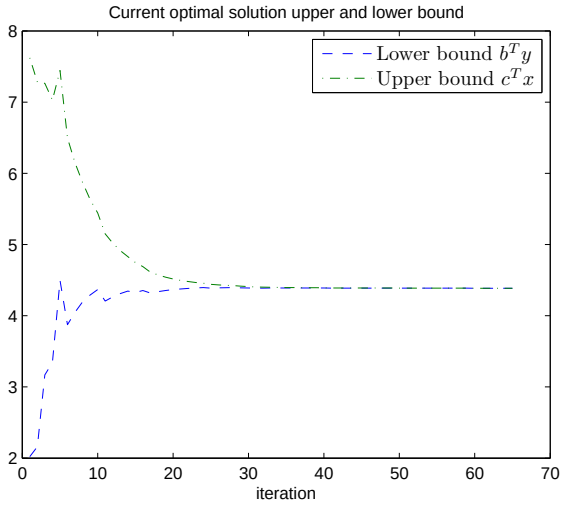

Figure 4.6: In this figure, the upper and the lower bounds of the optimal value of Example 4.1.5 converge to the same point which is an approximate optimal value.

the objective value $f^* = 4.3862$. The lower and upper bounds of the optimal value at each iteration is given in Figure 4.6.

Notice that in Figures 4.3 – 4.6 the lower and upper curves are *not* monotonically approaching the optimal value when the current iterate is far from the optimal solution. This phenomenon is due to the fact that these bounds are computed by evaluating the objective functions of the relaxed dual problem and its corresponding primal problem at the current approximate μ-centers. When violated cuts are identified, the feasible region of the relaxed dual problem is updated by adding new constraints. Since this problem is not solved to optimality, but evaluated at the approximate μ^+-center, the value of the objective function is unpredictable at every steps of the algorithm. However, as we get closer to the optimal solution of the original problem, these fluctuations reduce, and the lower and upper curves become lower and upper bounds of the optimal objective value. Therefore, from this

point the curves monotonically approach the optimal value.

Note that for all examples a good (10^{-4} precision) solution is obtained in about 40 iterations, and most notably our algorithm was able to produce high precision (10^{-8}) solutions in about 90 iterations. It is also worth understanding that at each cases the time spent in the oracle is significantly higher than the time used to re-center.

4.1.2 Examples a Using Sub-Gradient Method in the Oracle

Example 4.1.6. Consider the following quadratic optimization (QO) problem

$$\max \left\{ b^T y \ : \ (y-q)^T W^2 (y-q) \leq \prod_{i=1}^{m} w_i^2 \right\}, \tag{4.2}$$

where $b = (b_1, \ldots, b_m)^T$, $w = (w_1, \ldots, w_m)^T$, and $q = (q_1, \ldots, q_m)^T$ are given vectors in $\mathbb{R}^m$, $W = diag(w)$, and $y \in \mathbb{R}^m$ is the vector of variables.

Define the constraint function as

$$g(y) = (y-q)^T W^2 (y-q) - \prod_{i=1}^{m} w_i^2.$$

Then, the feasible set of the QO problem (4.2) is the subset of $\mathbb{R}^m$ that contains the points inside and on the boundary of the ellipsoid $g(y) = 0$. We can express this feasible set as the intersection of all the half-spaces introduced by the tangent lines to the boundary of the ellipsoid (see Figure 4.7).

Let $t = (t_1, \ldots, t_m)$ be a point on the boundary, i.e., $g(t) = 0$. The gradient of g is

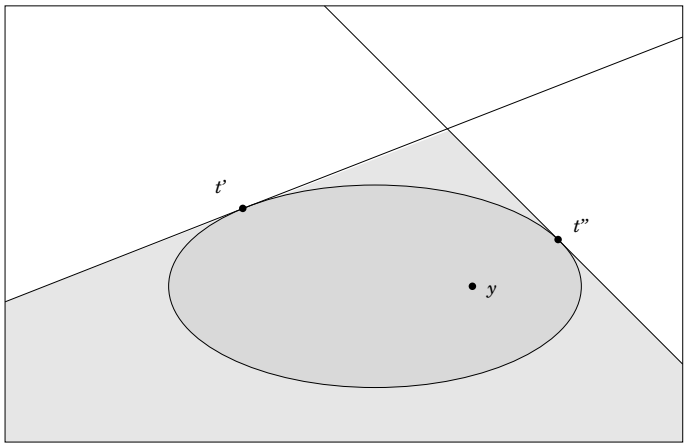

Figure 4.7: This figure shows how the feasible region of Example 4.1.6 can be considered as a SILP, by considering it as the intersection of infinitely many half-spaces introduced by the lines tangent to the boundary.

