IMPLEMENTING THE NEW SELF-REGULAR PROXIMITY BASED IPMS

IMPLEMENTING THE NEW SELF-REGULAR PROXIMITY BASED IPMS

By

Xiaohang Zhu

A thesis

submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Dregree
Master of Science

McMaster University

MASTER OF SCIENCE (2002) McMaster university

COMPUTING & SOFTWARE Hamilton Ontario

TITLE: Implementing the New Self-Regular Proximity Based IPMs

AUTHOR: Xiaohang Zhu

SUPERVISOR: Dr. Tamás Terlaky

NUMBER OF PAGES: xvi, 134

Abstract

We present our experiences with an implementation of some new self-regular proximity based interior point methods for linear optimization. After a brief review of the underlying algorithm, various issues with respect to implementation are addressed.

The software package McIPM uses MATLAB, LIPSOL's preprocessing, and WSMP as a sparse matrix package. This project utilized optimization theory, interior point methods, numerical analysis, sparse matrix techniques, multi-thread programming, data synchronization, messages passing, and shared memory.

Extensive testing proves that the McIPM software package is competitive with state of the art software packages, such as LIPSOL, and self-regular proximity approach offers avenues for improvement when solving difficult problems.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Dr. Tamás Terlaky, for introducing me to the wonderful areas of linear optimization, interior point methods, and optimization software development, and for his thoughtful guidance and many invaluable and inspirational discussions throughout the research leading to this thesis.

In the Advanced Optimization Lab (AdvOL), I especially want to thank Dr. Jiming Peng for his many insightful comments based on his research, Dr. Guoqing Zhang for all the valuable discussions and cooperation during the implementation, and my AdvOL lab mates for all the simulating discussions and the pleasant working environment they provided.

Hearty thanks go to Dr. Jeff Zucker for his careful reading of my thesis, and Dr. Sanzheng Qiao, Dave Gilbert, and Pusheng Song for helpful discussions.

I am indebted to Dr. Erling D. Andersen and Prof. Yin Zhang for useful

discussions based on their expertise.

Thanks are due to NSERC (the Natural Sciences and Engineering Research Council of Canada) for the PGS Scholarship, and to the Government of Ontario and McMaster University for the OGSST Scholarship, both of which supported my graduate study, and also to IBM for providing WSMP package and for their contribution to the computational power of AdvOL.

My special thanks goes to also my father Maosen Feng and my mother Lingzhi Zhu for everything they gave me. I would like to thank my daughter Jenny and my son Allen for the happiness they bring me. And last, but of course not least, I would like to thank my husband Xue-Feng Yang for his love, understanding, encouragement, and support.

Contents

List of Notations

A^T : transpose of the matrix A

e : all 1's vector

$\mathcal{I}, \mathcal{J}$: index set

L : the input length for an LO problem

μ : the barrier parameter or centering parameter

$\mathcal{R}^n$: n-dimension Euclidian vector space

$\mathcal{R}^n_+(\mathcal{R}^n_{++})$: nonnegative (positive) orthant in $\mathcal{R}^n$

x, s : the vector of primal variables

y, w, z : the vector of dual variables

x_i : the i-th coordinate of the vector x

x^T : transpose of x

xz : $= [x_1z_1, \quad x_2z_2, \quad \ldots, \quad x_nz_n]^T$, coordinate-wise product of x and z

$\Delta x, \Delta s$: the primal part of the search direction

$\Delta y, \Delta w, \Delta z$: the dual part of the search direction

d_x, ds : the primal part of the search direction in the scaled v-space

v : the scaled vector defined by $v_i = \sqrt{\frac{x_i z_i}{\mu}},\ \forall i \in \mathcal{I}$

α : step length

ϵ : accuracy parameter

τ : the homogenizing variable

κ : the artificial variable in the homogenous model

θ : updating factor for μ

$\psi(t)$: a function $\mathcal{R} \to \mathcal{R}$ defining the proximity

$\psi(x)$: $\psi(x) = (\psi(x_1), \ldots, \psi(x_n))^T$ a mapping from $\mathcal{R}^n$ into itself

$\mathcal{F}_P, \mathcal{F}_D$: the set of primal or dual feasible solutions

$\mathcal{F}_{SP}$: the set of feasible solutions of problem (SP)

$\Psi(x)$: the summation $\Psi(x) = \sum_{i=1}^{N} \psi(x_i)$

$\mathcal{N}(\mu, \tau)$: $=\{(x, z) : \Psi(x, z, \mu) < \delta\}$

X : the diagonal matrix whose diagonal entries x_i are the components of x

δ : proximity parameter

List of Abbreviation

CSC-LT : the lower triangle part in the compressed sparse column format
: used to store sparse matrix

IPMs : Interior Point Methods

SR-IPMs : Self-Regular based Interior Point Methods

LIPSOL : Linear-programming Interior Point SOLvers by Yin Zhang

LO : Linear Optimization

McIPM : McMaster Interior Point Methods by Xiaohang (Lois) Zhu

MSR/MSC : modified compressed sparse row/column used to store sparse matrix

NLCP : Nonlinear Complementarity Problem

QO : Quadratic Optimization

SDO : Semi-Definite Optimization

SOCO : Second-Order Conic Optimization

List of Figures

List of Tables

Preface

The present thesis is the result of my two years of M.Sc. study[1]. The thesis reviews basic knowledge of Linear Optimization (LO)[2] and Interior Point Methods (IPMs), and describes the details of the implementation of new self-regular proximity based IPMs.

LO problems have numerous important applications in operations research and in many other areas of science and engineering. These applications of LO in operations research include transportation planning, portfolio selection, production analysis, and so on [12]. In practice, LO problems are usually very large in terms of the numbers of variables and constraints. Hence, if larger optimization problems can be solved, then better decisions can be made. Therefore it is important to develop new tools to solve large-scale optimization problems.

The first efficient method for LO, the simplex method [4], was proposed in the forties (soon after World War II, motivated by military applications) and, although it has performed very well in practice, it runs in exponential time in the worst case [17]. The first polynomial-time algorithm, the ellipsoid algorithm [14], was discovered by Khachian at the end of the seventies. Karmarkar's projective algorithm [13] in the mid-eighties induced highly active research resulting in the development of numerous variations of IPMs.

Over the past ten years IPMs for LO have gained extraordinary interest

[1]Research supported by an NSERC PGS-A Scholarship and OGSST Scholarship.

[2]The name "linear programming" was used in the early stage of the field of linear optimization. Here we are following [23] to use the more natural name "linear optimization". This terminology also distinguishes the field from general programming works related to computers.

as an alternative to simplex based methods. This is particularly important in those applications that require the solution of very large LO problems. Current implementations of IPMs are sophisticated optimization software tools, capable of solving very large LO problems.

A new framework for the theory of primal-dual IPMs was introduced by Peng, Roos and Terlaky [22] in the year 2000. This theory introduces the notation of Self-Regular (SR) proximities and presents a unified novel approach for solving LO and conic LO problems. Some appealing features of the new theory are:

- Self-regular based IPMs methods allow us to design and analyze large-update and large neighborhood IPMs for large classes of optimization problems;

- IPMs based on the new framework are closer to the IPMs that have been successfully implemented in more efficient software packages;

- The theoretical worst case complexity of the new large-update IPMs is better than that of the existing large-update IPMs, while the new small-update IPMs have the same iteration bound as the best known iteration bound for existing small-update, small-neighborhood IPMs.

A software package, the McMaster Interior Point Method (McIPM), has been developed. The implementation is based on new SR-search directions and the homogeneous self-dual model [23]. This approach permits either identifying an optimal solution of the problem or detecting the infeasibility of the primal or dual problems. To enhance the practical performance of the algorithm, we also employ the predictor-corrector strategy [19] to update the barrier parameter and to obtain a second-order approximation of the central path.

Our McIPM implementation of IPMs is comparable to state-of-the-art research and commercial software. McIPM uses various numerical and computational techniques, such as sparse matrix techniques, shared memory, and so on. All of the implementation issues are described in the thesis.

Finally, our McIPM implementation is tested on the full NETLIB [6] test suite and we present bench marking computational results on this standard test set.

Chapter 1

Preliminaries

In this section, we recall the notations and basic theory of linear optimization problems.

1.1 The LO Problem

Linear optimization (LO) is the problem of optimizing a linear objective function of several decision variables subject to linear equality or inequality constraints. The standard LO problem can be described as

$$\begin{array}{lll} & \min & c^T x \\ (P) & \text{s.t.} & Ax = b, \\ & & x \geq 0, \end{array}$$

where $x, c \in \mathcal{R}^n,\ b \in \mathcal{R}^m,\ A \in \mathcal{R}^{m \times n}$ with $\text{rank}(A) = m,\ m \leq n$. The label (P) stands for the primal problem.

The linear function $c^T x$ is called the *objective function*, and x is the vector of *decision variables*. Further, $Ax = b,\ x \geq 0$ are *constraints* imposed on the selection of x, and matrix A is called the *constraint matrix*.

Now we review some basic notations and concepts for LO problems in this standard form.

Set of primal feasible solutions: $\mathcal{F}_P = \{x \mid Ax = b,\ x \geq 0\}$.

Primal feasible solution: A vector x is called a *primal feasible solution* if $x \in \mathcal{F}_P$.

Basic feasible solution: A primal feasible solution $x \in \mathcal{F}_P$ is called a *basic feasible solution* if it is a feasible solution and the columns of A corresponding to the nonzero components of x are linearly independent.

Primal optimal solution: A primal feasible solution x^* is called *a primal optimal solution* if $c^T x^* \leq c^T x$ for all $x \in \mathcal{F}_P$.

Primal unbounded: If there is a sequence $\{x^k\}_{k=1}^{\infty}$ such that $x^k \in \mathcal{F}_P$ and $c^T x^k \to -\infty$ (as $k \to \infty$) then (P) is said to be *unbounded*.

Primal infeasible: If $\mathcal{F}_P = \emptyset$ then we call the primal problem *infeasible*.

Each LO problem, what we usually refer to as the *primal problem* (P), is associated with another LO problem, called the *dual problem*. The dual problem (D) can be given as follows:

$$\begin{array}{lllll} & \max & b^T y & & \\ (D) & \text{s.t.} & A^T y & +z & = c, \\ & & & z & \geq 0, \end{array}$$

where $y \in \mathcal{R}^m,\ z \in \mathcal{R}^n$. The vectors y and z are called *dual variables* and *dual slacks*, respectively. The dual problem is constructed from the same data as the primal problem, but uses different variables and relations. Note that there is a dual variable y_i associated with each constraint in the primal problem and a dual slack z_j associated with each variable x_j in the primal problem. We see that (D) is also an LO problem where y is a "free" vector. Analogous to the primal problem (P), we have the following terminology for (D).

Set of dual feasible solutions: $\mathcal{F}_D = \{(y, z) \mid A^T y + z = c,\ z \geq 0\}$.

Dual feasible solution: A pair of vectors (y, z) is called a *dual feasible solution* if $(y, z) \in \mathcal{F}_D$.

Primal-dual feasible solution: A triple (x, y, z) is said to be a *primal-dual feasible solution* if $x \in \mathcal{F}_P$ and $(y, z) \in \mathcal{F}_D$.

Strictly primal-dual feasible solution: A primal-dual feasible solution (x, y, z) is said to be a *strictly primal-dual feasible solution* if $x > 0$ and $z > 0$.

Dual optimal solution: A feasible solution (y^*, z^*) is called *a dual optimal solution* if $b^T y^* \geq b^T y$ for all $(y, z) \in \mathcal{F}_D$.

Dual unbounded: If there is a sequence $\{y^k, z^k\}_{k=1}^{\infty} \in \mathcal{F}_D$ such that $b^T y^k \to +\infty$, then problem (D) is said to be *unbounded*.

Dual Infeasible: If $\mathcal{F}_D = \emptyset$, then we call the dual problem *infeasible*.

1.1.1 Duality Results

Duality theory studies the relationship between the primal problem (P) and its dual (D). This is the most important concept in LO. Duality provides an easy way to verify the optimality of solutions and is the foundation for algorithm design. The following propositions demonstrate important relations between the two problems (P) and (D).

Proposition 1.1.1. [Weak Duality Theorem] *Let $\mathcal{F}_P$ and $\mathcal{F}_D$ be nonempty. Then for any $x \in \mathcal{F}_P$, $(y, z) \in \mathcal{F}_D$*

$$c^T x \geq b^T y,$$

where equality holds if and only if the complementarity condition $x_i z_i = 0$ holds for all $i = 1, \cdots, n$.

Proof. The constraints of the primal and dual problems imply

$$c^T x - b^T y = x^T c - x^T A^T y = x^T (c - A^T y) = x^T z \geq 0.$$

Equality in the last inequality implies the complementarity condition. □

This proposition shows that if both the primal and dual problems are feasible, then both problems are bounded. The difference $c^T x - b^T y$ is called the *duality gap*. There are several simple consequences of the weak duality theorem.

Corollary 1.1.1. *If the primal problem (P) is unbounded, then the dual problem (D) is infeasible. If the dual is unbounded, then the primal is infeasible.*

Corollary 1.1.2. *If $x \in \mathcal{F}_P$, $(y, z) \in \mathcal{F}_D$, and $c^T x = b^T y$, i.e. $x_i z_i = 0$ for all $i = 1, \cdots, n$ (Proposition 1.1.1), then x and y are optimal with respect to (P) and (D), respectively.*

Now if a primal-dual feasible solution (x, y, z) has zero duality gap, then x must be a primal optimal solution because $c^T x$ is equal to the lower bound $b^T y$.

A natural question to ask is if there always exists a feasible primal-dual pair having zero duality gap. To answer this question, we need Farkas' Lemma.

Proposition 1.1.2. [Farkas' Lemma [5], primal form] *Exactly one of the following two alternatives holds:*

i) *There exists an x such that $Ax = b,\ x \geq 0$.*

ii) *There exists a y such that $A^T y \leq 0$ and $b^T y > 0$.*

Proof. Assume that i) holds. Then for any y satisfying $A^T y \leq 0$, we have

$$b^T y = x^T A^T y \leq 0$$

because $x \geq 0$. Thus ii) is false.

In the rest of the proof, we assume that i) is false, that is, $Ax = b,\ x \geq 0$ has no solution. The goal is to prove that ii) holds.

Let $\mathcal{K} = \{Ax : x \geq 0\} \subset \mathcal{R}^m$. The set $\mathcal{K}$ is a closed convex cone in $\mathcal{R}^m$. Since $Ax = b,\ x \geq 0$ has no solution, b does not belong to $\mathcal{K}$. Let p_b be the point that minimizes the Euclidean distance function $\|b - \hat{y}\|$, $\hat{y} \in \mathcal{K}$. For any $\hat{y} \in \mathcal{K}$ and $\lambda \in [0,\ \ 1]$, since $\mathcal{K}$ is convex, we have

$$\lambda \hat{y} + (1 - \lambda) p_b \in \mathcal{K}.$$

From the definition of p_b, we obtain

$$\begin{aligned} 0 &\leq \|b - (\lambda\hat{y} + (1-\lambda)p_b)\|^2 - \|b - p_b\|^2 \\ &= \|(b - p_b) - \lambda(\hat{y} - p_b)\|^2 - \|b - p_b\|^2 \\ &= -2\lambda(\hat{y} - p_b)^T(b - p_b) + \lambda^2\|\hat{y} - p_b\|^2, \quad \lambda \in [0,1]. \end{aligned} \tag{1.1.1}$$

Hence

$$(\hat{y} - p_b)^T(b - p_b) \leq \frac{\lambda}{2}\|\hat{y} - p_b\|^2, \quad \lambda \in [0,1].$$

Since $\lambda \in [0,1]$ is arbitrary, so

$$(\hat{y} - p_b)^T(b - p_b) \leq 0, \quad \hat{y} \in \mathcal{K}. \tag{1.1.2}$$

Because $\mathcal{K}$ is a closed convex cone, there exist $\hat{w} \geq 0$ and $x \geq 0$ such that $A\hat{w} = p_b$ and $\hat{y} = Ax$. This implies

$$(Ax - A\hat{w})^T\bar{y} \leq 0, \quad x \geq 0, \tag{1.1.3}$$

where $\bar{y} = b - p_b$.

Set $x = \hat{w} + e_i \geq 0$, where $e_i = [0, 0, \ldots, 1, \ldots, 0]^T$, the relation $e_i^T A^T\bar{y} = (\hat{w} + e_i - \hat{w})^T A^T\bar{y} \leq 0$ holds. i.e., the i-th component of $A^T\bar{y}$, $(A^T\bar{y})_i \leq 0$. Since this is true for all i, therefore $A^T\bar{y} \leq 0$.

On the other hand, since $0 \in \mathcal{K}$, so from (1.1.2) we have $p_b^T\bar{y} \geq 0$. Since $b \notin \mathcal{K}$, then $\bar{y} = b - p_b \neq 0$. Hence $\bar{y}^T b = \bar{y}^T(p_b + \bar{y}) = \bar{y}^T p_b + \bar{y}^T\bar{y} > 0$. □

Remark 1.1.1. *The Farkas Lemma is also called the Farkas Theorem of Alternatives, which was published in 1894 [5]. It is a fundamental result in LO.*

Let us look at the second condition in the Farkas Lemma. For any column vector a_j, $j = 1, 2, \cdots, n$ of A, either a_j is on the hyper-plane $y^T\varrho = 0$ (i.e., $y^T a_j = 0$), or a_j is strictly on one side of the hyper-plane (i.e., $y^T a_j < 0$). Moreover, b is on the other side of the hyper-plane $y^T\varrho = 0$. In other words, the hyper-plane $y^T\varrho = 0$ separates b on one side and all column vectors of A on the other side. On the other hand, condition one says that the vector b is in the convex hull of the vectors a_j, $j = 1, 2, \cdots, n$. In summary, the vector b either in the convex cone $\mathcal{K}$ or separated from $\mathcal{K}$ by a hyper-plane.

We present the dual form of Farkas's Lemma as well. Its proof is analogous to the primal one.

Proposition 1.1.3. [Farkas' Lemma, dual form] *Exactly one of the following two alternatives holds:*

i) *There exists an x such that $c^T x < 0, Ax = 0,\ x \geq 0$.*

ii) *There exists a y such that $A^T y \leq c$.*

Let us continue the discussion on the primal and dual problems.

Corollary 1.1.3. *If (D) is infeasible, then (P) is either infeasible or unbounded.*

Proof. If (D) is infeasible, then there does not exist any y such that

$$A^T y \leq c.$$

Then from the dual form of the Farkas Lemma, there exists an x such that

$$c^T x < 0, Ax = 0,\ x \geq 0.$$

If (P) is feasible then there exists an x^0 such that

$$Ax^0 = b, \quad x^0 \geq 0.$$

Therefore, we have $A(x+x^0) = b,\ \ (x+x^0) \geq 0$, which implies $(x+x^0) \in \mathcal{F}_P$. Clearly, for any $k > 0$, we have

$$\begin{aligned} A(kx + x^0) &= b, \\ kx + x^0 &\geq 0, \end{aligned} \tag{1.1.4}$$

so that $(kx + x^0) \in \mathcal{F}_P$. Moreover,

$$\lim_{k \to +\infty} c^T(kx + x^0) = -\infty$$

implying that (P) is unbounded.
The proof is complete. □

From the above discussions we know that if (P) has an optimal solution, then (P) is feasible and bounded, which implies that (D) is feasible. Correspondingly, if (P) is unbounded, then (D) must be infeasible. The reason is that if (P) is unbounded and (D) is feasible, then there exists a sequence of points so that the primal objective values at those points go to negative infinity, while all these values are upper bounds for the dual at the same time, thus an upper bound of $b^T y$ for any $y \in \mathcal{F}_D$ leading to a contradiction. Similarly, if (D) is unbounded, then (P) is infeasible. Hence, we have the following important result.

Theorem 1.1.1 (Strong Duality Theorem [23]). *Consider a pair of primal and dual problems (P) and (D). If one of the problems has an optimal solution then so does the other, and the optimal objective values are equal.*

Proof. Assume without loss generality (by duality) that (P) has an optimal solution. Let

$$\begin{array}{rl} \zeta_P & = \min\{c^T x \ : \ Ax = b, x \geq 0\}, \\ \zeta_D & = \max\{b^T y \ : \ A^T y \leq c\}. \end{array} \tag{1.1.5}$$

By the *Weak Duality Theorem* 1.1.1, $\zeta_P \geq \zeta_D$. We only need to show that $\zeta_D \geq \zeta_P$.

For any $\zeta < \zeta_P$, by definition of ζ_P in (1.1.5) we see that the system

$$\begin{array}{rl} c^T x & \leq \zeta, \\ Ax & = b, \\ x & \geq 0 \end{array} \tag{1.1.6}$$

has no solution. Note that the first inequality of (1.1.6) can be written as $-c^T \geq -\zeta$, so the system (1.1.6) is equivalent to

$$\begin{array}{rl} -c^T x - \rho & = -\zeta, \\ Ax & = b, \\ x, \quad \rho & \geq 0. \end{array} \tag{1.1.7}$$

Since (1.1.7) is infeasible, it follows from the Farkas Lemma there is a $(\lambda, y)^T$ satisfying

$$\begin{pmatrix} -c^T & -1 \\ A & 0 \end{pmatrix}^T \begin{pmatrix} \lambda \\ y \end{pmatrix} \leq 0, \quad \text{and} \quad b^T y - \lambda\zeta > 0,$$

i.e.,

$$\begin{aligned} A^T y - \lambda c &\leq 0, \\ b^T y - \lambda\zeta &> 0, \\ -\lambda &\leq 0. \end{aligned} \tag{1.1.8}$$

Next we show $\lambda \neq 0$. If $\lambda = 0$, from (1.1.8) we get:

$$A^T y \leq 0, \quad b^T y > 0. \tag{1.1.9}$$

By (1.1.9) and the Farkas Lemma we infer that (P) is infeasible, which contradicts the assumption. Therefore, we must have $\lambda \neq 0$. This along with (1.1.8) implies that $\lambda > 0$. Now, dividing the second inequality of (1.1.8) by λ, we get:

$$b^T \frac{y}{\lambda} > \zeta.$$

Let $y := y/\lambda$, then from the second equation of (1.1.8) and (1.1.5) it follows that

$$\zeta < b^T y \leq \zeta_D \leq \zeta_P.$$

Therefore

$$\zeta_D \geq \lim_{\zeta \to \zeta_P} b^T y = \zeta_P.$$

The proof is complete. □

In summary, if both (P) and (D) are feasible, then both problems have optimal solutions and the optimal objective values are equal, that is, there is no duality gap at optimality. If one of (P) and (D) has no feasible solutions, then the other is either unbounded or has no feasible solution. If one of (P) and (D) is unbounded then the other is infeasible.

Using the Strong Duality Theorem, finding optimal solutions to (P) and (D) is equivalent to solving the following primal-dual system, which is referred to as the *optimal conditions*.

$$\begin{aligned} Ax &= b, \quad x \geq 0 \\ A^T y + z &= c, \quad z \geq 0 \\ x_j z_j &= 0, \quad j = 1, \ldots, n. \end{aligned} \tag{1.1.10}$$

Here the first line represents primal feasibility, the second line is dual feasibility, and the last one is the *complementarity* condition. These together guarantee that the primal and dual objective values are equal: $c^T x = b^T y$.

1.1.2 Strict Complementarity

The optimality (strict) conditions (1.1.10) can be written as

$$\begin{aligned} Ax &= b, \quad x \geq 0 \\ A^T y + z &= c, \quad z \geq 0 \\ xz &= 0, \end{aligned} \tag{1.1.11}$$

where $xz = [x_1 z_1, \quad x_2 z_2, \quad \ldots, \quad x_n z_n]^T$ denotes the coordinate-wise product of the vectors x and z.

Definition 1.1.1. *A feasible primal-dual pair (x, y, z) satisfying the conditions*

$$x_j z_j = 0 \text{ and } x_j + z_j > 0, \quad j = 1, \ldots, n$$

is said to be a strictly complementary optimal solution.

The following theorem proves the existence of strictly complementary optimal solutions for LO.

Theorem 1.1.2. [Goldman-Tucker Theorem [7]] *If (P) and (D) are feasible, then there exists a strictly complementary pair of optimal solutions.*

The complementarity conditions state that both $x_j > 0$ and $z_j > 0$ cannot hold at the same time for any j at an optimal solution. However, there may exist an optimal primal-dual pair such that both $x_j = z_j = 0$ for some $1 \leq j \leq n$. On the other hand, by the Goldman-Tucker Theorem we know that a strictly complementary optimal solution exists. This means that a given LO problem may have multiple primal-dual optimal solutions. Some are strictly complementary and others are not. The following trivial example illustrates this property.

Example 1.1.1. *Consider*

$$\min\ x_1 \quad s.t. \quad x_1 + x_2 + x_3 = 1, \quad x \geq 0,$$

whose dual is

$$\max\ y \quad s.t. \quad \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} y + z = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, z \geq 0$$

For any $t \in [0, 1]$, the solutions

$$x^*(t) = (0, t, 1-t)^T, \quad y^* = 0, \quad z^* = (1, 0, 0)^T$$

are optimal. It is also clear that for $t \in (0, 1)$, (x^*, y^*, z^*) is a strictly complementary solution. If $t = 0$ or $t = 1$, the primal-dual solutions are not strictly complementary.

1.2 From simplex Methods to IPMs

From the geometric point of view $\mathcal{F}_P$ (and $\mathcal{F}_D$) is a polyhedron, and at least one optimal solution of an LO problem is at a vertex of the polyhedron. There are two major classes of methods for solving LO problems. One is the class of simplex methods, and the other is the class of IPMs. In order to discuss the class of simplex methods, let us recall the following LO fundamental theorem.

Proposition 1.2.1. [Fundamental theorem of LO] *Given (P) and (D)*

- *if there is a feasible solution, then there is a basic feasible solution.*
- *if there is an optimal solution, then there is an basic optimal solution.*

The simplex method [4], the first practical method for LO, starts at a basic solution, at a vertex of the feasible region. At each iteration it moves to another adjacent basis (vertex), and proceeds this way improving (or leaving unchanged) the objective function value, until it reaches an optimal basis (vertex) corresponding to an optimal LO solution. Thus the simplex method travels along edges of the polyhedron until it hits an optimal vertex. It always keeps the primal equality (i.e. $Ax = b$), dual equality (i.e. $A^T y + z = c$), and complementarity conditions satisfied (i.e. $xz = 0$). Primal feasibility is preserved while nonnegativity of the dual variables is reached only at optimality. LO problems are solved in two phases since the simplex method requires a feasible basic solution to start with.

Obviously the ideal situation is that when moving from one vertex to another, the objective function value is improved by a significant amount. The worst case is that the objective function value remains unchanged when moving from one basis to another. As a result, simplex methods may cycle for degenerate LO problems[1]. Until approximately 1984 all LO problems were solved by the simplex method. Today it still remains one of the most efficient methods for solving the great majority of practical problems. However, at the time of writing, no polynomial time version of the simplex method is known. The complexity of most simplex methods (in the worst case) is $O(2^n)$ where n is the number of variables.

The first polynomial-time algorithm for LO is the ellipsoid method which was proposed by Khachian in 1979 [14]. The algorithm has a polynomial $O(n^2L)$ iteration complexity with a total of $O(n^4L)$ bit operations, where

[1]The primal-dual pair of LO problems (P) and (D) is called *primal degenerate* [9] if there exists a primal feasible x with less than m positive coordinates, and it is called *dual degenerate* if there exists a dual feasible z with less than $n - m$ positive coordinates. The pair (x, z) is called *degenerate* if it is primal or dual degenerate.

L is the size of the problem (number of bits to encode A, b, and c on a binary tape). Unfortunately, the ellipsoid method's performance for practical problems is poor compared with simplex methods.

A much more efficient algorithm, that also enjoys a polynomial $O(nL)$ iterations complexity with a total $O(n^{3.5}L)$ bit operations, was found by Karmarkar (1984). It has the further advantages of being highly efficient in practice and outperforms the simplex method for large-scale sparse problems [18]. This method goes through the middle of the polyhedron (making it a so-called interior point method). Arguably, IPMs were known as early as the 1950s in the form of barrier function methods, but Karmarkar's completely results and efficiency claims [13] led to great attention. Karmarkar's paper generated a lot of interest because the new IPMs have excellent (theoretical) convergence properties, and superior practical performance. The name "IPM" arises from the fact that these methods move through the interior of the feasible region towards an optimal solution. This is in contrast with simplex methods that follow a sequence of adjacent extreme points to an optimal solution. The path of extreme points may contain an exponentially large number of points, therefore moving through the interior is advantageous and the problem with a large number of extreme points is avoided.

Chapter 2

Interior-Point Methods

Among the different interior point algorithms, the primal-dual path following algorithm has gained reputation of being the most efficient method for practical LO problems.

2.1 Primal-Dual IPMs

Most of the IPMs can be seen as variants of the logarithmic barrier method. The principle of the logarithmic barrier method is to solve (P) by approximating the problem by a series of nonlinear problems, called *barrier problems*.

Definition 2.1.1. *The problems (P) and (D) satisfy the interior point condition (IPC) if there is a strictly primal-dual feasible solution.*

Assume the IPC condition holds, then we can remove the nonnegativity constraints by augmenting the objective function by a *logarithmic barrier function*:

$$(\mathrm{P}_\mu) \qquad \begin{aligned} \min \quad & c^T x - \mu \sum_{j=1}^{n} \log x_j \\ & Ax = b, \\ & x > 0, \end{aligned} \tag{2.1.1}$$

where $\mu > 0$ is a constant. The logarithmic barrier function forces the solution of the above perturbed problem away from the boundary of the feasible region. For each μ, the minimum of (P_μ) is attained at an interior point. So we can apply calculus to analyze the minimizers $x(\mu)$. Problems (P) and (P_μ) are closely related. As μ goes to zero, the optimal solution for (P_μ) converges to an optimal solution for (P).

Viewed as a function of μ, the set of optimal solutions to the barrier problems forms a path through the interior of $\mathcal{F}_P$. This path is called the *central path of (P).*

Problem (P_μ) is also associated with another concept, the so called "analytic center", when $\mu \to +\infty$.

Definition 2.1.2. *The solution of the convex optimization problem*

$$(P_\infty) \qquad \begin{aligned} \min \quad & \mu \sum_{j=1}^{n} \log x_j \\ & Ax = b, \\ & x > 0, \end{aligned} \tag{2.1.2}$$

is called the analytic center of the feasible region $\mathcal{F}_P$.

The Karush-Kuhn-Tucker (KKT) condition [20] for problem (2.1.1) is

$$\begin{aligned} A^T y + \mu X^{-1} e &= c, \\ Ax &= b, \quad x > 0, \end{aligned} \tag{2.1.3}$$

where X denotes the diagonal matrix whose diagonal entries are the components of x, and e denotes the vector of all 1's. Denoted $z = \mu X^{-1} e$, we can rewrite the KKT conditions as:

$$\begin{aligned} Ax &= b, \ x > 0 \\ A^T y + z &= c, \ z > 0 \\ Xz &= \mu e. \end{aligned} \tag{2.1.4}$$

Note that the first equation ensures primal feasibility while the second equation ensures dual feasibility, and the third constraint is a perturbation of our *complementarity* condition.

The KKT conditions, as written in (2.1.4), are $2n+m$ equations in $2n+m$ unknowns. If these equations were linear, the problem (2.1.4) could be solved by Gaussian elimination, and the subject of LO would be no more difficult than solving systems of linear equations. However, the perturbed complementarity conditions are nonlinear. Nonlinearity makes the subject of interior point methods for linear optimization a challenging one.

As usual, we need to explore the existence and uniqueness of the solution for (2.1.4). To show uniqueness, we need more information on the barrier problem (2.1.1). The objective function is the sum of a linear function and a strictly convex function, and the feasible region is convex, so there can be at most one point that minimizes the objective function, and if it exists, then it must be a global minimum. So the solution is unique if it exists.

Existence of the minimizer is not obvious, as the following example illustrates.

Example 2.1.1. *Consider the following trivial optimization problem on the nonnegative half-line:*

$$\min\{0 \mid x \geq 0\}.$$

For this problem, the barrier function is

$$f(x) = -\mu \log x,$$

which does not have a minimum (or, less precisely, the minimum is $-\infty$ which is attained as $x \to \infty$). To get an example such that the corresponding barrier function has a global minimum, we change the objective function in the above to

$$\min\{x \mid x \geq 0\}.$$

In this case, the barrier function becomes

$$f(x) = x - \mu \log x,$$

which is a function whose unique minimum is attained at $x = \mu$.

In general, we have the following result [26]:

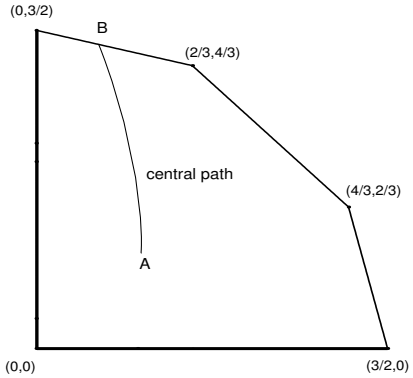


Figure 2.1: Primal Central Path for Example 2.1.2

Proposition 2.1.1. *Problem (2.1.4) has a unique solution if and only if rank*$(A) = m$ *and the IPC holds, which means both the primal and the dual feasible regions have nonempty interiors.*

Equation (2.1.4) defines a smooth curve $\{(x(\mu), y(\mu), z(\mu)) : \mu > 0\}$ called the *primal-dual central path* that plays a fundamental role in IPMs for LO. Also $\{(x(\mu) : \mu > 0\}$ is called the *primal central path*, and $\{(y(\mu), z(\mu)) : \mu > 0\}$ is called the *dual central path*. Now we use an example to show some properties of this central path.

Example 2.1.2. *Let us consider the problem*

$$\begin{array}{rrrl} \min & -2x_1 & -8x_2 & \\ & -x_1 & -x_2 & \geq -2, \\ & -4x_1 & -x_2 & \geq -6, \\ & -x_1 & -4x_2 & \geq -6, \\ & x_1, & x_2 & \geq 0, \end{array} \tag{2.1.5}$$

Figure 2.1 shows the feasible region for this example (2.1.2). The central path is shown as well. Note that the slack variables for the problem are not included. The following can be observed.

1. *The central path is a smooth curve from point A to point B. Point A is the analytic center. As* $\mu \to \infty$, $x(\mu)$ *converges to the analytic*

center A. On the other hand, as μ goes to zero, $x(\mu)$ converges to B. The point B is an optimal solution and is in the analytic center of the optimal set. The optimal set consists of the line segment connecting (0,3/2) and (2/3, 4/3).

2. *The central path stays in the interior of the feasible region. Therefore, an algorithm that follows the central path should generate points in the interior of $\mathcal{F}_P$.*

3. *The example demonstrates that if the central path is followed towards an optimal point, then in degenerate case no basic solution is obtained. The optimal point is in the relative interior of the set of optimal solutions , (e.g. in Example 2.1.2).*

In general, it is impossible to solve the system of nonlinear equations (2.1.4) explicitly. Instead of solving (2.1.4) exactly, IPMs generate points 'close' to the central path. Therefore we will need to measure somehow the distance from the present iterate to the central path. Primal-dual path following IPMs are algorithms that generate points in a certain neighborhood of the central path that converge towards the optimal set. This means that the system of nonlinear equations (2.1.4) can only be solved approximately. A neighborhood of the central path is defined by

$$\mathcal{N}(\delta,\mu) := \{(x,y,z) \ : \ (x,z) > 0, \ Ax = b, \ A^T y + z = c, \ \Psi(xz,\mu) \leq \delta\} \tag{2.1.6}$$

where $\Psi(xz,\mu)$ is a proximity function to measure the distance from the present point to the central path, and $\delta > 0$ is the proximity parameter. Various centrality measures have been developed in the literature of IPMs. The following are two popular proximity measures for IPMs [23]:

$$\Psi_1(xz,\mu) := \frac{1}{2}\left\| \left(\frac{xz}{\mu}\right)^{\frac{1}{2}} - \left(\frac{\mu}{xz}\right)^{\frac{1}{2}} \right\|, \tag{2.1.7}$$

$$\Psi_2(xz,\mu) := \frac{x^T z}{2\mu} - \frac{n}{2} + \frac{n\log\mu}{2} - \frac{1}{2}\sum_{i=1}^{m}\log(x_i z_i), \tag{2.1.8}$$

It easy to see that (for $i = 1, 2$) $\Psi_i(xz, \mu) = 0$ if the point (x, z) is on the central path, and $\Psi_i(xz, \mu) \to \infty$, if point (x, z) approaches the boundary of the nonnegative orthant.