$\nabla g(y) = 2W^2(y-q)$. Using the first order Taylor expansion of g around vector we have

$$\begin{aligned} g(y) &\approx g(t) + \nabla g(t)^T(y-t) \\ &= (t-q)^T W^2(t-q) + 2(t-q)^T W^2(y-t) - \prod_{i=1}^{m} w_i^2. \end{aligned} \tag{4.3}$$

By replacing $g(y)$ by its linear approximation (4.3) at any boundary point t, the optimization problem (4.2) is equivalent to the following SILP problem

$$\max \left\{ b^T y \ : \ (t-q)^T W^2(t-q) + 2(t-q)^T W^2(y-t) \leq \prod_{i=1}^{m} w_i^2, \ t \in T \right\}, \tag{4.4}$$

where the parameter set T is the set $g^{-1}(\{\ 0\ \})$.

We experiment with our algorithm to solve problems in the form of problem (4.2). We chose $m = 3,\ w = (1,1,1)^T,\ q = (0,0,0)^T,$ and $b = (1,1,1)^T$.

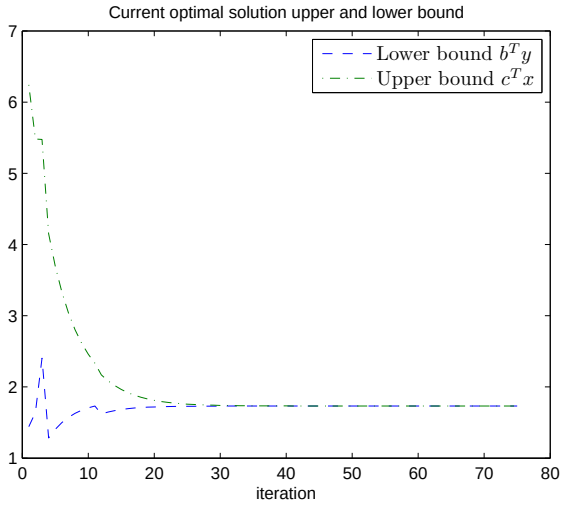

Figure 4.8: The primal and the dual objective values, upper and lower bounds, of Example 4.1.6 along the iterations.

After solving this problem with our algorithm, the approximate optimal solution is $y^* = (0.5126, 0.7126, 0.5067)^T$ and the approximate optimal value is $b^T y^* = 1.7320508$. Note that in this specific case, the solution can be analytically verified, i.e., the exact optimal solution is $(1/2, 1/\sqrt{2}, 1/2)$ and the real optimum value is $\sqrt{3}/2$. Note that our approximate optimal value coincides with the exact optimum up to 8 digits.

Upper and lower bounds, the primal and the dual objective values, of the optimal point at each iteration is given in Figure 4.8. The CPU time usage given in Figure 4.9 shows that the time used by the oracle is significantly less than the time used by the centering procedure. The reason is that in this example a sub-gradient based oracle is used, see Section 3.1.2.

Example 4.1.7. As the second large-scale test set we choose to implement our algorithm to solve general SOCO problems using randomly generated data.

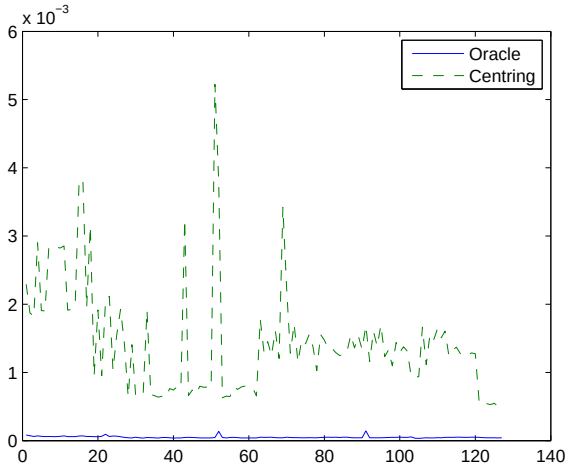

Figure 4.9: This figure shows the CPU time used by the oracle and the centering process in solving Example 4.1.6. In this example, the sub-gradient based oracle is employed.