2.1.1 The Newton Direction

Our aim is to find $(\Delta x, \Delta y, \Delta z)$ such that the new point $(x + \Delta x, y + \Delta y, z + \Delta z)$ closer to the primal-dual central path, i.e. it is close to the point $(x(\mu), y(\mu), z(\mu))$. Recalling the defining equations (2.1.4), we see that if the new point $(x + \Delta x, y + \Delta y, z + \Delta z)$ lies exactly on the central path, then it would satisfy

$$\begin{aligned} A(x + \Delta x) &= b, \\ A^T(y + \Delta y) + (z + \Delta z) &= c, \\ (x + \Delta x)(z + \Delta z) &= \mu e. \end{aligned}$$

Here (x, y, z) is the current feasible point and $(\Delta x, \Delta y, \Delta z)$ is the unknown displacement. We rewrite these equations with the unknowns on the left hand side and the data on the right hand side. Using feasibility, we get

$$\begin{aligned} A\Delta x &= 0, \\ A^T\Delta y + \Delta s &= 0, \\ x\Delta z + z\Delta x + \Delta z\Delta x &= \mu e - xz. \end{aligned} \tag{2.1.9}$$

These equations form a system of nonlinear equations (for the "delta" variables). The Newton step is obtained by dropping the nonlinear terms, resulting in the following linear system:

$$\begin{aligned} A\Delta x &= 0, \\ A^T\Delta y + \Delta z &= 0, \\ x\Delta z + z\Delta x &= \mu e - xz. \end{aligned} \tag{2.1.10}$$

Now (2.1.10) is a linear system of $2n + m$ equations in $2n + m$ unknowns. This is called *the primal-dual Newton system.* If matrix A has full rank, then this system is nonsingular and therefore it has a unique solution that defines

the Newton directions for the path-following method. Along this step the equality constraints hold, thus the new iteration point can always be kept in the interior of the primal-dual feasible sets by choosing a proper step length. An important relation between Δx and Δz, namely *orthogonality*, is given below.

Lemma 2.1.1. *Given* $(\Delta x, \Delta y, \Delta z)$ *as a solution of the Newton system (2.1.10), then*

$$\Delta x^T \Delta z = 0.$$

Proof.

$$\Delta x^T \Delta z = \Delta x^T (-A^T \Delta y) = -(A\Delta x)\Delta y = 0.$$

□

2.1.2 Step Length

The Newton direction is determined under the assumption that the step length α would be equal to one (i.e., $\bar{x} = x + \Delta x$, etc.). However, taking such a step might get out of the feasible set and one would lose strict positivity. Hence, in order to keep strict feasibility, the step size has to be strictly less than $\alpha^{\max}$, where $\alpha^{\max}$ is defined by

$$\alpha^{\max} = \underset{\alpha \geq 0}{\operatorname{argmax}} \left\{ \begin{pmatrix} x \\ z \end{pmatrix} + \alpha \begin{pmatrix} \Delta x \\ \Delta z \end{pmatrix} \geq 0 \right\},$$

the maximum possible step size. It will not guarantee strict inequality, so a damp factor $\rho \in (0,\ 1)$ is needed. A possible choice of α is

$$\alpha = \min\{\rho\ \alpha^{\max}, 1\}. \tag{2.1.11}$$

In practice, $\alpha^{\max}$ is computed as follows

$$\alpha_P^{\max} = \min_j \{ \frac{-x_j}{\Delta x_j} : \Delta x_j < 0, j = 1, \ldots, n \}$$

and

$$\alpha_D^{\max} = \min_i \{\frac{-z_i}{\Delta z_i} : \Delta z_i < 0, i = 1, \ldots, n\}$$

then

$$\alpha^{\max} = \min\{\alpha_P^{\max},\ \alpha_D^{\max}\}.$$

This formula only require an order of $2n + m$ operations.

2.1.3 Centering Parameter

The barrier parameter μ is also called the *centering parameter*. Now we need to consider how to choose μ. If μ is chosen to be too large, then the sequence could converge to the analytic center of the feasible set, which is not our intention. On the other hand, if μ is chosen to be too small, then the sequence could stay too far from the central path and the algorithm could *jam* into the boundary of the feasible set. How to find a reasonable μ between these two extremes is not easy. So first we need to figure out a value that represents the current value of μ, and then we choose a smaller number than that, say a fixed fraction of it.

If (x, z) is lies on the central path, then we can use $\mu = \frac{x^T z}{n}$. We use this formula to produce an estimate for μ even when the current solution (x, z) does not lie on the central path. Actually this value of μ given the minimum value of $\Psi_2(xz, \mu)$. Of course, the algorithm needs a value of μ that represents a point closer to optimality than the current solution. We can reduce μ by a certain fraction:

$$\mu = (1 - \theta)\frac{x^T z}{n}$$

where θ is a number between 0 and 1. How to chose the parameter θ is another important issue for IPMs, which will be discussed in more detail in the Section 2.3.

2.1.4 Interior Point Algorithms

The primal-dual path-following algorithm was first proposed by Kojima, Mizuno and Yoshise [15]. This method seeks to approach a solution through a sequence of strictly feasible points while the complementarity condition is relaxed.

The method starts from a point that satisfies *the interior point condition* IPC[1]

$$Ax^0 = b, \quad x^0 > 0; \qquad A^T y^0 + z^0 = c, \quad z^0 > 0,$$

and is in the neighborhood $\mathcal{N}_\delta$ of the central path with an appropriate value μ. If (IPC) holds and $\text{rank}(A) = m$, then the system (2.1.4) has a unique solution for any μ. As we discussed, the unique solution for the system (2.1.4) cannot be calculated exactly, therefore Newton steps are used to get approximate solutions. The primal-dual path-following interior point algorithm can be stated as follows.

[1]IPC can be assumed without loss of generality. We will deal with this in Section 2.4.

Primal-Dual Path-Following Algorithm

Input:

A proximity parameter $\delta > 0$;
an accuracy parameter ε;
a fixed barrier update parameter θ, $0 < \theta < 1$;
(x^0, z^0) and $\mu^0 = 1$ such that $\Psi(x^0 z^0, \mu^0) \leq \delta$.

begin
$x := x^0$; $z := z^0$; $\mu := \mu^0$;
while $x^T z \geq \varepsilon$ **do** (outer iteration)
begin
$\mu := (1 - \theta)\mu$;
while $\Psi(xz, \mu) \geq \delta$ **do** (inner iteration)
begin
Solve (2.1.10) for $\Delta x, \Delta y, \Delta z$;
Compute a step size α;
$x := x + \alpha \Delta x; \quad z := z + \alpha \Delta z; \quad y := y + \alpha \Delta y$
end
end
end

At each outer iteration the duality gap is reduced by the factor $1-\theta$. Usually, if the choice of θ is a constant (e.g. $\frac{1}{2}$) independent of n (the dimension of the problem), then the algorithm is called a large-update (or long-step) method. If θ depends on the dimension n of the problem (e.g. $\theta = \frac{1}{2\sqrt{n}}$) then the algorithm is called a small-update (or short-step) method with respect to this choice. There is a gap between the practical performance and the theoretical worst-case complexity of these two classes of IPMs. The following results can be obtained from literature [22].

	Iter. bound	performance
large-update ($\theta = \frac{1}{2}$)	$O(n \log \frac{n}{\epsilon})$	efficient
small-update ($\theta = \frac{1}{2\sqrt{n}}$)	$O(\sqrt{n} \log \frac{n}{\epsilon})$	poor

Experience shows that in practice, setting $\theta \approx 0.995$ works quite well. This algorithm is referred to as a *feasible IPMs*, since feasibility holds at each iteration. The following is an illustration running this algorithm for Example 2.1.2. The printout is a MATLAB script file. The parameters setting, proximity function, and initial point are

Parameter:	Initial point:	Proximity function:
$\delta = 0.6, \quad \mu^0 = 2.8,$ $\varepsilon = 10^{-6},$ $\theta = 0.99995,$	$x^0 = [1, 0.5]^T,$ $z^0 = [5, 2]^T,$ $y^0 = [0, 0, 0]^T.$	$\Psi_1(xz, \mu)$ as given in (2.1.7)

```
 ======== Primal-Dual Path-Following IPMs ===============
ITER     mu        gap      x1      x2     obj     delta
 0   1.4e-004  8.0e+000  1.000   0.500  -6.000  1.6e+002
 1   1.4e-004  4.3e+000  0.955   0.749  -7.902  1.1e+002
 2   1.4e-004  1.9e+000  0.841   1.080  -10.322 7.4e+001
 3   1.4e-004  5.8e-001  0.686   1.281  -11.620 3.9e+001
 4   1.4e-004  8.6e-002  0.628   1.343  -12.000 1.3e+001
 5   1.4e-004  7.7e-004  0.627   1.343  -11.998 1.2e+000
 6   1.4e-004  3.4e-004  0.485   1.379  -12.002 5.9e-001
 6   7.0e-009  3.4e-004  0.485   1.379  -12.002 1.6e+002
 7   7.0e-009  1.7e-006  0.485   1.379  -12.002 1.1e+001
 8   7.0e-009  2.6e-008  0.483   1.379  -12.002 6.2e-001
 9   7.0e-009  1.6e-008  0.340   1.415  -12.000 1.6e-001
```

The algorithm terminated after 9 iterations with duality gap $= 1.6 \times 10^{-8}$. Figure 2.2[2] shows the practical performance of the Primal-Dual Path-Following IPM with large-update ($\theta = 0.99995$). As the figure demonstrates, after the first $\mu-$reduction center iteration, the new point is out of the neighborhood of the new center, thus in the inner loop, the Newton direction for the system $xz = \mu e$ is employed to force the new iterate move closer to the new center. The inner loop is repeated with fixed μ until the iterate enters the neighborhood $\mathcal{N}(\delta, \mu)$ again. If using small-update (e.g. $\theta = \frac{1.5}{\sqrt{n}} = 0.6780$), then more iterations are needed to terminate the algorithm. Figure 2.3 shows that through the small-update method has better worst-case complexity, it is less efficient in practice than larger-update methods.

[2]Both pictures are drawn in w-space(logarithmic scaled), where $w = \sqrt{xz}$

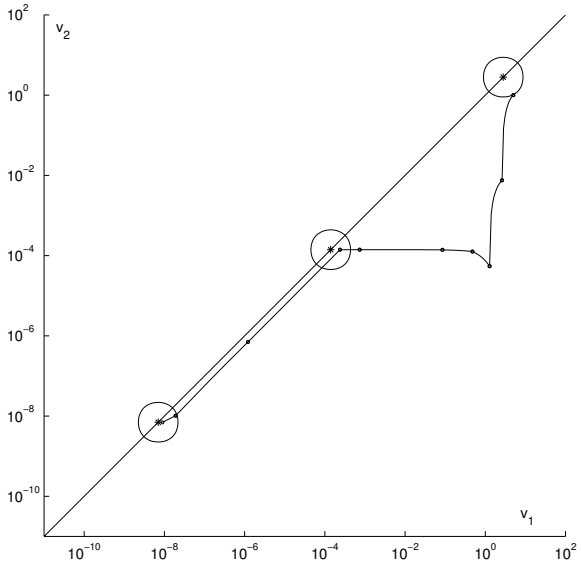


Figure 2.2: Large-update ($\theta = 0.99995$) IPM for Example 2.1.2

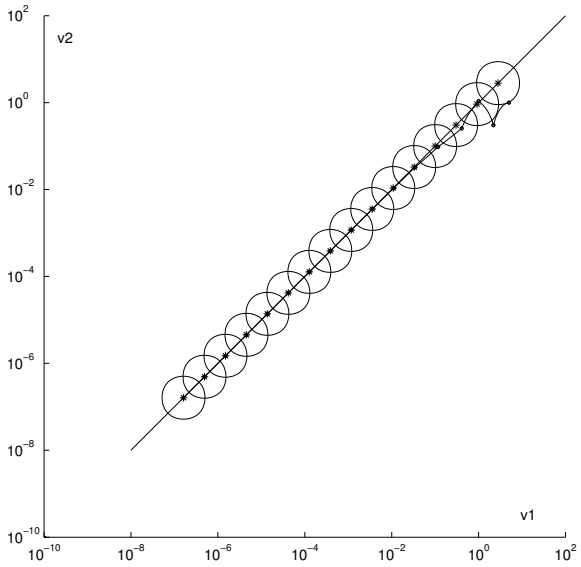


Figure 2.3: Small-update ($\theta = \frac{1.5}{\sqrt{n}}$) IPM for Example 2.1.2

2.2 Solving the Newton System

The most time-consuming part of each iteration of the above primal-dual path-following algorithm is to solve the Newton system (2.1.10). After some manipulations, the Newton system (2.1.10) can be written in matrix form as

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \mu e - Xz \end{pmatrix}, \tag{2.2.1}$$

where X and Z denote the diagonal matrices whose diagonal entries are the components of x and z.

From the last equation of (2.2.1) we have

$$\Delta z = X^{-1}(\mu e - Xz - Z\Delta x).$$

Then the system (2.2.1) can be reduced to

$$\begin{pmatrix} -D^{-2} & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} -X^{-1}(\mu e - Xz) \\ 0 \end{pmatrix}, \tag{2.2.2}$$

where $D^2 = XZ^{-1}$. This system is a symmetric indefinite system and is referred to as *the augmented system*.

From the first equation of (2.2.2), we have

$$\Delta x = -D^2 X^{-1}(\mu e - Xz) + D^2 A^T \Delta y,$$

Finally the augmented system (2.2.2) is reduced to:

$$AD^2 A^T \Delta y \ = -AD^2 \, X^{-1}(\mu e - Xz). \tag{2.2.3}$$

Since $D^2 = XZ^{-1}$ is a nonsingular positive diagonal matrix, equation (2.2.3) is often called the *normal equation form* because it is the normal equation for a linear least squares problem with coefficient matrix DA^T. Most implementations of IPMs are based on the normal equation form.

All IPMs solve a system of linear equations almost identical to (2.2.3); the only difference is in the value of the diagonal matrix D^2 and in the right-hand side. This is the reason why the comparison of different variants of IPMs is often simplified to a comparison of the number of iterations (Newton steps). This linear system can be solved using *direct* methods. A Cholesky decomposition, $AD^2A^T = L\Lambda L^T$, where L is lower triangle and Λ is diagonal, allows back solving.

2.3 Predictor-Corrector Algorithm

Computationally the most expensive step in any implementation of an IPM is the solution of the Newton system (2.1.10). This system has to be solved at each iteration, and requires computing a Cholesky factorization of the symmetric positive matrix defined in (2.2.3). The factorization phase is computationally more expensive than the back-solve phase. Therefore, we can allow several solves in each iterations if these solves help to reduce the total number of interior point iterations and thus the number of factorizations.

Mehrotra's [19] predictor-corrector method is one of these approaches. This method has two components: an adaptive choice of the centering parameter, and the computation of a second-order approximation to the central path.

The first step of the predictor-corrector strategy is to compute the affine scaling (predictor) direction. The affine scaling direction solves the Newton system (2.1.10) for $\mu = 0$ and is denoted by $(\Delta x_a, \Delta y_a, \Delta z_a)$. If a step size α is taken in the affine scaling direction, then the complementarity gap is reduced by the factor $(1 - \alpha)$ as shown from the following and Lemma 2.1.1.

$$\begin{aligned} &(x + \alpha\Delta x_a)^T(z + \alpha\Delta z_a) \\ =&x^T z + \alpha(z^T\Delta x_a + x^T\Delta z_a) + \alpha^2\Delta x_a^T\Delta z_a \\ =&x^T z + \alpha(-x^T z) = (1 - \alpha)x^T z \end{aligned} \tag{2.3.1}$$

Therefore, if a large step can be made in the affine scaling direction, then

considerable progress is achieved in the optimization. On the other hand, if the feasible step size in the affine scaling direction is small, then the current point is probably too close to the boundary. In this case the barrier parameter should not be reduced too much.

After the affine-scaling direction has been computed, the maximum step size α_a along this direction is computed by (2.1.11). Then the predicted complementarity gap

$$g_a = (x + \alpha_a \Delta x_a)^T (z + \alpha_a \Delta z_a)$$

is computed and the centering parameter as suggested by Mehrotra is

$$\mu = \left(\frac{g_a}{g}\right)^2 \frac{g_a}{n}. \tag{2.3.2}$$

Next, the corrector direction is computed. Since we want that the next iterate is well centered, the equation $(x + \Delta x)(z + \Delta z) = \mu e$ is written as

$$Z\Delta x + X\Delta z = -Xz + \mu e - \Delta x_a \Delta z_a, \tag{2.3.3}$$

where instead of neglecting the second order term $\Delta x \Delta z$, as Mehrotra proposed, $\Delta x \Delta z$ is replaced by its approximate value $\Delta x_a \Delta z_a$ computed from the affine scaling direction. The corrector direction is obtained by solving the Newton system (2.1.10) where the last equation is replaced by (2.3.3) and the centering parameter μ is given by (2.3.2). Thus the corrector direction is the unique solution of

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \mu e - Xz - \Delta x_a \Delta z_a \end{pmatrix}. \tag{2.3.4}$$

By now it is clear that at each iteration of the predictor corrector algorithm we need to solve two linear systems of equations instead of one. However, the computational cost is almost the same as in the primal-dual IPM. Since the two systems have the same coefficient matrix, we need to factor the matrix only once, and must perform two back-substitutions, one for each right-hand sides.

Mehrotra's predictor-corrector Algorithm

Input:

An accuracy parameter $\epsilon > 0$;

(x^0, y^0, z^0) initial primal-dual pair.

begin

$x := x^0$; $z := z^0$;

while $x^T z \geq \epsilon$ **do**

begin

Solve(2.1.4) with $\mu = 0$ for $\Delta x_a, \Delta y_a, \Delta z_a$,

Compute the step size α_a by (2.1.11);

$g_a = (x + \alpha_a \Delta x)^T (z + \alpha_a \Delta z)$;

$\mu = (\frac{g_a}{g})^2 \frac{g_a}{n}$;

Solve (2.3.4) for $\Delta x, \Delta y, \Delta z$;

Compute a step size α;

$x := x + \alpha \Delta x; \quad z := z + \alpha \Delta z; \quad y := y + \alpha \Delta y;$

end

end.

2.4 Homogeneous Self-Dual Embedding

The feasible primal-dual algorithm needs a strictly feasible initial solution to start with, and it works with LO problems for which a primal-dual optimal solution exists. However, there exists LO problems that are infeasible. Thus we need to deal with two important issues: initialization and detection of infeasibility. The elegant answer for both questions is the self-dual embedding algorithm. The main idea for this algorithm is to build a slightly larger problem that has a strictly feasible solution, and the solution of the new system should contain enough information about the original problem to decide optimality or infeasibility.

2.4.1 Self-Dual LO Problems

Let us consider the following problem

$$\begin{array}{lrcrcl} (GSP) \quad \min & h^T u & + & g^T v & & \\ & M_1 u & + & Qv & = & -h, \\ & -Q^T u & + & M_2 v & \geq & -g, \\ & u \text{ free}, & & v \geq 0, & & \end{array} \tag{2.4.1}$$

where u and v are variable vectors, $h \geq 0$ and g are constant vectors, Q is a matrix with suitable dimensions, and the matrices M_1 and M_2 are skew-symmetric, that is $M_1^T = -M_1$ and $M_2^T = -M_2$. The dual of (GSP) is:

$$\begin{array}{lrcrcl} (GSD) \quad \max & -h^T u' & - & g^T v' & & \\ & M_1^T u' & - & Qv' & = & h, \\ & Q^T u' & + & M_2^T v' & \leq & g, \\ & u' \text{ free}, & & v' \geq 0. & & \end{array} \tag{2.4.2}$$

Since both M_1 and M_2 are skew-symmetric, (2.4.2) can be written as

$$\begin{array}{lrcrcl} (GSD) \quad - \min & h^T u' & + & g^T v' & & \\ & M_1 u' & + & Qv' & = & -h, \\ & -Q^T u' & + & M_2 v' & \geq & -g, \\ & u' \text{ free}, & & v' \geq 0. & & \end{array} \tag{2.4.3}$$

Now the dual (GSD) is exactly the same as its primal (GSP), therefore problem (GSP) is *self-dual*. The following proposition follows directly from the self-dual property.

Proposition 2.4.1. *If a self-dual LO problem is feasible, then*

1. *its optimal objective value is equal to zero,*
2. *the optimal set is bounded.*

The Goldman-Tucker Theorem 1.1.2 implies the following.

Proposition 2.4.2. *If a self-dual LO is feasible, then it has a strictly complementary solution.*

2.4.2 Embedding

As we discussed in Section 1.1, to solve an LO problem (P), it is enough to find a solution for a system formed by the primal and the dual feasibility constraints, and the inequality requiring that the dual objective value is greater or equal than the primal objective value. The following system, known as the Homogeneous Linear Feasibility problem (HLF), is derived by Goldman-Tucker [7]:

$$\begin{array}{lrrrl} (\mathbf{HLF}) & & Ax & -b\tau & = 0, \\ & -A^T y & & +c\tau & \geq 0, \\ & +b^T y & -c^T x & & \geq 0, \\ & & y \text{ is free}, \ x \geq 0, & \tau \geq 0. & \end{array} \tag{2.4.4}$$

Proposition 2.4.3. *The problem (HLF) has the following properties:*

(i) *(HLF) is an LO problem with zero objective function and a zero right hand side.*

(ii) *(HLF) is a homogeneous linear feasibility problem, that has zero as its trivial solution.*

(iii) *(HLF) is a* self-dual *problem.*

(iv) *(HLF) does not satisfy the interior point condition.*

Clearly the trivial zero solution is not useful for us. We are looking for some nonzero solutions of (HLF); specifically, we would like to have $\tau > 0$. By the Weak Duality Theorem (Proposition 1.1.1), any solution with $\tau > 0$ will give us a primal and dual pair $(\frac{x}{\tau}, \frac{y}{\tau})$ with zero duality gap. Moreover, when we solve (HLF) by an IPM, a strictly feasible point is needed. To reach this goal we introduce an "artificial variable" ν to allow an arbitrary positive vector to be feasible, and we have to introduce an extra constraint to preserve self-duality. Let the vectors $x^0,\ z^0 \in \mathcal{R}^n_{++},\ y^0 \in \mathcal{R}^m$, and the numbers $\tau^0,\ \nu^0 \in \mathcal{R}_{++}$, be arbitrary. The skew-symmetric system (SP) is

reformulated from (HLF) as follows:

$$\begin{array}{lrcrcrcl} (SP) & \min & & & & & \beta\nu & \\ & \text{s.t.} & & Ax & - & b\tau & -r_P\nu & = 0, \\ & & -A^Ty & & + & c\tau & -r_D\nu & \geq 0, \\ & & b^Ty & - \; c^Tx & & & -r_G\nu & \geq 0, \\ & & r_P^Ty & + \; r_D^Tx & + & r_G\tau & & \geq -\beta, \end{array} \tag{2.4.5}$$

$$y \text{ is free}, \; x \geq 0, \; \tau \geq 0, \; \nu \geq 0,$$

where

$$\begin{array}{llll} r_P & = (Ax^0 - b\tau^0)/\nu^0, & r_D & = (-A^Ty^0 + c\tau^0 - e)/\nu^0, \\ r_G & = (b^Ty^0 - c^Tx^0 - 1)/\nu^0, & \beta & = -r_P^Ty^0 - r_D^Tx^0 - r_G\tau^0. \end{array} \tag{2.4.6}$$

All data used here are from problem (P), so an LO problem (P) can be embedded or built into a self-dual problem of the form (SP), which satisfies the IPC.

Proposition 2.4.4. *The problem (SP) is self-dual and has an interior feasible point* $(y^0, x^0, \tau^0, \nu^0)$.

For simplicity we can choose $y^0 = 0, \quad x^0 = e, \quad \tau^0 = 1, \quad \nu^0 = 1$ as initial point. Then we can apply any IPM, e.g. a primal-dual path-following algorithm or the predictor-corrector algorithm, to solve problem (SP). Because IPMs produce a strictly complementary solution, the following result is critical w.r.t the applicability of IPMs.

Proposition 2.4.5. *If* $(y^*, x^*, \tau^*, \nu^*)$ *is a strictly complementary optimal solution for (SP) then we have [23]:*

(i) *If* $\tau^* > 0$, *then* $\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}$ *are strictly complementary optimal solutions of (P) and (D).*

(ii) *When* $\tau^* = 0$, *there are three possibilities:*

- *If* $c^Tx^* < 0$, *then (P) is infeasible.*
- *If* $b^Ty^* > 0$, *then (D) is infeasible.*

- *If* $c^T x^* < 0$ *and* $b^T y^* > 0$*, then both (P) and (D) are infeasible.*

The main advantage of this approach is

(i) It solves the LO problem (P) without any assumption concerning the existence of optimal, or of interior feasible solutions.

(ii) If the problem is infeasible or unbounded, the algorithm correctly detects infeasibility for either (P) or (D).

(iii) The algorithm may start from any positive primal-dual pair, feasible or infeasible, and the initial point can be chosen to be on the central path.

(iv) It preserves $O(\sqrt{n}\log\frac{n}{\varepsilon})$ iteration complexity for small-update methods, and $O(n\log\frac{n}{\varepsilon})$ for large-update methods.

Although (SP) is bigger in size, the cost of one iteration is almost the same as for the original problem. This issue will be discussed in Chapter 5.

Chapter 3

Self-Regular Proximity Based IPMs

A new framework for the theory of primal-dual IPMs by Peng, Roos and Terlaky [22], based on the bedrock of self-regularity*, will be described. The complexity bound for large-update IPMs is improved by using self-regular functions.*

3.1 Self-Regular Functions and Their Properties

Proximity measures or potential functions play an important role in the theory and the implementation of IPMs. To describe the new family of algorithms, we introduce the following notation.

Let

$$v := \sqrt{\frac{xz}{\mu}}, \quad v^{-1} := \sqrt{\frac{\mu e}{xz}} \tag{3.1.1}$$

denote the vectors whose i^{th} components are $\sqrt{\frac{x_i z_i}{\mu}}$ and $\sqrt{\frac{\mu}{x_i z_i}}$, respectively. By using this notation, the centrality condition can be rewritten as $v = e$, or $v^{-1} = e$, or equivalently, in coordinate-wise notation, $v_i = 1$ for all $i =$

$1, 2, \ldots, n$. Theoretically, any strictly convex nonnegative univariate function in $\mathcal{R}_{++}$ that attains its global minimum at 1 can be used to measure the proximity of any point in $\mathcal{R}_{++}$ to 1. This is one of the main features of the new class of *self-regular* functions. For working with IPMs, the measure should ensure that v_i stays positive, thus the measure needs to have a certain barrier property that prevents the iterate from moving to the boundary of the feasible region. Let $\mathcal{C}^2$ be the set of twice differentiable functions.

Definition 3.1.1. *The function $\psi(t) \in \mathcal{C}^2 : (0, \infty) \to \mathcal{R}_+$ is* self-regular *if it satisfies the following conditions:*

SR1: *$\psi(t)$ is strictly convex with respect to $t > 0$ and vanishes at its global minimal point $t = 1$, i.e., $\psi(1) = \psi'(1) = 0$. Furthermore, there exist positive constants $\nu_2 \geq \nu_1 > 0$ and $p \geq 1, q \geq 1$ such that*

$$\nu_1(t^{p-1} + t^{-1-q}) \leq \psi''(t) \leq \nu_2(t^{p-1} + t^{-1-q}), \quad t \in (0, \infty); \tag{3.1.2}$$

SR2: *For any $t_1, t_2 > 0$,*

$$\psi(t_1^r t_2^{1-r}) \leq r\psi(t_1) + (1-r)\psi(t_2), \quad r \in [0, 1]. \tag{3.1.3}$$

The parameter q and p are called the *barrier degree* and the *growth degree* of $\psi(t)$, respectively. From $\psi(1) = \psi'(1) = 0$, we have

$$\psi(t) = \int_1^t \int_1^\xi \psi''(\zeta) d\zeta d\xi.$$

Now let us consider the special case $\nu_1 = \nu_2 = 1$. Then (3.1.2) implies

$$\psi''(t) = t^{p-1} + t^{-1-q},$$

and $\psi(t)$ is uniquely determined by p and q. This special case of self-regular functions plays an important role both in theory and in our implementations. Therefore we introduce a special notation $\Upsilon_{p,q}(t)$ for this class of self-regular functions with $p \geq 1$,

$$\Upsilon_{p,q}(t) = \begin{cases} \frac{t^{p+1}-1}{p(p+1)} + \frac{p-1}{p}(t-1) - \log t, & q = 1 \\ \frac{t^{p+1}-1}{p(p+1)} + \frac{t^{1-q}-1}{q(q-1)} + \frac{p-q}{pq}(t-1), & q > 1. \end{cases} \tag{3.1.4}$$

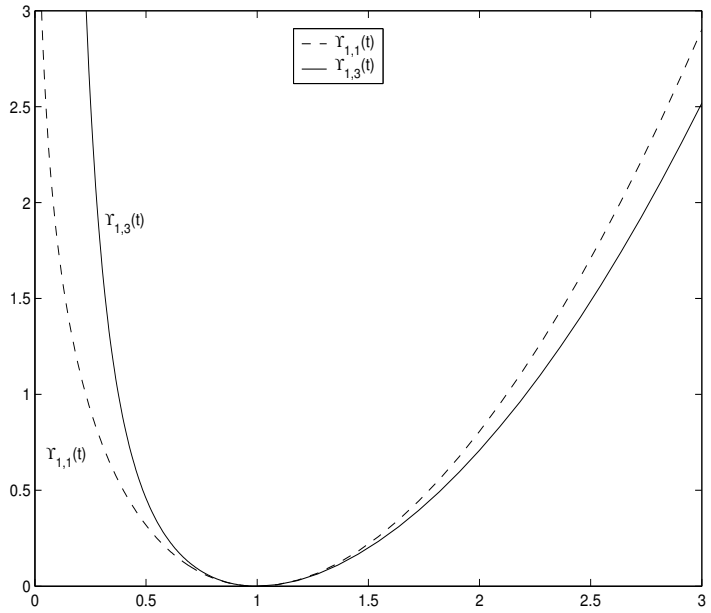


Figure 3.1: Two special Self-Regular functions

Let $\psi(t) = \Upsilon_{p,q}(t)$. Then we have

$$\psi'(t) = \begin{cases} \frac{t^p}{p} - \frac{1}{t} + \frac{p-1}{p}, & q = 1 \\ \frac{t^p}{p} - \frac{t^{-q}}{q} + \frac{p-q}{pq}, & q > 1; \end{cases}$$

$$\psi''(t) = \begin{cases} t^{p-1} + t^{-2}, & q = 1 \\ t^{p-1} + t^{-1-q}, & q > 1. \end{cases}$$

When $p = 1$ and $q = 1$ we get the well-known (univariate) logarithmic barrier function [23]

$$\Upsilon_{1,1}(t) = \frac{t^2 - 1}{2} - \log t.$$

One may easily verify the following lemma.

Lemma 3.1.1. *[22] The function $\Upsilon_{p,q}(t)$ is self-regular when $p, q \geq 1$.*

Figure 3.1 demonstrates the growth and barrier behaviors of two prototype self-regular functions $\Upsilon_{1,1}(t)$ and $\Upsilon_{1,3}(t)$. From this figure we can see

that the growth behaviors of these two functions are quite similar as $t \to \infty$. However, when $t \to 0$, the function $\Upsilon_{1,3}(t)$ has a much stronger barrier property than $\Upsilon_{1,1}(t)$.

Let $\Gamma_{p,q}(t)$ denote another sub-class of self-regular function:

$$\Gamma_{p,q}(t) = \frac{t^{p+1}-1}{p+1} + \frac{t^{1-q}-1}{q-1}, \quad p \geq 1, \ q > 1. \tag{3.1.5}$$

Let $\psi(t) = \Gamma_{p,q}(t)$. Then we have

$$\psi'(t) = t^p - t^{-q} \tag{3.1.6}$$

$$\psi''(t) = pt^{p-1} + qt^{-q-1}. \tag{3.1.7}$$

Note that if $p = q > 1$ then $\Gamma_{p,p}(t) = p\Upsilon_{p,p}(t)$.

The following results about the class of self-regular functions are all from [22] and the interested reader may consult that reference for proofs and other details.

Proposition 3.1.1. *If the functions $\psi_1(t)$ and $\psi_2(t)$ are self-regular, then any combination of the two functions $\beta_1\psi_1(t)+\beta_2\psi_2(t)$ with $\beta_1, \beta_2 \geq 0$, $\beta_1+\beta_2 > 0$ is self-regular.*

Lemma 3.1.1 means that the set of self-regular functions is a pointed convex cone. Observe that this result does not imply that the cone is closed.

Proposition 3.1.2. *If $\psi(t) = \psi(t^{-1})$ and $\psi(t)$ satisfies condition* **SR1**, *then $\psi(t)$ is self-regular.*

So far we have been dealing with univariate functions. However LO problems are defined in multidimensional spaces, thus we need to extend the notion of *self-regular functions* to the positive orthant $\mathcal{R}^n_{++}$ of the n-dimensional Euclidean space.

Definition 3.1.2. *A mapping*

$$\psi(v) : \mathcal{R}^n_{++} \to \mathcal{R}^n_+$$

defined by

$$\psi(v) = (\psi(v_1), \ldots, \psi(v_n))^T \tag{3.1.8}$$

is said to be self-regular if the kernel function $\psi(t) : \mathcal{R}_{++} \to \mathcal{R}_+$ *is self-regular. Analogously, the function* $\Psi(v) = e^T\psi(v)$ *is called self-regular if the based function* $\psi(v)$ *is self-regular.*

3.2 Self-Regular Proximities

The new Self-Regular proximity (SR-proximity) functions for IPMs were defined in [22] are as

$$\Psi(v) := \sum_{i=1}^{n} \psi(v_i), \tag{3.2.1}$$

where $\psi(\cdot)$ is a self-regular function. For notational convenience, we also define

$$\psi'(v) = (\psi'(v_1), \psi'(v_2), \ldots, \psi'(v_n))^T,$$
$$\psi''(v) = (\psi''(v_1), \psi''(v_2), \ldots, \psi''(v_n))^T,$$

and

$$\sigma = \|\nabla\Psi(v)\|,$$

where

$$\nabla\Psi(v) = \nabla\left(\sum_{i=1}^{n} \psi(v_i)\right) = \psi'(v).$$

The following characterization of SR-proximity functions transfers the Self-Regular property to $\mathcal{R}^n$.