Consider the following SOCO problem:

$$\max_{y\in\mathbb{R}^m}\left\{\, b^T y \;:\; lb \le y \le ub,\ (c_j - A_j^T y) \in \mathbb{L}^{n_j},\ j = 1,2,\ldots,k \,\right\}, \tag{4.5}$$

where b is a non-zero vector in $\mathbb{R}^m$, $lb < ub$ are real vectors indicating lower and upper bounds of y respectively, and $\mathbb{L}^n$ is an $n-$dimensional *Lorentz Cone*, defined by:

$$\mathbb{L}^n = \left\{\, s \in \mathbb{R}^n \;:\; \sqrt{s_2 + \cdots + s_n} \le s_1 \,\right\}. \tag{4.6}$$

The boundary constraints $lb \le y \le ub$ are added to ensure bounded feasible set. We use the MATLAB function "`randn.m`" to generate data for matrices A_j and vectors c_j from a

			Optimum value		CPU time (in second)		
k	$\bar{n}$	cuts	CIPM	SeDuMi	CIPM	Oracle	SeDuMi
3	1E+6	30	2.9913802	2.9913802	19	18	99
9	5E+5	31	2.9878005	2.9878005	26	24	260
27	1E+5	30	2.9751433	2.9751433	11	10	105
81	5E+4	31	2.9492360	2.9492360	14	13	147
243	1E+4	26	2.8790904	2.8790904	10	9	86
729	5E+3	32	2.8021491	2.8021491	13	11	135
2187	1E+3	31	2.5023178	2.5023178	11	10	158
6561	5E+2	28	2.3104156	2.3104156	31	27	248
19683	1E+2	33	1.7423363	1.7423363	110	101	115
59049	5E+1	29	1.2555162	1.2555162	861	854	1120

Table 4.1: Comparison of CPU times of SeDuMi and our CIPM implemented on a randomly generated SOCO problem set with $m = 3$ and different values of $\bar{n}$ and k. In this experiment we chose the duality gap in both solvers to be less than or equal to 10^{-8}. The Oracle column shows the proportion of time used by the oracle in CIPM algorithm.

normal distribution with mean zero and standard deviation one. We let

$$c_{j1} = 2\sqrt{c_{j2}^2 + \cdots + c_{jn_j}^2}, \quad j = 1, \ldots, k$$

to ensure feasibility, and $n_j = \bar{n}$ for all j, and thus $n = \sum_{j=1}^{k} \bar{n}_j + 2m$.

At each iteration of the constraint generation algorithm an oracle is called to return a cut for a violated second-order cone constraint. This is obtained by computing the gradient of the constraint functions at the current approximate μ-center. If no violated constraint is detected the algorithm is continued by updating the centering parameter. Our oracle uses a random search for identifying violated constraints. This technique works well when the number of cones (k) is relatively small. A more efficient technique is needed to detect violated cuts for problems with large number of conic constraints.

			gap		CPU time in second		
k	$\bar{n}$	cuts	CIPM	SeDuMi	CIPM	Oracle	SeDuMi
2	1E+6	44	5.74E-03	–	139	134	–
4	5E+5	44	5.94E-03	–	91	87	–
8	1E+5	46	6.05E-03	6.93E-03	25	19	154
16	5E+4	44	5.72E-03	8.34E-03	14	7	166
32	1E+4	46	5.83E-03	2.06E-03	10	4	87
64	5E+3	49	6.13E-03	8.25E-03	9	3	72
128	1E+3	49	5.95E-03	4.06E-03	9	3	18
256	5E+2	53	6.17E-03	3.50E-03	7	2	15
512	1E+2	59	6.26E-03	1.71E-03	6	1	8
1024	5E+1	61	6.59E-03	2.45E-03	5	1	5
2048	1E+1	63	6.58E-03	1.85E-03	5	1	2

Table 4.2: Comparison of CPU time of SeDuMi and CIPM implemented on randomly generated SOCO problem with $m = 30$ and different values of $\bar{n}$ and k. The software process was stopped when the duality gap reached 10^{-3}.

Tables 4.1 and 4.2 show the numerical results of this implementation. Each row shows a different random problem with characteristics given in the first two columns: k, the number of second-order cone constraints in problem (4.5) and $\bar{n}$, the size of each cone, respectively. The column under "cuts" presents the number of gradient inequalities needed to add until an approximate optimal solution is reached.

The next pair of columns in Table 4.1 compare the optimal objective values by solving SOCO problem by our constraint generation algorithm CIPM and SeDuMi. The corresponding columns in Table 4.2 illustrate the duality gap at the final solution. The CPU times used to achieve these values by CIPM and SeDuMi, rounded to the nearest integer in seconds, are reported in the last two columns of these tables. For our algorithm, we report CIPM and Oracle to show the total CPU time used by our algorithm and the CPU time used by the oracle.

A close examination of these results reveals that our constraint generation algorithm outperforms classical interior point methods when we deal with problems with large number of conic constraints of large size, and when m, the dimension of y is relatively small. Except for the last two instances of Table 4.2, our algorithm outperforms SeDuMi in terms of cpu time. However, it should be mentioned that primal-dual interior point methods, and in particular SeDuMi, is superior to our constraint generation algorithm for problems with small to moderate values of k, n, and for large values of m.

Table 4.1 reveals an interesting information. When m is small, duality gap 10^{-8} is achieved quickly in all of the test problems. This is not a typical behavior of cutting plane methods. These techniques are known to have difficulties near the optimal solution (see [24, 25]). As m increases the algorithm returns to its traditional performance. This is the reason that in Table 4.2 we run the test problems to only three digits of accuracy. In this table, we show that our algorithm can reach an approximate solution with reasonable precision faster than SeDuMi.

A disadvantage of our algorithm is that it requires the value of m to be relatively small. When the dimension of this space is large, the constraint generation algorithm requires to add too many constraints before the desired accuracy is reached. Also the CIPM and the Oracle data in Table 4.2 shows that a substantial portion of the CPU time is consumed by the oracle in the random search. Clearly a more efficient search could reduce this time and consequently enhance the performance of our algorithm.

Since SeDuMi is designed to work with sparse matrices, we tried our algorithm on sparse matrices, too. Table 4.3 shows 10 of our experiences. All the columns are the same as in the Table 4.2 except the *density* column which shows how dense is the coefficient

				Duality Gap		CPU Time(s)		
k	$\bar{n}$	%density	cuts	CIPM	SeDuMi	CIPM	Oracle	SeDuMi
2	1E+6	0.2	40	5.60E-03	4.17E-03	81	58	103
		0.1	46	6.03E-03	9.28E-03	27	16	66
		0.05	46	5.78E-03	9.35E-03	35	14	41
4	5E+5	0.2	48	5.88E-03	1.66E-02	33	14	89
		0.1	49	5.90E-03	3.37E-02	20	10	54
		0.05	47	5.87E-03	5.67E0-2	12	8	32
8	1E+5	0.2	51	5.96E-03	1.18E-03	10	5	29
		0.1	48	5.89E-03	2.73E-02	8	2	20
		0.05	49	6.05E-03	1.09E-02	5	2	16
16	5E+4	0.2	52	6.07E-03	1.954E-03	20	4	75
		0.1	50	6.12E-03	1.14E-02	9	2	29
		0.05	52	6.01E-03	2.45E-03	5	2	22
32	1E+4	0.2	55	6.12E-03	1.23E-03	4	0.6	7
		0.1	54	6.10E-03	6.57E-03	3	0.6	5
		0.05	57	6.17E-03	9.63E-03	2	0.2	4
64	5E+3	0.2	55	6.11E-03	6.28E-03	4	1	9
		0.1	52	6.03E-03	5.47E-03	4	0.7	7
		0.05	54	6.10E-03	1.06E-03	3	0.5	5
128	1E+3	0.2	56	6.24E-03	6.11E-03	3	0.3	3
		0.1	59	6.26E-03	4.23E-02	3	0.2	2
		0.05	59	6.30E-03	3.49E-03	5	0.1	2
256	5E+2	0.2	58	6.38E-03	3.42E-02	4	0.2	3
		0.1	59	6.28E-03	1.47E-03	3	0.1	2
		0.05	58	6.54E-03	1.92E-02	3	0.1	1
512	1E+2	0.2	60	6.31E-03	2.84E-03	3	0.1	1
		0.1	60	6.31E-03	2.47E-03	3	0.1	0.8
		0.05	59	6.56E-03	1.89E-02	3	0.1	0.6
1024	5E+1	0.2	62	6.41E-03	1.79E-03	3	0.1	1.5
		0.1	60	6.34E-03	4.75E-03	3	0.1	1
		0.05	60	6.32E-03	4.38E-03	4	0.1	0.7
2048	1E+1	0.2	63	6.42E-03	2.13E-03	4	0.0	0.7
		0.1	63	6.62E-03	2.74E-03	5	0.0	0.5
		0.05	63	6.41E-03	2.21E-03	4	0.0	0.4