Proposition 3.2.1. *[22] Let the functions* $\psi(v)$ *and* $\Psi(v)$ *be defined by (3.1.8) and (3.2.1) respectively. If the kernel function* $\psi(t)$ *is* self-regular *with parameters* $\nu_2 \geq \nu_1 > 0$ *and* $p, q \geq 1$*, then*

(i) $\Psi(v)$ *is strictly convex with respect to* $v \in \mathcal{R}^n_{++}$*, and vanishes at its global minimal point* $v = e$ *(where* e *is the unit vector), i.e.,* $\Psi(e) = 0$ *and* $\psi(e) = \psi'(e) = 0$*. Further,*

$$\nu_1(v^{p-1} + v^{-1-q}) \leq \psi''(v) \leq \nu_2(v^{p-1} + v^{-1-q}), \quad \forall v \in \mathcal{R}^n_{++}; \tag{3.2.2}$$

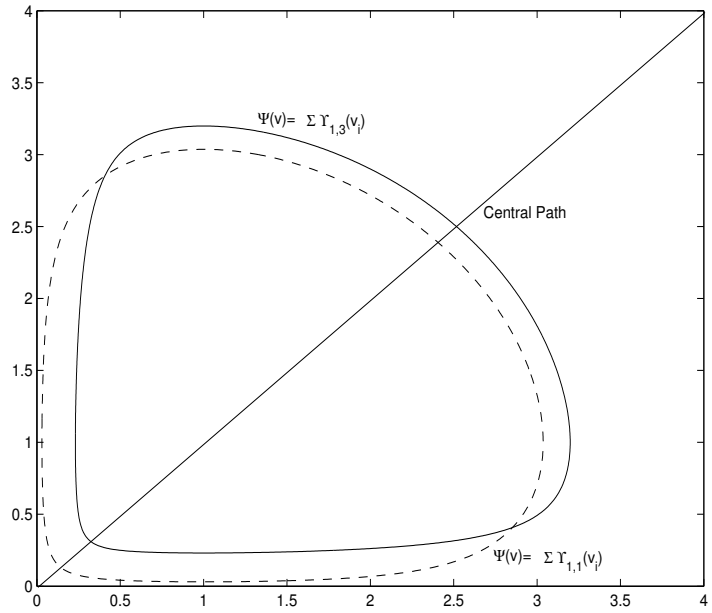


Figure 3.2: Two Special neighborhoods defined by SR proximities

(ii) For any $x, s \in \mathcal{R}^n_{++}$,

$$\psi(x^r s^{1-r}) \leq r\psi(x) + (1-r)\psi(s), \quad \forall r \in [0,1]. \tag{3.2.3}$$

The next result establishes a relation between the proximity value and the norm of its gradient. Further, the growth of the proximity along a ray ϑv ($\vartheta > 0$) is estimated. Those properties of the proximity $\Psi(v)$ are shared by general self-regular functions in $\mathcal{R}^n_{++}$.

Proposition 3.2.2. *[22] Let the proximity* $\Psi(v)$ *be defined by 3.2.1. Then*

$$\Psi(v) \leq \tfrac{\sigma^2}{2\nu_1},$$

and for any $\vartheta > 1$,

$$\Psi(\vartheta v) \leq \frac{\nu_2}{\nu_1}\left(\vartheta^{p+1}\Psi(v) + \vartheta\nabla\Upsilon_{p,q}(\vartheta)\sqrt{2n\nu_1\Psi(v)} + n\nu_1\Upsilon_{p,q}(\vartheta)\right).$$

The proximity $\Psi(v)$ can be defined as the deviation/distance of the given point v from the central path. In Section 2.1 we defined the neighborhood

as

$$\mathcal{N}(\delta,\mu) := \{(x,y,z) \ : \ (x,z) > 0, \ Ax = b, \ A^T y + z = c, \ \Psi(xz,\mu) \leq \delta\}.$$

If the proximity function $\Psi(v)$ is an SR function, then we say the neighborhood $\mathcal{N}(\delta,\mu)$ is defined by SR proximity.

Figure (3.2) exhibits two neighborhoods defined by two special SR proximities, $\Psi(v) = \sum_{i=1}^{2} \Upsilon_{1,1}(v_i)$ and $\Psi(v) = \sum_{i=1}^{2} \Upsilon_{1,3}(v_i)$, respectively. The picture is drawn in scaled $v-$space with $\delta = 3$, $\mu = 1$.

One can see that the neighborhood defined by $\Upsilon_{1,3}$ has a better control on the distance to the boundary of the feasible set than the neighborhood defined by the logarithmic barrier function $\Upsilon_{1,1}$.

3.3 Self-Regular Search Directions

In this section we introduce the following Self-Regular search directions (SR-directions) based on SR-proximities and discuss some of their properties. To facilitate this development we introduce the following notation:

$$d_x := \frac{v\Delta x}{x}, \quad d_z := \frac{v\Delta z}{z}, \tag{3.3.1}$$

and

$$\bar{A} = \frac{1}{\mu} A V^{-1} X, \quad V = \text{diag}(v), \quad X = \text{diag}(x).$$

By the definition of v (3.1.1), the centrality condition $xz = \mu e$ can be rewritten as $v = v^{-1} = e$, and the new notation allows us to rewrite the Newton system (2.1.10) in the $v-$space as

$$\begin{aligned} \bar{A} d_x &= 0, \\ \bar{A}^T d_y + d_z &= 0, \\ d_x + d_z &= v^{-1} - v, \end{aligned} \tag{3.3.2}$$

Let $d_v = d_x + d_z = v^{-1} - v$. We can easily verify that

1. if $v_i \geq 1$ then $d_{v_i} \leq 0$, i.e. the component v_i will decrease at the next iteration;
2. if $v_i < 1$ then $d_{v_i} \geq 0$, i.e., the component v_i will increase at the next iteration.

Therefore the Newton step increases those components v_i which are less than one and decreases those components v_i which are larger than one. Consequently, the Newton direction "decreases the proximity", "brings the iterate closer", and "points towards" the central path. It is clear from (3.3.1) and (3.3.2) that d_x and d_z are the orthogonal decomposition of the vector d_v w.r.t. the null space and row space of $\bar{A}$. Another interesting observation is that we can always decompose system (3.3.2) into two systems by replacing the last equation in (3.3.2) by

$$d_v^P = -v \quad \text{and} \quad d_v^C = v^{-1}.$$

The direction d_v^P is the same as the Newton direction with $\mu = 0$ that aims to decrease the duality gap to zero in one step. This direction is called the predictor, or affine scaling direction. On the other hand, the direction d_v^C serves the purpose of centering (it points towards the "analytic center" of the feasible set).

The motivation of the new search direction is that if we can increase the small components and decrease the large components of v more, we can follow the central path more efficiently while staying in a large neighborhood of the path. Let us define a family of new search directions by a modified system (3.3.2) as follows:

$$\begin{aligned} \bar{A}d_x &= 0, \\ \bar{A}^T\Delta y + d_z &= 0, \\ d_x + d_z &= v^{-q} - v^p; \end{aligned} \tag{3.3.3}$$

where $p, q \geq 1$ are parameters. That is, instead of the classical centering direction $d_v^P = v^{-1}$ and $d_v^P = v$, a new set of directions

$$d_v^P = v^{-q} \quad \text{and} \quad d_v^C = v^p$$

is employed. The new search directions with $q > 1$ and $p > 1$ will decrease the large components $v_i > 1$ and increase the small components $v_i < 1$ more than the classical search direction. This further indicates that the new search directions might help us to find a new primal-dual iterate (x, y, z) close to the central path faster. Note those directions are the projected steepest descent directions in the $v-$space for the SR-proximity function

$$\Psi(v) = \sum_{i=1}^{n} \Gamma_{p,q}(v_i),$$

because

$$-\nabla \left(\sum_{i=1}^{n} \Gamma_{p,q}(v_i) \right) = v^{-q} - v^{p}.$$

Then the system (3.3.3) can be rewritten as

$$\begin{aligned} \bar{A} d_x &= 0, \\ \bar{A}^T \Delta y + d_z &= 0, \\ d_x + d_z &= -\nabla \Psi(v); \end{aligned} \tag{3.3.4}$$

or equivalently

$$\begin{aligned} A \Delta x &= 0, \\ A^T \Delta y + \Delta z &= 0, \\ z \Delta x + x \Delta z &= -\mu v \nabla \Psi(v). \end{aligned} \tag{3.3.5}$$

The last equation (3.3.5) represents the Newton system for the equation

$$xz = -\mu v(\nabla \Psi(v) - v).$$

When we use the new proximity $\Psi(v)$ with a large q to define the neighborhood and employ the new system (3.3.5), the small entries $x_i z_i$ (or equivalently v_i) are heavily penalized in terms of the neighborhood as well as in the definition of the search directions. This is the main reason why the new search directions improve the complexity of large-update IPMs [22]. It is natural to extend $\Psi(v) = \sum_{i=1}^{n} \Gamma_{p,q}(v_i)$ to a general SR-proximity function

$\Psi = \sum_{i=1}^{n} \psi(v_i)$, where $\psi(v_i)$ is an SR function; the new search directions defined by (3.3.5) are then called *SR-directions*. In the next Section we talk about a new family of IPMs based on SR-directions.

3.4 SR-Based IPMs

In this section we outline the IPM based on SR-directions and SR-proximities that we introduced in Sections 3.2 and 3.3.

The basic idea is this: whenever another proximity measure is used in the algorithm, one should adapt the search direction correspondingly. SR-based IPMs use SR-proximity functions in the path following method to control the "distance" of a point to the central path, and the search direction is obtained by solving system (3.3.5), where $\nabla\Psi(v)$ is the gradient of the SR-proximity in the scaled $v-$space.

Comparing the classical Newton system (3.3.2) and the modified system (3.3.5), one sees that only the right hand sides are different. Thus if $\text{rank}(A) = m$, then for any $\mu > 0$, system (3.3.5) has a unique solution $(\Delta x,\ \Delta y,\ \Delta z)$. Further we can assume without loss of generality that if a threshold value δ for the proximity is used in the algorithm, then we can assume that a triple (x^0, y^0, z^0) is given with $\Psi(x^0 z^0, \mu^0) \leq \delta$ for $\mu^0 = 1$ [23].

If, for the current iteration (x, y, z) and barrier parameter value μ, the proximity $\Psi(xz, \mu)$ exceeds δ, then we use one or more damped Newton steps to re-center; otherwise μ is reduced by the factor $1-\theta$. This is repeated until $n\mu < \varepsilon$. Thus the algorithm can be stated as follows.

Self-Regular Based Primal-Dual Algorithm

Input:

A SR-proximity function $\Psi(v)$ and a proximity parameter $\delta > \nu_1^{-1}$;
an accuracy parameter $\varepsilon > 0$;
a fixed barrier update parameter θ, $0 < \theta < 1$;
(x^0, z^0) and $\mu^0 = 1$ such that $\Psi(x^0 z^0, \mu^0) \leq \delta$.

begin
$x := x^0$; $z := z^0$; $\mu := \mu^0$;
while $n\mu \geq \varepsilon$ **do**
begin
$\mu := (1 - \theta)\mu$;
while $\Psi(xz, \mu) \geq \delta$ **do**
begin
Solve (3.3.5) for $\Delta x, \Delta y, \Delta z$;
Compute the step size α;
$x := x + \alpha\Delta x; \quad y := y + \alpha\Delta y; \quad z := z + \alpha\Delta z;$
end
end
end

The new SR-IPMs keep the same algorithm structure as classic primal-dual path-following IPMs, but new proximity (SR-proximity) and new search directions (SR-directions). This new family of IPMs have the best known worst case complexity [22] and help us to solve difficult problems.

3.5 Complexity

There are many choices for the parameters δ and θ in the above new algorithm. If δ is a small constant independent of n and $\theta = O(\frac{1}{\sqrt{n}})$, then the algorithm is called an IPM with small neighborhood, which has the best known iteration bound $O\left(\sqrt{n}\log\frac{n}{\varepsilon}\right)$. If δ is chosen to be a number related to n, e.g. $\delta = n$, then we call the algorithm an IPM with large neighborhood.

For large neighborhood IPMs, the best known iteration bound before the SR-proximity measure [22] was introduced was $O\left(n \log \frac{n}{\varepsilon}\right)$. This is improved by using a SR-proximity measure as presented in Theorem 3.4.3 of [22]. The iterative complexity of result for large-update LO algorithms is

$$O\left(n^{\frac{q+1}{2q}} \log \frac{n}{\varepsilon}\right)$$

which is better than the previously known $O(n \log \frac{n}{\varepsilon})$ iteration bound. If we choose $p = 2,\ q = \log n$, then the upper bound for the total number of iterations is [22]:

$$O\left(\sqrt{n} \log n \log \frac{n}{\varepsilon}\right).$$

This gives the best known complexity bound for large-update methods at the time we prepare this thesis.

Chapter 4

From Theory to Computational Practice

In this chapter we adapt the SR-based IPMs presented in Chapter 3 to the case when general upper bounds are present in the LO problem. Moreover, we work out all the formulas in such a detail as to allow direct and efficient implementation.

4.1 Introduction

As discussed in Section 2, the computationally most expensive part of an IPM is to solve the augmented system:

$$\begin{pmatrix} -D^{-2} & A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \text{RHS1}, \tag{4.1.1}$$

or the normal equation

$$AD^2A^T\Delta y = \text{RHS2}. \tag{4.1.2}$$

Therefore the size and structure of A will significantly affect the computation cost.

In most practical LO problems, in addition to the equality and nonnegativity constraints, some variables may have bounds. If a variable has lower

bound, i.e. $x_i \geq l_i$, then we can just shift the lower bound to zero, i.e. let $x_i := x_i - l_i \geq 0$. After this step matrix A remains the same size. If a variable has an upper bound, i.e. $x_i \leq u_i$, an extra slack variable will be needed to change this inequality constraint to an equality constraint. Therefore both the numbers of rows and columns of A will increase by the number of the upper bounded variables. In this case we need to deal with the primal LO problem in the following form:

$$\begin{array}{lll} \min & c^T x & \\ \text{s.t} & Ax = b, & \\ & 0 \leq x_i \leq u_i, & i \in \mathcal{I} \\ & 0 \leq x_j, & j \in \mathcal{J}, \end{array}$$

where $A \in \mathcal{R}^{m \times n}$, $c, x \in \mathcal{R}^n$, $b \in \mathcal{R}^m$, and $\mathcal{I}$ and $\mathcal{J}$ are index sets such that $\mathcal{I} \bigcup \mathcal{J} = \{1, 2, \ldots, n\}$ and $\mathcal{I} \bigcap \mathcal{J} = \emptyset$.

Without loss of generality, the last two constraints can be written as

$$Fx + s = u, \; x \geq 0, \; s \geq 0, \quad F \in \mathcal{R}^{m_f \times n},$$

where $m_f = |\mathcal{I}|$ and $s \in \mathcal{R}^{m_f}$ is the slack vector, and the rows of F are unit vectors. Hence, the above LO problem can be written as:

$$\begin{array}{lll} \text{(LP)} \quad \min & c^T x & \\ \text{s.t} & Ax & = b, \\ & Fx + s & = u, \\ & x, s \geq 0, & \end{array}$$

and its dual as

$$\begin{array}{lll} \text{(LD)} \quad \max & b^T y - u^T w & \\ \text{s.t} & A^T y - F^T w + z = c, & \\ & w, z \geq 0, & \end{array}$$

where $y \in \mathcal{R}^m$, $w \in \mathcal{R}^{m_f}$ and $z \in \mathcal{R}^n$.

In the sequel we show that the Newton equation of the LO problem (LP) can be reduced to a linear system with the size almost identical to the normal equation (2.2.3) of (P). Now let us calculate two basic components of IPMs. The actual duality (or complementarity) *gap* can be computed as:

$$\begin{aligned} gap &= c^T x - (b^T y - u^T w) \\ &= (A^T y - F^T w + z)^T x - y^T (Ax) + w^T (Fx + s) \\ &= x^T z + s^T w. \end{aligned} \tag{4.1.3}$$

The gap is zero if the complementary condition is satisfy, i.e., $x_i z_i = 0$ for all i and $w_j s_j = 0$ for all j.

As a result, the Optimality Conditions for (LP) and (LD) can be written as

$$\begin{aligned} Ax &= b, \\ Fx + s &= u, \\ A^T y + z - F^T w &= c, \\ Xz &= 0, \\ Sw &= 0. \end{aligned} \tag{4.1.4}$$

4.2 Initialization by Self-dual Embedding

Our algorithm is based on the self-dual embedding model (see Section 2.4). This model embeds the original (LP) problem in a slightly larger LO problem, called the homogenous model, that always has an optimal solution. The optimal solution of the homogenous model either proves that the original problem does not have an optimal solution or can easily be converted to an optimal solution of the original problem. In order to describe how to build up this large problem, we first need to construct a homogeneous and self-dual linear program related to (LP) and (LD).

The embedding problem can be formulated as in Section 2.4.2. Let the vectors $x^0,\ z^0 \in \mathcal{R}^n_{++},\ w^0,\ s^0 \in \mathcal{R}^{m_f}_{++},\ y^0 \in \mathcal{R}^m,$ and the numbers $\tau^0,\ \nu^0,\ \kappa^0 \in \mathcal{R}_{++}$, be arbitrary.

Define $r_{\text{P1}} \in \mathcal{R}^m,\ r_{\text{P2}} \in \mathcal{R}^{m_f},\ r_{\text{D}} \in \mathcal{R}^n$ (the scaled errors at the arbitrary initial interior solutions), and $r_{\text{G}} \in \mathcal{R},\ \beta \in \mathcal{R}$ (parameters):

$$\begin{aligned} r_{\text{P1}} &= (Ax^0 - b\tau^0)/\nu_0, \\ r_{\text{P2}} &= (-Fx^0 + u\tau^0 - s^0)/\nu_0, \\ r_{\text{D}} &= (F^T w^0 + c\tau^0 - A^T y^0 - z^0)/\nu_0, \\ r_{\text{G}} &= (-u^T w^0 + b^T y^0 - c^T x^0 - \kappa^0)/\nu_0, \\ \beta &= -r_{\text{P1}}^T w^0 - r_{\text{P2}}^T y^0 - r_{\text{D}}^T x^0 - r_{\text{G}}\tau^0. \end{aligned}$$

The self-dual embedding model is constructed as the following LO problem:

$$\begin{array}{llccccccccccc} \text{(SP) min} & & & & & & & & \beta\nu & & & & \\ \text{s.t.} & & & & & Ax & - & b\tau & -r_{\text{P1}}\nu & & & & = 0, \\ & & & - & Fx & + & u\tau & -r_{\text{P2}}\nu & -s & & & = 0, \\ & -A^T y & + & F^T w & & + & c\tau & -r_{\text{D}}\nu & & -z & & = 0, \\ & b^T y & - & u^T w & - & c^T x & & -r_{\text{G}}\nu & & & -\kappa & = 0, \\ & r_{\text{P1}}^T y & + & r_{\text{P2}}^T w & + & r_{\text{D}}^T x & + & r_{\text{G}}\tau & & & & = -\beta, \\ & y, \ \nu \ \text{free}, & & w \geq 0, & & x \geq 0, & & \tau \geq 0, & z \geq 0, & & s \geq 0, & \kappa \geq 0. & \end{array}$$

The first four constraints in (SP), with $\tau = 1$ and $\nu = 0$, represent primal-dual feasibility (with $x \geq 0, s \geq 0, w \geq 0, z \geq 0$) and the reversed weak duality inequality, thus they define primal and dual optimal solutions for (LP) and (LD). Then an artificial variable ν with appropriate coefficient is added to achieve feasibility for (x^0, s^0) and $(y^0,\ w^0, z^0)$, and the fifth constraint is added to achieve self-duality. So a general linear problem can be transformed into a problem of the form (SP).