Table 4.3: This table compares the CPU time used by SeDuMi and CIPM in solving sparse SOCO problems. We have $m = 100$ and the box constraint $|y_i| \leq 300$ for all the problems. The algorithm process was stopped when the duality gap reached 10^{-3}.

matrix A. For each problem we experimented with three different density levels: %0.2, %0.1, and %0.05. Both algorithms show better results as we increase sparsity. Our constraint generation algorithm exhibits better speed with a small number of large dimensional Lorentz cones. However, the reverse will happen if we go to large number of cones with smaller dimensions, i.e., SeDuMi performs better when a large number of cones are present and the size of the cones are fairly small.

Chapter 5

Conclusions and Future Work

In this thesis we have presented an implementation of our new algorithm to solve a general class of semi-infinite linear optimization problem. We developed a well structured software in MATLAB. Since the implementation does not dependent much on the built-in functions of MATLAB, the implementation can easily be translated to other more efficient languages, such as C or C++. To develop such an implementation is the subject of future work. We can also use our code to solve second order conic linear optimization problem as we reformulate them into semi-infinite linear ones. Our numerical experiences show that this approach is particularly efficient when one has a few second order cones with huge dimensions. Although the complexity of our algorithm is exponential (see [26]), in many practical cases it provides approximate optimal solutions much faster than central path following methods that are using all the constraints at the same time, instead of using only those which are needed. Another advantage of our proposed algorithm is the use of memory. As our numerical experiences show, SeDuMi runs out of memory when it is dealing

with second order cones with huge size. Instead, since we are not putting all the constraints in memory, our algorithm uses much less memory.

Our implementation needs to be improved in the following three parts. The first part is the solution of the linear normal equation system (KKT system). The other part, which is strongly depend on the structure of the given problem, is the oracle. As it is clear from Tables 4.1, 4.2, and 4.3, a high percentage of the CPU time is used by the oracle, specially at the later iterations when very few violated cuts needed to be found. Having a good strategy to find violated constraints would help reducing the time of the search process used by the oracle. Another possibility to enhance the computational efficiency of our algorithm is to use some efficient heuristics, such as Mehrotra's update, to reduce the centering parameter μ to make convergence to optimality much faster.

Appendix A

Duality Theory

A.1 Linear Optimization Duality

Consider the linear optimization problem (1.1) and its dual (1.2). For simplicity, denote primal problem (1.1) by (P) and denote its dual problem (1.2) by (D).

Theorem A.1.1 (Weak Duality)**.** *Let x and s be feasible for (P) and (D), respectively. Then $c^T x - b^T y = x^T s \geq 0$. Consequently, $c^T x$ is an upper bound for the optimal value of (D), if it exists, and $b^T y$ is a lower bound for the optimal value of (P), if it exists. Moreover, if the duality gap $x^T s$ is zero, then x is an optimal solution of (P) and (y, s) is an optimal solution of (D).*

Theorem A.1.2 (Duality Theorem)**.** *If (P) and (D) are feasible then, both problems have optimal solutions. Then, $x \in \mathcal{F}_P$ and $(y, s) \in \mathcal{F}_D$ are optimal solutions if and only if $x^T s = 0$. Otherwise, neither of the two problems has an optimal solution; either both are*

infeasible or one of them are infeasible and the other one is unbounded.

Theorem A.1.3 (Goldman-Tucker Theorem)**.** *If* (P) *and* (D) *are feasible then, there exists a strictly complementary pair of optimal solutions, that is, there is an optimal solution pair* (x, s) *satisfying* $x^T s = 0$ *and* $x + s > 0$.

Proofs can be found in [27].

A.2 Semi-Infinite Linear Optimization Duality Theory

Consider the following semi-infinite linear optimization problem;

$$\inf \left\{ \sum_{i=1}^{\infty} c_i x_i \ : \ \sum_{i=1}^{\infty} x_i a_i = b, \ x_i \geq 0 \right\}, \qquad (P_\infty)$$

where $c_i \in \mathbb{R}$, $a_i \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, and $x_i > 0$ for only a finite number of indices i, i.e., all the sums have only a finite number of non zeros components.

Considering $T = \{ 1, 2, 3, \dots \}$, the linear optimization problem (1.20) is the dual form of the problem (P_∞) which is

$$\sup \left\{ b^T y \ : \ a_i^T y \leq c_i, \ i = 1, 2, 3, \dots \right\} \qquad (D_\infty)$$

Recall that a program is called *consistent* if is has a feasible solution. In general, and without any regularization technics, there is no reason that all such problems as (P_∞) can be discritized in order to be solved iteratively, or the optimal value of (P_∞) and (D_∞) coincide.

Example A.2.1. Consider the SILP problem

$$\inf \left\{ y_1 \ : \ ty_1 + (1-t)y_2 \geq t - t^2, \ t \in (0,1) \right\}. \tag{A.1}$$

For any fixed $t \in (0,1)$ the optimal value of the SILP problem given by (A.1) is $-\infty$, but the optimal value of problem (A.1) is 0, (see [12], Example 1.1). Thus, some conditions are needed to ensure that problem (P_∞) can be solved by discretization, or by constraint generation technics.

Theorem A.2.2 (Weak Duality)**.** *If* (P_∞) *and* (D_∞) *are both consistent, then the optimum value of* (P_∞) *is grater than the optimum value of* (D_∞)*, and both values are finite.*

Theorem A.2.3 (Complementary Slackness)**.** *If* $x = (x_i)_{i=1}^{\infty}$ *is feasible for* (P_∞), $y \in \mathbb{R}^m$ *is feasible for* (D_∞) *and*

$$\sum_{i=1}^{\infty} x_i(c_i - a_i^T y) = 0, \tag{A.2}$$

then x *is optimal for* (P_∞) *and* y *is optimal for* (D_∞).

Having formulated the primal and dual problems, we now state conditions which will ensure that their optimal values coincide. First we need to define two sets:

$$M_A = \left\{ \sum_{i=1}^{\infty} x_t a_i \ : \ x \geq 0 \right\},$$
$$M_B = \left\{ \left(\sum_{i=1}^{\infty} x_i a_i, \sum_{i=1}^{\infty} x_i c_i \right) \ : \ x_i \geq 0, \ i = 1, 2, 3, \ldots \right\}.$$

Note that M_A and M_B are *cones* in $\mathbb{R}^m$ and $\mathbb{R}^{m+1}$, respectively.

Theorem A.2.4. *If the optimal value of* (D_∞) *is finite and* M_B *is closed, then the optimal values of* (D_∞) *and* (P_∞) *coincide.*

The proofs of the theorems can be found in [1].

Example A.2.5. Considering the Example of Karney (see Example 1.3.1), M_A is the cone generated by

$$\{\,(-1,0)\,,\ (0,1)\,,\ (-1,1/3)\,,\ (-1,1/4)\,,\ \ldots\,\},$$

and M_B is the cone generated by

$$\{\,(-1,0,1)\,,\ (0,1,0)\,,\ (-1,1/3,0)\,,\ (-1,1/4,0)\,,\ \ldots\,\}.$$

Note that the vector $b = (-1,0)^T$ in this example is on the boundary, not in the interior of M_A, and M_B is not a closed cone. Hence, we cannot expect to have zero duality gap in this example.

Bibliography

[1] E.J. Anderson and P. Nash. *Linear Programming in Infinite-Dimensional Space, Theory and Applications*. John Wiley and Sons, 1987.