We can easily verify the following lemma.

Lemma 4.2.1. *The initial solution* $(y^0,\ w^0,\ x^0,\ \tau^0,\ \nu^0,\ z^0,\ s^0,\ \kappa^0)$ *is interior feasible for problem (SP) and it is on the central path.*

Also note that if , e.g., we choose $y^0 = 0,\ w^0 = e,\ x^0 = e,\ \tau^0 = 1,\ \nu^0 = 1, z^0 = e,\ s^0 = e,\ \kappa^0 = 1$, then this solution is a perfectly centered initial solution for problem (SP) with $\mu = 1$.

Lemma 4.2.2. *The problem (SP) satisfies the (IPC).*

Let $\mathcal{F}_{\mathcal{SP}}$ be the set of feasible solutions of problem (SP). We have

Lemma 4.2.3. *If* $(y,\ w,\ x,\ \tau,\ \nu,\ z,\ s,\ \kappa) \in \mathcal{F}_{SP}$ *then*

$$x^T z + s^T w + \tau\kappa = \nu\beta. \tag{4.2.1}$$

Proof. Multiplying the first, second, third, fourth and fifth equality constraints by $y^T,\ w^T,\ x^T,\ \tau$ and ν, and summing them, we have

$$x^T z + s^T w + \tau\kappa = \nu\beta.$$

The lemma is proved. □

Now let us prove this LO problem is skew-symmetric and self-dual.

Lemma 4.2.4. *The problem (SP) is a self-dual LO problem.*

Proof. Let us rearrange the order of the variables.

$$\begin{array}{lccccccccccccc}
\min & & & \beta\nu & & & & & & & & & & \\
\text{s.t.} & & - & r_{\mathrm{P1}}\nu & & & + & Ax & - & b\tau & & & = 0, \\
& r_{\mathrm{P1}}^T y & & & + & r_{\mathrm{P2}}^T w & + & r_{\mathrm{D}}^T x & + & r_{\mathrm{G}}\tau & & & = -\beta, \\
& & - & r_{\mathrm{P2}}\nu & & & - & Fx & + & u\tau & -z & & = 0, \\
& -A^T y & - & r_{\mathrm{D}}\nu & + & F^T w & & & + & c\tau & & -s & = 0, \\
& b^T y & - & r_{\mathrm{G}}\nu & - & u^T w & - & c^T x & & & & -\kappa & = 0, \\
& & & & & y,\ \nu \text{ free},\ w \geq 0,\ x \geq 0,\ \tau \geq 0, z \geq 0,\ s \geq 0,\ \kappa \geq 0. & & & & & & &
\end{array}$$

Let

$$h = \begin{pmatrix} 0 \\ \beta \end{pmatrix}, \quad Q = \begin{pmatrix} 0 & A & -b \\ r_{\mathrm{P2}}^T & r_{\mathrm{D}}^T & r_{\mathrm{G}} \end{pmatrix},$$

$$M_1 = \begin{pmatrix} 0 & -r_{\mathrm{P1}} \\ r_{\mathrm{P1}}^T & 0 \end{pmatrix}, \text{ and } \quad M_2 = \begin{pmatrix} 0 & -F & u \\ F^T & 0 & c \\ -u^T & -c^T & 0 \end{pmatrix}.$$

The problem (SP) can be written in the more general form that we have analyzed in the Section 2.4. With this notation the (SP) model is in the form of the self-dual problem (2.4.1). The proof is complete. □

Theorem 4.2.1. *The self-dual problem (SP) has the following properties:*

(i) *Problem (SP) has an optimal solution, and its optimal solution set is bounded.*

(ii) *The optimal value of (SP) is zero. If* $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^*,\ z^*,\ s^*,\ \kappa^*)$ *is an optimal solution of (SP), then* $\nu^* = 0$.

(iii) *There is a strictly complementary optimal solution* $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ z^*,\ s^*,\ \kappa^*) \in \mathcal{F}_{SP}$.

Proof. **(i)** This is due to the self-dual property: the dual of (SP) is also feasible, and it has non-empty interior.

(ii) Let $(y, w, x, \tau, \nu, z, s, \kappa)$ be a feasible point for (SP), then it is also a feasible point for its dual. The duality gap is

$$\beta\nu - (-\beta\nu) = 2\beta\nu = 2(x^T z + s^T w + \tau\kappa).$$

Assume that $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^*,\ z^*,\ s^*,\ \kappa^*)$ is an optimal solution of (SP). We have $2\beta\nu^* = 0$, that is $\nu^* = 0$.

(iii) Since both the primal and the dual problem have feasible solutions, due to the Goldman-Tucker Theorem (Theorem 1.1.2) of LO, a strictly complementary solution exists.

□

Theorem 4.2.2. *Let* $(y^*,\ w^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ z^*,\ s^*,\ \kappa^*) \in \mathcal{F}_{sp}$ *be a strictly complementary solution for (SP).*

(i) *If* $\tau^* > 0$ *(so* $\kappa^* = 0$*) then* $(\frac{x^*}{\tau^*}, \frac{z^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{w^*}{\tau^*}, \frac{s^*}{\tau^*})$ *is a strictly complementary optimal solution to (LP) and (LD).*

(ii) *If* $\tau^* = 0$ *(so* $\kappa^* > 0$*) then*

- *if* $c^T x^* < 0$*, then (LD) is infeasible;*

- *if* $-b^T y^* + u^T w^* < 0$*, then (LP) is infeasible;*
- *if* $c^T x^* < 0$ *and* $-b^T y^* + u^T w^* < 0$*, then both (LP) and (LD) are infeasible.*

Proof. **(i)** If $\tau^* > 0$, then the result is directly from (SP).

(ii) If $\tau^* = 0$, then $\kappa^* > 0$, and we have

$$\begin{aligned} Ax^* &= 0, \\ Fx^* + s^* &= 0, \\ A^T y^* - F^T w^* + z^* &= 0. \end{aligned}$$

Further, the fourth constraint and $\nu^* = 0$ imply

$$c^T x^* - (b^T y^* - u^T w^*) < 0,$$

i.e. at least one of $c^T x^*$ and $(-b^T y^* + u^T w^*)$ is strictly less than zero.

- In case $c^T x^* < 0$, if there would exist a feasible solution for (LD), i.e., a $(\hat{y}, \hat{w}, \hat{z})$ such that $\hat{w} \geq 0, \hat{z} \geq 0$ and $A^T \hat{y} - F^T \hat{w} + \hat{z} = c$, then

$$\begin{aligned} 0 &> c^T x^* = (A^T \hat{y} - F^T \hat{w} + \hat{z})^T x^* \\ &= \hat{y}^T (Ax^*) - \hat{w}^T F x^* + \hat{z}^T x^* \\ &= -\hat{w}^T (-s^*) + \hat{z}^T x^* \\ &\geq 0, \end{aligned}$$

  that is a contradiction. Therefore in this case (LD) is infeasible.

- In case $(-b^T y^* + u^T w^*) < 0$, if there would be a feasible solution for (LP), i.i., a $(\hat{x}, \hat{z})$ such that $\hat{x} \geq 0, \hat{z} \geq 0$ and $A\hat{x} = b$ and $F\hat{x} + \hat{z} = u$, then

$$\begin{aligned} 0 &> (-b^T y^* + u^T w^*) \\ &= (-A\hat{x})^T y^* + (F\hat{x} + \hat{s})^T w^* \\ &= \hat{x}^T (-A^T y^* + F^T w^*) + \hat{s}^T w^* \\ &= \hat{x}^T z^* + \hat{s}^T w^* \\ &\geq 0, \end{aligned}$$

  that is a contradiction. Therefore in this case (LP) is infeasible.

□

4.3 Search Directions

Compared with the size of the original LO problem (LP), the size of the resulting embedding problem (SP) is more than doubled . However, computing the search direction for the resulting embedding problem is only slightly more expensive than that for the original problem. This is because the problem is self-dual, which has a very special structure, thus the formulas can be simplified significantly. The small extra effort in solving (SP) brings several important advantages: having a centered initial interior starting point, detecting infeasibility by convergence, and applicability of any feasible IPM without degrading theoretical complexity. The rest of this section is devoted to showing the details of the computation of the Newton direction for the embedding problem (SP). We will show that the augmented and normal equation system for (SP) reduce to almost the same size as that of the augmented (2.2.2) or the normal equation (2.2.3) systems for the problem (LP).

The central path for (SP) is defined as the set of solutions for all $\mu > 0$ of the system:

$$\begin{array}{rrrrrrrrl}
 & & Ax & -b\tau & -r_{\mathrm{P1}}\nu & & & & = 0, \\
 & & -Fx & +u\tau & -r_{\mathrm{P2}}\nu & -s & & & = 0, \\
-A^Ty & +F^Tw & & +c\tau & -r_{\mathrm{D}}\nu & & -z & & = 0, \\
b^Ty & -u^Tw & -c^Tx & & -r_{\mathrm{G}}\nu & & & -k & = 0, \\
r_{\mathrm{P1}}^Ty & +r_{\mathrm{P2}}^Tw & +r_{\mathrm{D}}^Tx & +r_{\mathrm{G}}\tau & & & & & = -\beta, \\
 & & & & & & & Xz & = \mu e, \\
 & & & & & & & Sw & = \mu e, \\
 & & & & & & & \tau\kappa & = \mu.
\end{array} \tag{4.3.1}$$

The Newton system for (4.3.1) is the following:

$$\begin{pmatrix} 0 & 0 & A & -b & -r_{\text{P1}} & 0 & 0 & 0 \\ 0 & 0 & -F & u & -r_{\text{P2}} & -I & 0 & 0 \\ -A^T & F^T & 0 & c & -r_{\text{D}} & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & -r_{\text{G}} & 0 & 0 & -I \\ r_{\text{P1}}^T & r_{\text{P2}}^T & r_{\text{D}}^T & r_{\text{G}}^T & 0 & 0 & 0 & 0 \\ 0 & S & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & Z & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta w \\ \Delta x \\ \Delta \tau \\ \Delta \nu \\ \Delta s \\ \Delta z \\ \Delta \kappa \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ r_{sw} \\ r_{xz} \\ r_{\tau\kappa} \end{pmatrix}. \quad (4.3.2)$$

As was discussed in Chapter 1, different IPMs use different right hand sides, but the coefficient matrix remains the same. In what follows we derive a surprising result that allow us to calculate $\Delta\nu$ without solving the equation system (4.3.2).

Calculating $\Delta\nu$ for the Classical Newton Directions

Let $(y, w, x, \tau, \nu, z, s, \kappa) \in \mathcal{F}_{SP}$ be the current iteration. Let $\mu_{\text{old}} = (x^T z + s^T w + \tau\kappa)/(n + m_f + 1)$, and $\theta \in [0, 1]$ be a constant. The target $\mu = (1 - \theta)\mu_{\text{old}}$. The right hand side of (4.3.2) is:

$$\begin{aligned} r_{sw} &= -sw + \mu e, \\ r_{xz} &= -xz + \mu e, \\ r_{\tau\kappa} &= -\tau\kappa + \mu e. \end{aligned} \quad (4.3.3)$$

Lemma 4.3.1. *Let $(\Delta y, \Delta w, \Delta x, \Delta\tau, \Delta\nu, \Delta s, \Delta z, \Delta\kappa)$ be a solution of the Newton system (4.3.2) with (4.3.3) as the right hand side. Then $\Delta\nu = -\theta\nu$.*

Proof. Let $\alpha_{\max}$ be the maximum step length in the Newton direction. Due to (4.2.1), for any $\alpha \in [0, \alpha_{\max}]$, we have

$$\begin{aligned} (x + \alpha\Delta x)^T (z + \alpha\Delta z) + (s + \alpha\Delta s)^T (w + \alpha\Delta w) & \\ +(\tau + \alpha\Delta\tau)(\kappa + \alpha\Delta\kappa) &= (\nu + \alpha\Delta\nu)\beta \end{aligned} \quad (4.3.4)$$

The coefficients of α in both sides must equal, thus we have

$$\begin{aligned}\Delta\nu\beta = &\ z^T\Delta x + x^T\Delta z + w^T\Delta s + s^T\Delta w + \kappa\Delta\tau + \tau\Delta\kappa \\ = &\ -x^T z - s^T w - \tau\kappa + (n + m_f + 1)(1-\theta)\mu_{\text{old}} \\ = &\ -(n + m_f + 1)\theta\mu_{\text{old}}.\end{aligned}$$

By (4.2.1) we have $\mu_{\text{old}} = (x^T z + s^T w + \tau\kappa)/(n + m_f + 1) = \beta\nu/(n + m_f + 1)$ and then the result follows directly. □

Calculating $\Delta\nu$ for the SR-Directions

Let

$$v_{sw} = \sqrt{\frac{sw}{\mu}}, \quad v_{xz} = \sqrt{\frac{xz}{\mu}}, \quad v_{\tau\kappa} = \sqrt{\frac{\tau\kappa}{\mu}}.$$

The right hand side will be:

$$\begin{aligned} r_{sw} &= -\mu v_{sw}\nabla\Psi(v_{sw}), \\ r_{xz} &= -\mu v_{xz}\nabla\Psi(v_{xz}), \\ r_{\tau\kappa} &= -\mu v_{\tau\kappa}\nabla\Psi(v_{\tau\kappa}). \end{aligned} \tag{4.3.5}$$

Lemma 4.3.2. *Let $(\Delta y, \Delta w, \Delta x, \Delta\tau, \Delta\nu, \Delta s, \Delta z, \Delta\kappa)$ be a solution of the Newton system (4.3.2) with (4.5.3) as the right hand side. Then we have*

$$\Delta\nu = -\frac{\mu}{\beta}\left[(v_{sw})^T\nabla\Psi(v_{sw}) + (v_{xz})^T\nabla\Psi(v_{xz}) + (v_{\tau\kappa})^T\nabla\Psi(v_{\tau\kappa})\right]. \tag{4.3.6}$$

Proof. From (4.2.1), for any $\alpha \in [0, \alpha_{\max}]$ we have

$$(x + \alpha\Delta x)^T(z + \alpha\Delta z) + (s + \alpha\Delta s)^T(w + \alpha\Delta w) \tag{4.3.7}$$

$$+(\tau + \alpha\Delta\tau)(\kappa + \alpha\Delta\kappa) = (\nu + \alpha\Delta\nu)\beta. \tag{4.3.8}$$

Comparing the coefficients of α on both sides, we get

$$\begin{aligned}\Delta\nu\beta = &\ z^T\Delta x + x^T\Delta z + w^T\Delta s + s^T\Delta w + \kappa\Delta\tau + \tau\Delta\kappa \\ = &\ -\mu((v_{sw})^T\nabla\Psi(v_{sw}) + (v_{xz})^T\nabla\Psi(v_{xz}) + (v_{\tau\kappa})^T\nabla\Psi(v_{\tau\kappa})).\end{aligned}$$

Then (4.3.6) follows directly. □

4.3.1 Simplifying the Newton System (4.3.2)

In this section we simplify the Newton system in order to reduce the computational cost. From the last four lines of (4.3.2), we have

$$\begin{aligned} \Delta s &= W^{-1}(r_{sw} - S\Delta w), \\ \Delta z &= X^{-1}(r_{xz} - Z\Delta x), \\ \Delta\kappa &= \tau^{-1}(r_{\tau\kappa} - \kappa\Delta\tau). \end{aligned} \tag{4.3.9}$$

Combining this with $\Delta\nu$ from the Lemma 4.3.1 or Lemma 4.3.2, the system (4.3.2) can be reduced to

$$\begin{pmatrix} 0 & 0 & A & -b \\ 0 & W^{-1}S & -F & u \\ -A^T & F^T & X^{-1}Z & c \\ b^T & -u^T & -c^T & \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta w \\ \Delta x \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} -r' \\ -r'_{sw} \\ -r'_{xz} \\ -r'_{\tau\kappa} \end{pmatrix}, \tag{4.3.10}$$

where

$$\begin{aligned} r' &= r_{\mathrm{P}1}\Delta\nu, \\ r'_{sw} &= r_{\mathrm{P}2}\Delta\nu + W^{-1}r_{sw}, \\ r'_{xz} &= r_{\mathrm{D}}\Delta\nu + X^{-1}r_{xz}, \\ r'_{\tau\kappa} &= r_{\mathrm{G}}\Delta\nu + \tau^{-1}r_{\tau\kappa}. \end{aligned}$$

From the second equation of (4.3.10), we have

$$\Delta w = WS^{-1}r'_{sw} + WS^{-1}F\Delta x - WS^{-1}u\Delta\tau, \tag{4.3.11}$$

and then system (4.3.10) can be written as

$$\begin{pmatrix} 0 & A & -b \\ -A^T & X^{-1}Z + F^TWS^{-1}F & c - F^TWS^{-1}u \\ b^T & -c^T - u^TWS^{-1}F & \frac{\kappa}{\tau} + u^TWS^{-1}u \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta x \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} r'' \\ r''_{xz} \\ r''_{\tau\kappa} \end{pmatrix},$$

where

$$\begin{aligned} r'' &= r', \\ r''_{xz} &= r'_{xz} - F^TWS^{-1}r'_{sw}, \\ r''_{\tau\kappa} &= r'_{\tau\kappa} + u^TWS^{-1}r'_{sw}. \end{aligned}$$

Since $X^{-1}Z + WS^{-1}$ is a nonsingular diagonal matrix, we can introduce the notation

$$D^2 = (X^{-1}Z + F^T WS^{-1}F)^{-1}. \tag{4.3.12}$$

Then we have

$$\begin{pmatrix} 0 & A & -b \\ A^T & -D^{-2} & c - F^T WS^{-1}u \\ b^T & -c^T - u^T WS^{-1}F & \frac{\kappa}{\tau} + u^T WS^{-1}u \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta x \\ \Delta \tau \end{pmatrix} = \begin{pmatrix} r'' \\ r''_{xz} \\ r''_{\tau\kappa} \end{pmatrix}. \tag{4.3.13}$$

This is the augmented system for the LO problem (SP). From the second equation of (4.3.13), we have

$$\Delta x = D^2 r''_{xz} + D^2 A^T \Delta y - D^2(c - F^T WS^{-1}u)\Delta\tau. \tag{4.3.14}$$

This implies that system (4.3.13) can be reduced to

$$\begin{pmatrix} ADA^T & a^1 \\ -(a^2)^T & a^3 \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta \tau \end{pmatrix} = \begin{pmatrix} r''' \\ r'''_{\tau\kappa} \end{pmatrix}, \tag{4.3.15}$$

where

$$\begin{aligned} a^1 &= -b - AD^2(c - F^T WS^{-1}u) \\ a^2 &= -b^T + (c^T + u^T WS^{-1}F)D^2 A^T \\ a^3 &= \tfrac{\kappa}{\tau} + u^T WS^{-1}u + (c^T + u^T WS^{-1}F)D^2(c - F^T WS^{-1}u) \\ r''' &= r'' - AD^2 r''_{xz} \\ r'''_{\tau\kappa} &= r''_{\tau\kappa} + (c^T + u^T WS^{-1}F)D^2 r''_{xz}. \end{aligned}$$

From the last equation of (4.3.15), we have

$$\Delta\tau = \frac{1}{a^3}(r'''_{\tau\kappa} + (a^2)^T \Delta y); \tag{4.3.16}$$

thus system (4.3.15) can be reduced to

$$\left[AD^2 A^T + \bar{a}\hat{a}^T\right] \Delta y \;\; = \xi, \tag{4.3.17}$$

where $\bar{a} = \frac{a^1}{a^3}$, $\hat{a} = a^2$, and $\xi = r''' - \frac{r'''_{\tau\kappa}}{a^3} a^1$. This is the normal equation corresponding to the problem (SP). As mentioned with respect to the enlarged

system, we can see that the difference between the normal equations of the problem in standard form (P) and those of the embedding model (SP) of the problem (LP) is one additional constraint and variable. Note that the rank-one matrix in term (4.3.17) makes the system neither symmetric nor skew-symmetric. However, the computational cost can be reduced by using the Sherman-Morrison formula.