[2] E.J. Anderson and A.B. Philpott. *Infinite Programming*. Springer Verlag, Berlin, 1985.

[3] D.S. Atkinson and P.M. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, series B, 69:1–43, 1995.

[4] O. Bahn, O. du Merle, J.-L. Goffin, and J.-Ph. Vial. Cutting plane method from analytic centers for stochastic programming. *Mathematical Programming*, series B, 69:45–73, 1995.

[5] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming*. John Wiley & Sons, New York, second edition, 1993.

[6] I.D. Coop and G.A. Watson. A projected Lagrangian algorithm for semi-infinite programming. *Mathematical Programming*, 32:337–356, 1985.

[7] D. den Hertog. *Interior Point Approach to Linear, Quadratic and Convex Programming -Algorithms and Complexity-*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.

[8] A.V. Fiacco and K.O. Kortanek. *Semi-Infinite Programming and Applications*. Lecture Notes in Economics and Mathematical Systems, Springer Verlag, 215, 1983.

[9] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley and Sons, New York, 1968.

[10] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin, and M.H. Wright. On projected newton barrier methods for linear programming and an equivalence to Karmarkars projective method. *Mathematical Programming*, 36:183209, 1986.

[11] M.A. Goberna and M.A. López. *Semi-Infinite Programming, Recent Advances*. Kluwer A.P.C., 2001.

[12] M.A. Goberna and M.A. López. *Linear Semi-Infinite Optimization*. John Wiley and Sons, Chichester, 1998.

[13] J.-L. Goffin and F. Sharifi Mokhtarian. Using the primal dual infeasible Newton method in the analytic center method for problems defined by deep cutting planes. *Journal of Optimization Theory and Applications*, 101:35–58, 1999.

[14] J.-L. Goffin and J.-Ph. Vial. Multiple cuts in the analytic center cutting plane method. *SIAM Journal on Optimization*, 11:266–288, 2000.

[15] J.-L. Goffin and J.-Ph. Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84:89–103, 1999.

[16] J.-L. Goffin, A. Haurie, and J.-Ph. Vial. Decomposition and non-differentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.

[17] J.-L. Goffin, Z.-Q. Luo, and Y. Ye. Complexity analysis of an interior cutting plane for convex feasibility problems. *SIAM Journal on Optimization*, 6:638–652, 1996.

[18] J.-L. Goffin, J. Gondzio, R. Sarkissian, and J.-Ph. Vial. Solving nonlinear multicommodity flow problems by the analytic center cutting plane method. *Mathematical Programming*, series B, 76 1:131–154, 1997.

[19] J. Gondzio, O. du Merle, R. Sarkissian, and J.-Ph. Vial. ACCPM - A library for convex optimization based on an analytic center cutting plane method. *European Journal of Operational Research*, 94:206–211, 1996.

[20] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[21] F.A. Lootsma. Hessian matrices of penalty functions for solving constrained optimization problems. *Philips Resarch Reports*, 24:322–330, 1969.

[22] Z.-Q Luo, C. Roos, and T. Terlaky. Complexity analysis of logarithmic barrier decomposition methods for semi-infinite linear programming. *Applied Numerical Mathematics*, 29:379–394, 1999.

[23] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 1999.

[24] M.R. Oskoorouchi and J.L. Goffin. A matrix generation approach for eigenvalue optimization. *Mathematical Programming*, 109:155–179, 2007.

[25] M.R. Oskoorouchi and J.E. Mitchell. A second-order cone cutting surface method: Complexity and applications. *Computational Optimization and Applications*, to appear, 2008.

[26] M.R. Oskoorouchi, H.R. Ghaffari, and T. Terlaky. An interior point constraint generation method for semi-infinite linear programming. Technical report, Advanced Optimization Laboratory, McMaster University, Hamilton, Ontario, July 2008.

[27] C. Roos, T. Terlaky, and J.-Ph. Vial. *Interior Point Methods for Linear Optimization*. John Wiley & Sons, second edition, 1997.

[28] S. Wright. Modified Cholesky factorizations in interior-point algorithms for linear programming. *SIAM Journal on Optimization*, 9:1159–1191, 1999.

[29] Y. Ye. Potential reduction algorithm allowing column generation. *SIAM Journal on Optimization*, 2:7–20, 1992.

Index