4.3.2 Sherman-Morrison Formula

Let $P \in \mathcal{R}^{n\times n}$ be a nonsingular matrix, and let $R \in \mathcal{R}^{n\times k}$ and $S \in \mathcal{R}^{n\times k}$ be matrices with the same low rank, i.e. rank(S)=rank(R)=$k << n$. Assume $(P + RS^T)$ is nonsingular. Then the Sherman-Morrison formula [11]:

$$(P + RS^T)^{-1} = P^{-1} - P^{-1}R(I + S^T P^{-1}R)^{-1}S^T P^{-1}, \tag{4.3.18}$$

can be used to solve the system (4.3.17). Let $P = AD^2A^T$. Then the solution of system (4.3.17) can be given as

$$\Delta y = P^{-1}\xi - P^{-1}\bar{a}(I + \hat{a}^T P^{-1}\bar{a})^{-1}\hat{a}^T P^{-1}\xi. \tag{4.3.19}$$

We never compute inverse matrices, but just solve the relevant linear system. The low-rank updates strategy presented in [1] is employed to solve (4.3.17) efficiently. The method can be described as follows:

1. Solve the normal equation system $(AD^2A^T)x_0 = \xi$.

2. Solve $(AD^2A^T)x_1 = \bar{a}$.

3. Finally,

$$\Delta y = \frac{x_0 + x_0\hat{a}^T x_1 - \hat{a}^T x_0 x_1}{1 + \hat{a}^T x_1}.$$

There are several ways to solve system $AD^2A^Tx =$ RHS, e.g., the QR factorization of DA^T, an LU factorization of AD^2A^T, or a Cholesky factor-

ization of AD^2A^T. Since we assume that A has full row rank, we know that AD^2A^T is a symmetric positive definite matrix. As discussed, we need to solve the equation $AD^2A^T\Delta y = \text{RHS}$ twice with different right hand sides. The same Cholesky factorization can be used at both steps 1 and 2. Therefore Cholesky factorization of $AD^2A^T = LL^T$ is the most successful way in practice. Comparing this to the computation of the Newton steps in standard form, the total cost for the embedding algorithm is two more back substitutions for a triangular matrix and a few additional calculations for those extra variables.

If AD^2A^T is sparse, we can use a sparse linear system solver. Usually the sparse matrix package contains Cholesky factorization. However, a drawback of this algorithm is that if one or more columns of A are dense, the matrix AD^2A^T will be dense. Let us denote the i-th column of A by a_j. It can be see from

$$AD^2A^T = \sum_{j=1}^{n} d_{jj}^2 a_j a_j^T,$$

that AD^2A^T will be generally dense as long as there is a dense column in A. Clearly we need to do something to reduce the cost of solving such dense linear system.

4.3.3 Dense Columns

The standard strategy for dense columns is to separate the relatively dense columns from sparse columns. Now we show how to perform this separation. For simplicity, let us assume that all the dense columns are the last columns, thus matrix A and D^2 can be partitioned as

$$A = (A_s, A_d), \quad D^2 = \begin{pmatrix} D_s^2 & 0 \\ 0 & D_d^2 \end{pmatrix},$$

where A_s contains the sparse and A_d contains dense columns of A, and D_s^2 and D_d^2 have the same columns as A_s and A_d, respectively. Now we have

$$AD^2A^T = (A_s, A_d)\begin{pmatrix} D_s^2 & 0 \\ 0 & D_d^2 \end{pmatrix}\begin{pmatrix} A_s^T \\ A_d^T \end{pmatrix} = A_sD_s^2A_s^T + A_dD_d^2A_d^T, \qquad (4.3.20)$$

Let $P = A_sD_s^2A_s^T$ and $U = A_dD_d$. The dense matrix AD^2A^T can be split as

$$AD^2A^T = P + UU^T,$$

where P is the product of the sparse columns and U is the collection of dense columns multiplied by the corresponding diagonal elements of D. Matrix U usually has a low rank. Then the system $AD^2A^T\Delta y = \xi$ changes to

$$(P + RS^T)\Delta y = \xi, \qquad (4.3.21)$$

where $R = [U \;\; \bar{a}]$, $S^T = [U \;\; \hat{a}]^T$. If A_s is nonsingular, then P is nonsingular too. In this case, using the Sherman-Morrison formula (4.3.18), the solution of (4.3.21) is

$$\Delta y = P^{-1}\xi - P^{-1}R(I + S^TP^{-1}R)^{-1}S^TP^{-1}\xi. \qquad (4.3.22)$$

As we discussed before, $P^{-1}\xi$ can be calculated by solving the system $Px = \xi$. Similarly, we can get $P^{-1}R$ by solving the multi-right-hand-side system $Px = R$. Because the same Cholesky factors can be used, we only need multi-right-hand-side substitutions. Since R and S have low rank, the matrix $I + R^TP^{-1}S$ is dense but very small. Now we can get $(I + S^TP^{-1}R)^{-1}S^T$ by solving the system

$$(I + S^TP^{-1}R)y_0 = S^Tx_0. \qquad (4.3.23)$$

Thus the computational cost will not increase too much. In summary, if A_s is nonsingular, then the strategy for dense columns can be describe as follows:

Strategy for dense columns

1. Separate the relatively dense columns to form

$$AD^2A^T = P + UU^T,$$

where P is the sparse part and U is the collection of dense columns that has low rank.

2. Then system (4.3.17) changes to

$$(P + RS^T)\Delta y = \xi,$$

where $R = [U \quad \bar{a}]$, $S = [U \quad \hat{a}]$.

3. Solve the symmetric sparse systems

$$Px_0 = \xi \quad \text{and} \quad PX_0 = R.$$

4. Solve the dense system $(I + S^T X_0)y_0 = S^T x_0$.

5. Finally, the solution is $\Delta y = x_0 - X_0 y_0$.

The assumption in this algorithm is that P must be nonsingular. However, after separating the dense column, A_s might be singular, and then P is singular too. As a consequence, we cannot apply the above algorithm directly. To handle this case efficiently, a clever trick of K.D. Andersen [3] is needed.

4.3.4 Dealing With Singularity

To ensure the non-singularity of the sparse matrix P, a diagonal subsidiary matrix H that makes $P + HH^T$ nonsingular is needed. We can construct H by applying Gussian Elimination on matrix A_s to get the upper triangular matrix $\tilde{A}_s$. If A_s is a singular matrix, then there is at least one zero row in $\tilde{A}_s$. Assume $i_1, i_2, \ldots, i_{m_1}$, $m_1 < m$, are the indices of zero rows in matrix

$\tilde{A}_s$. Let

$$H = (h_{i_1}, h_{i_2}, \ldots, h_{i_{m_1}}),$$

where h_k is the k-th unit vector. Now let us modify the system (4.3.21) as follows:

$$[(P + HH^T) + (-HH^T + RS^T)]\Delta y = \xi. \tag{4.3.24}$$

The matrix $P + HH^T$ is nonsingular and can be constructed by:

$$P + HH^T = [A_s\ H]D_s^2[A_s\ H]^T.$$

Let $\bar{P} := P + HH^T$. Then $\bar{P}$ is a sparse symmetric positive definite matrix. System (4.3.21) becomes

$$(\bar{P} + \bar{R}\bar{S}^T)\Delta y = \xi, \tag{4.3.25}$$

where $\bar{R} = [-H\ R]$, $\bar{S} = [H\ S]$. The price we have to pay is that the dimensions of the matrices $\bar{R} = [-H\ R]$, and $\bar{S}$ are increased by the size of the matrix H from the dimensions of the matrices R and S.

4.4 Step Length

After solving the system (4.3.17), we can compute $\Delta x, \Delta z, \Delta w, \Delta s, \Delta\tau, \Delta\kappa$ from Δy. Now, to ensure that the next point will stay in the interior, we calculate the maximum step length α as follows:

$$\begin{aligned}
\alpha_x^{\max} &= \min_j\{\frac{-x_j}{\Delta x_j} : \Delta x_j < 0, j = 1, \ldots, n\};\\
\alpha_z^{\max} &= \min_j\{\frac{-z_j}{\Delta z_j} : \Delta z_i < 0, i = 1, \ldots, n\};\\
\alpha_w^{\max} &= \min_i\{\frac{-w_i}{\Delta w_i} : \Delta w_j < 0, j = 1, \ldots, m_f\};\\
\alpha_s^{\max} &= \min_i\{\frac{-s_i}{\Delta s_i} : \Delta s_i < 0, i = 1, \ldots, m_f\};\\
\alpha_{tk}^{\max} &= \min\{\frac{-\tau}{\Delta\tau},\ \ \frac{-\kappa}{\Delta\kappa} : \Delta\tau < 0, \Delta\kappa < 0\};\\
\alpha^{\max} &= \min\{\alpha_x^{\max},\ \alpha_z^{\max},\ \alpha_w^{\max},\ \alpha_s^{\max},\ \alpha_{tk}^{\max}\},\\
\alpha &= \min\{\rho\alpha^{\max},\ \ 1\},
\end{aligned} \tag{4.4.1}$$

where ρ is a damping parameter to ensure that the next point is not on the boundary.

4.5 Predictor-Corrector Technique

Now we establish formulas for Methrotra's predictor-corrector strategy [19] that we use in our implementation as well.

The first step of the predictor-corrector strategy is to compute the predictor (affine scaling) direction. The predictor direction solves the Newton equation system (4.3.2) with right hand-side (4.3.3) for $\mu = 0$. After the predictor direction, $\Delta_a x,\ \Delta_a z,\ \Delta_a s,\ \Delta_a w,\ \Delta_a \tau$, and $\Delta_a \kappa$ are computed, and the maximum feasible step size α_a along this direction is determined. Then the predicted complementarity gap

$$\begin{aligned} g_a = (x + \alpha_a \Delta_a x)^T (z + \alpha_a \Delta_a z) + (s + \alpha_a \Delta_a s)^T (w + \alpha_a \Delta_a w) \\ + (\tau + \alpha_a \Delta_a \tau)(\kappa + \alpha_a \Delta_a \kappa) \end{aligned} \tag{4.5.1}$$

is calculated, and the barrier parameter μ is estimated by the following heuristic: Let $\sigma = \left(\frac{g_a}{g}\right)^2$, then

$$\mu = \sigma \frac{g_a}{n + m_f + 1}. \tag{4.5.2}$$

Next, the second-order component of the predictor step is computed by solving system (4.3.2) with the following right hand side

$$\begin{aligned} r_{sw} &= -\mu v_{sw} \nabla \Psi(v_{sw}) - \Delta_a s \Delta_a w, \\ r_{xz} &= -\mu v_{xz} \nabla \Psi(v_{xz}) - \Delta_a x \Delta_a z, \\ r_{\tau\kappa} &= -\mu v_{\tau\kappa} \nabla \Psi(v_{\tau\kappa}) - \Delta_a \tau \Delta_a \kappa. \end{aligned} \tag{4.5.3}$$

4.6 Algorithm

In our implementation, we use a SR-based IPM with the self-dual embedding model for initialization and infeasibility detection. Further, the predictor-

corrector strategy for updating the centering parameter and calculating the search direction is used. In summary, the algorithm can be stated as follows:

SR-based IPMs with Predictor-Corrector and Embedding Strategies

---

Input:

A SR-proximity function $\Psi(v)$ and parameter δ;

An accuracy parameter $\varepsilon > 0$;

$y^0 = 0, w^0 = e, x^0 = e, \tau^0 = 1, \nu^0 = 1, s^0 = e, z^0 = e, \kappa^0 = 1$;

begin

Set up embedding model (SP) with initial point;

$y = y^0, w = w^0, x = x^0, \tau = \tau^0, \nu = \nu^0, s = s^0, z = z^0, \kappa = \kappa^0$;

while *gap* (4.1.3) $> \varepsilon$ **do**

begin

Get the affine scaling direction by solving (4.3.2)
with right hand side (4.3.3) for $\mu = 0$;

Compute the step size α_a by (4.4.1);

Get target μ by (4.5.1) and (4.5.2);

Solve (4.3.2) with right hand (4.5.3);

Compute a step size α from (4.4.1);

Update the current point

$y := y + \alpha\Delta y;\ w := w + \alpha\Delta w;\ x := x + \alpha\Delta x;\ \tau := \tau + \alpha\Delta\tau;$
$\nu := \nu + \alpha\Delta\nu;\ s := s + \alpha\Delta s;\ z := z + \alpha\Delta z;\ \kappa := \kappa + \alpha\Delta\kappa;$

end

if $\tau > 0$

$(\frac{x}{\tau}, \frac{z}{\tau}, \frac{y}{\tau}, \frac{w}{\tau}, \frac{s}{\tau})$ is a optimal solution for (LP) and (LD).

elseif $c^T x < 0$

(LD) is infeasible.

elseif $-b^T y + u^T w < 0$

(LP) is infeasible.

end

end.

First you have to choose a SR-proximity function. With different SR-proximity functions, the performances of SR-IPMs are different, as we will see in Chapter 6.

In this algorithm, the initial point is given as a unit vector. However, mentioned in Section 4.2, any vector

$$y^0,\ \ w^0 > 0,\ x^0 > 0,\ \tau^0 > 0,\ \nu^0,\ z^0 > 0,\ s^0 > 0,\ \kappa^0 > 0$$

can be taken as initial point. Also in Chapter 5.4.2 we will discuss a heuristic algorithm for a better starting point. Our implementation is based on this algorithm. Implementation details are discussed in the next chapter.

Chapter 5

Implementation Issues

A software package, called McIPM: McMaster Interior Point Method, is developed. The implementation is based on the algorithm that is discussed in detail in Chapter 4.

More than fifteen years after the publication of Karmarkar's seminal paper [13], the area of IPMs is well understood both in theory and in practice. The current implementations are sophisticated optimization tools capable of solving very large linear optimization problems. They are extremely powerful and robust, and often significantly faster than the simplex methods based codes. In this chapter we will discuss our implementation in detail.

5.1 Computational Environment

McIPM was developed on an IBM RS-6000-44p-270 workstation, with four processors, under the AIX 4.3 operating system and MATLAB[1] 6.1 environment. MATLAB is a high-level technical computing environment for numerical computation and visualization.

Our implementation is written in M-files[2] and MEX-files[3]. MEX-files are

[1]MATLAB® is the product of MathWorks Inc.. It stands for "Matrix Laboratory".
[2]M-files are MATLAB script files.
[3]MEX-files are MATLAB Executable files.

dynamically linked to subprograms compiled from C or Fortran source codes which can be called and executed from within MATLAB in the same way that one calls MATLAB M-files or built-in functions. There are external interface facilities to enable MATLAB to interact with programs written in C or Fortran, and to provide functionality for transferring data between MEX-files and MATLAB.

In McIPM, the routine tasks, mostly matrix and vector operations, are done by MATLAB's M-files. We utilize WSMP[4] [10], the Watson Sparse Matrix Package, as our large linear sparse system solver. Since we do not have the source code of WSMP, the mass data communication between MATLAB and WSMP is a major issue. The details of the design and implementation of the data transfers are discussed in Section 5.3.2. The data communication codes are MEX-files generated from C programs by the IBM C/C++ compiler xlc.

5.2 Program Structure

In McIPM LO problems are solved by a standard procedure whose global structure is shown in Figure 5.1. The following subsections explain the main steps of McIPM in more detail.

5.2.1 Input Format

McIPM is designed to solve large-scale sparse LO problems in MAT-format or in MPS-format.

[4]http://www.cs.umn.edu/~agupta/wsmp.html

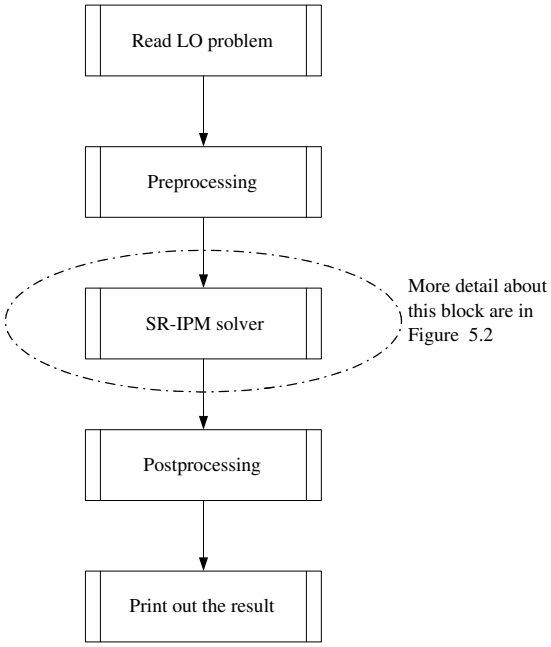


Figure 5.1: The Structure of McIPM

MAT-format

A MATLAB data file for LO in MAT-format contains the following seven quantities:

$$A,\ b,\ c,\ lbounds,\ ubounds,\ BIG,\ NAME \tag{5.2.1}$$

representing the LO problem:

$$\begin{array}{rrl} \min & c^T x & \\ \text{s.t.} & Ax & = b, \\ & lbounds \leq x & \leq ubounds. \end{array}$$

Further, *NAME* is a string storing the name of the LO problem, and *BIG* is a large number such that $ubounds_i = BIG$ if there is no upper-bound for the i-th variable x_i.

MPS-format

Currently, MPS is the default industrial input format for LO problems. LIPSOL includes an MPS reader *mps2mat*, a Fortran program to read MPS files and translate the data into MATLAB data files. In the MPS reader, a free variable is represented as the difference of two nonnegative variables. For an introduction and more details about the MPS format see e.g., [21].

The input of McIPM is based on LIPSOL[5], which is a MATLAB-based package for solving linear programs by IPMs. It uses an infeasible-primal-dual IPM. McIPM uses the LIPSOL functions *findproblem.m* and *loadata.m* to input an LO problem. The function *findproblem.m* searches for a given problem and translates it into MAT-format (if it is in MPS-format). The function *loadata.m* will load problem data (5.2.1) into the MATLAB's workspace.

5.2.2 Preprocessing and Postprocessing

One way to improve the efficiency of LO solvers is to reduce the size of the matrix A and make it sparser. This aim can be achieved by analyzing the problem (LP) and removing redundancies. In practice, almost all large-scale problems contain redundancies. There are several reasons of this phenomenon. First of all, model makers, tend to chose a formulation that is easy to understand and modify. This goal often leads to the introduction of superfluous variables and redundant constraints. The purpose of preprocessing is to reformulate the model to standard form and to reduce the problem size so as to reduce the solution time. Postprocessing recovers the original problem and transforms the solution of the preprocessed problem into a solution of the original one.

We use LIPSOL's preprocessing and postprocessing functions. The preprocessing function *preprocessing.m* performs a number of tasks, such as

[5]LIPSOL is free software under the terms of the GNU General Public License. It can be downloaded from *http://www.caam.rice.edu/~zhang/lipsol/*

checking obvious infeasibility, eliminating fixed variables, zero rows and zero columns from the matrix A, solving equations with one variable, and shifting nonzero lower bounds to zero. For a thorough description of preprocessing techniques see e.g., [23].

The postprocessing function *postprocessing.m* recovers the original problem and provides a solution for the original problem once SR-IPM solves the preprocessed problem.

5.2.3 The SR-IPM Solver

After reading in the data and preprocessing, the SR-IPM solver comes into action. The SR-IPM solver is based on the algorithm that we have discussed in Section 4.6. The general structure of our algorithm combined with the sparse linear system solver is presented in Figure 5.2. We can see that from Figure 5.2, first we need to do the following:

Compute the sparse structure of AD^2A^T: This is done in the following steps:

1. Split A into A_s and A_d, where A_s contains the sparse columns and A_d contains the dense columns of A, respectively. Based on the row numbers m of matrix A, the nonzero ratio "nzratio", i.e., the ratio used to determine if a column is dense, is defined as follows in our McIPM implementation:

$500 < m \leq 1000$	nzratio=0.2
$1000 < m \leq 5000$	nzratio=0.1
$m > 5000$	nzratio=0.05

2. Find the indices of the dense and the sparse columns.

3. Compute the sparse structure of $A_sA_s^T$ for the linear system solver WSMP.

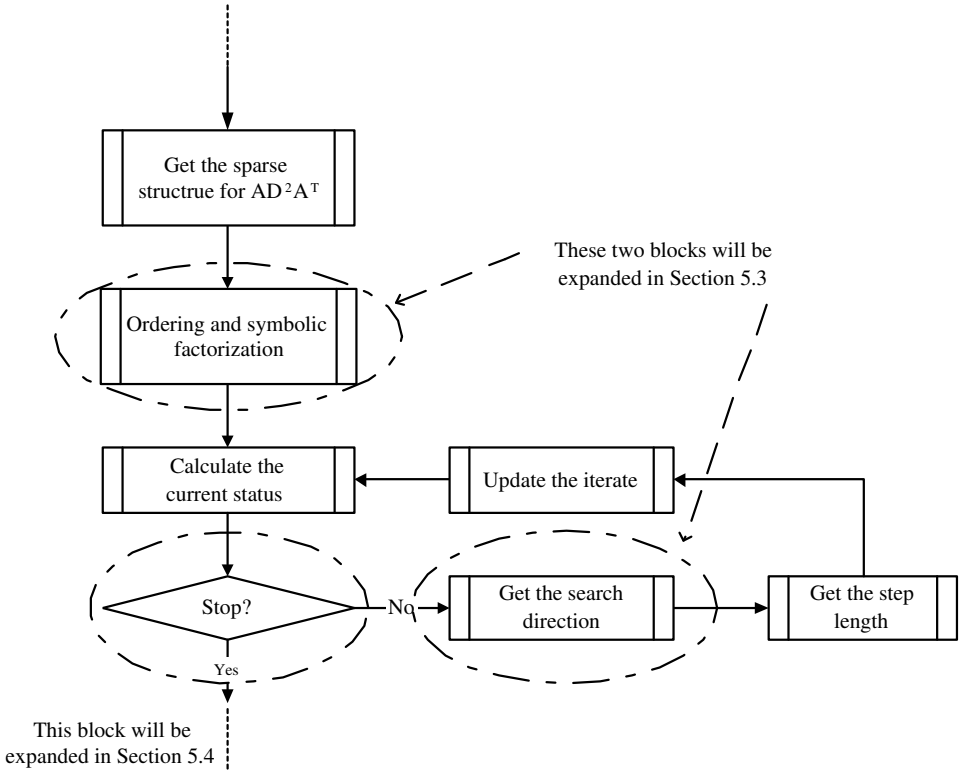


Figure 5.2: The general structure of SR-IPM

The ordering and symbolic factorization: Call the linear system solver WSMP to perform ordering and symbolic factorization. This depends on the linear system solver, therefore we will discuss about this in Section 5.3.1.

After ordering and symbolic factorization, the main loop of the algorithm begins. There are five blocks in this loop.

Calculate the current status: All information about the current point, i.e., feasibility, duality gap, and other components used in the algorithm are calculated.

Check the stopping criteria: This is another important issue in our implementation. We will discuss the details in Section 5.4.3. When

$stop = 1$, the loop ends.

Find the search direction: Use the sparse linear system solver to get the search direction. How to do this depends on which linear system solver is used, so more details will be discussed in Section 5.3. This block and the "ordering and symbolic factorization" block will be expanded to a more detailed flow chart in Figure 5.4.

Find the step length: The maximum step length is calculated by using formulas (4.4.1). Extra line search may be needed to get a better step length. More details be given in Section 5.4.4.

Update the point: Use the calculated step length and direction to update the iterate.

5.3 Calculate the Search Directions

The most important part of the implementation of our algorithm is to solve the system $Px = \text{RHS}$ where P is the symmetric matrix defined by (4.3.21). As discussed in Section 4.3.2, we need to use a sparse Cholesky factorization tool to solve this system efficiently. In practice Cholesky factorization involves three stages. First, a symmetric ordering of the rows and columns of P is chosen in an attempt to minimize the fill-in's, the new nonzero elements created during factorization. The problem of finding the ordering that minimizes the amount of fill-in is an NP-hard problem [27]. Therefore, the problem can only be solved approximately in a reasonable time. Second, one performs a symbolic factorization to figure out the sparsity pattern of L. No numerical computations are involved in these stages. Finally, the actual numerical factorization occurs using the data structures computed in the second stage. Since in IPMs the sparsity pattern of P remains the same at each iteration, so ordering and symbolic factorization are performed only once.

5.3.1 WSMP

The Watson Sparse Matrix Package[6] (WSMP) [10] is our large sparse system solver. WSMP is a high performance, robust software package for solving large sparse systems of linear equations, $Ex = B$, using direct methods, on IBM RS-6000 workstations and IBM Scalable Parallel (SP) systems.

The WSMP routine works for symmetric positive-definite, quasi-definite, and indefinite matrices whose factorization is numerically stable irrespective of the pivot sequence. For solving symmetric systems, WSMP uses a modified version of the multifrontal algorithm [16] for sparse Cholesky factorization.

Input Format of WSMP

Since the input matrix is symmetric, only the triangular portion of the sparse coefficient matrix is accepted as input. WSMP accepts two input formats for the coefficient matrix: Compressed Sparse Row/Column (CSR/CSC) and Modified Sparse Rows/Column (MSR/MSC). More precisely, CSR and CSC are denoted (respectively) as CSR-UT and CSC-LT, where the input is the upper and lower triangular part (respectively), including the diagonal, in compressed sparse row and column from coefficient matrix (respectively).

The performance of WSMP is slightly better when using the CSR-UT/CSC-LT format compared with the MSR/MSC format. We decided to use the CSC-LT format in our implementation, therefore, only the CSC-LT format is described here. If the reader is interested in other formats, he/she is referred to the WSMP manual [10]. The CSC-LT format uses two integer arrays, *IA* and *JA*, and the double precision array *AVALS* to store the sparse matrix E. Assume that N is the dimension of E, *NNZ* is the number of the total nonzero elements in the triangular portion of the symmetric matrix E, including the diagonal. Then these three arrays are:

IA: An integer array of size $N + 1$, where *IA(i)* points to the first column index of row i in the array *JA*.

[6] http://www.cs.umn.edu/~agupta/wsmp.html

JA: An integer array of size *NNZ* that contains column indices of the lower triangular part of the symmetric sparse matrix E.

AVALS: Contains the actual double precision values corresponding to the indices in *JA*. The size of *AVALS* is the same as the size of *JA*.

Functionality of WSMP

The function

wssmp(N, IA, JA, AVLAS, DIAG, PERM, INVP, B, LDB, NRHS, AUX, NAUX, MRP, IPAPM, DPARM).

is the primary routine in WSMP for solving symmetric sparse systems of linear equations. It allows the user to perform any appropriate subset of the following tasks:

1. Ordering
2. Symbolic factorization
3. Numerical factorization
4. Back substitution
5. Iterative refinement

The performance of these functions can be controlled by setting various parameters in *wssmp*. A brief explanation of the parameters of *wssmp* can be found in Appendix A. The complete description is in the WSMP manual [10]. Now let us describe those five key functions.

Ordering: The ordering routine needs the arrays *IA* and *JA*, the indices of the nonzero elements in the matrix, and some control integers, e.g. N and *NNZ*, as input, and generates a symmetric permutation of the input matrix. This permutation is designed to minimize fill during factorization and to provide ample parallelism and load-balance during message-passing or multi-threaded parallel factorization.

Symbolic factorization: Symbolic factorization calculates the possible nonzero positions in the Cholesky factorization and sets up the internal data structure to be used by the subsequent numerical factorization and solve phases. It can be performed only after ordering, and uses only the arrays *IA* and *JA*.

Numerical factorization: The numerical factorization performs either LL^T or $L\Lambda L^T$ factorization of the input matrix.

Once the symbolic factorization step has been performed, numerical factorization can be called any number of times for matrices with identical nonzero pattern but having possibly different numerical values. It needs *AVALS* as input.

Back substitution: The back substitution phase generates the actual solution to the system of linear equations. This phase requires the factor generated by the previous call of numerical factorization.

Iterative refinement: Iterative refinement can be used to improve the precision of the solution produced by the back-substitution phase. The option of using extended precision arithmetic in iterative refinement is available.

5.3.2 Data Communication between WSMP and MATLAB

Since we do not have the source code of WSMP, we cannot make MEX-files to link it to MATLAB. One simple way to do the data transfer would be to read/write data from/to disk, but this is certainly not efficient.

One creative way to solve this problem is running two communicating processes in parallel. A slave process that makes calls to WSMP runs in the background waiting for a 'command' and relevant data (for WSMP input) from the master process. The slave process can be started from MATLAB by making one call in a proper position to a piece of C code that forks this process. The MATLAB code then calls a wrapper code (instead of calling WSMP directly), which passes the data to the slave process, waits to get the

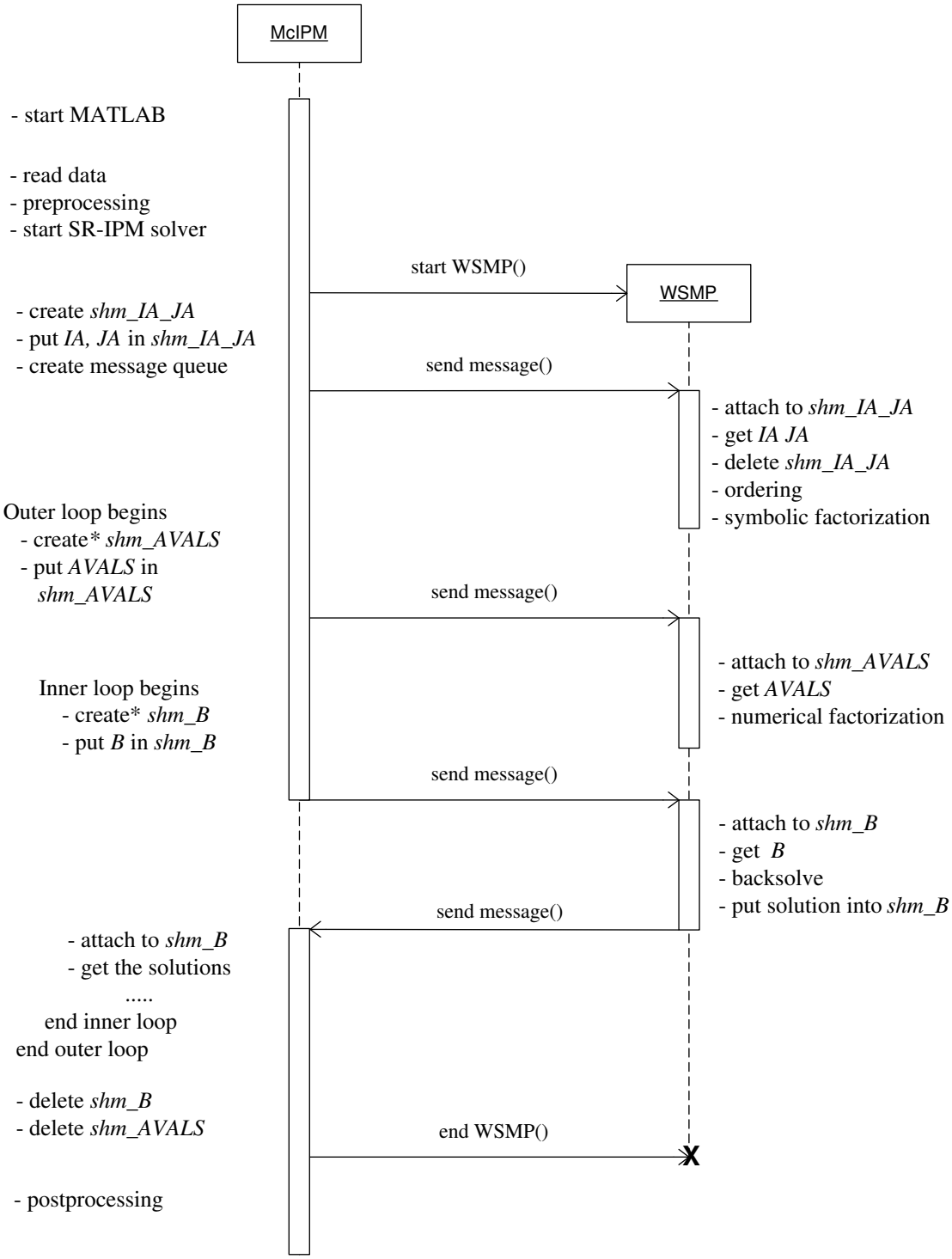
results back, and returns them to MATLAB. After finishing a request, the slave process goes back to waiting for the next command to perform some WSMP tasks. Message passing and shared memory are the communication tools that are used in our implementation. Figure 5.3 shows how data passes between these two parallel computations.

Shared memory allows two or more processes to share a given region of memory. This is the fastest form of interprocess communication (IPC), because the data does not need to be copied between the client and sever. The only trick in using shared memory is synchronizing the access to a given region among multiple processes. If the server is placing data into a shared memory region, the client should not try to access the data until the server is done. Message queues are used to control the data synchronization. These are a linked list of messages stored within the kernel and identified by a message queue identifier.

In our implementation, we need three shared memory regions for store arrays *IA, JA, AVALS,* and *B*. The integer arrays *IA* and *JA* are stored in the shared memory region *shm_IA_JA*, and the double precision array *AVALS* and *B* are in the shared memory region *shm_AVALS* and *shm_B* respectively. A message queue is created at the beginning of the program. New messages are added to the end of the queue. Every message has a positive long integer type field, *msgtype*, and the message is fetched based on its *msgtype*.

We give a more detailed flow chart of this data communication mechanism in Figure 5.4. This is an expansion of the block's "ordering and symbolic factorization" and "get the search direction" in Figure 5.2 which depends on our linear system solver WSMP. Each block in Figure 5.4 does the following:

Compute P and U: According what we have discussed in Section 4.3.3, if the matrix A has dense columns, then AD^2A^T must be split into two parts: $AD^2A^T = P + UU^T$, where P is the sparse part that we need to send to WSMP for numerical factorization and U will be used to construct the small dense system (4.3.23) which is solved by MATLAB. Further, if P is singular, then the strategy that was discussed in



* performed only at the first cycle of the loop.

Figure 5.3: Data Communication Diagram

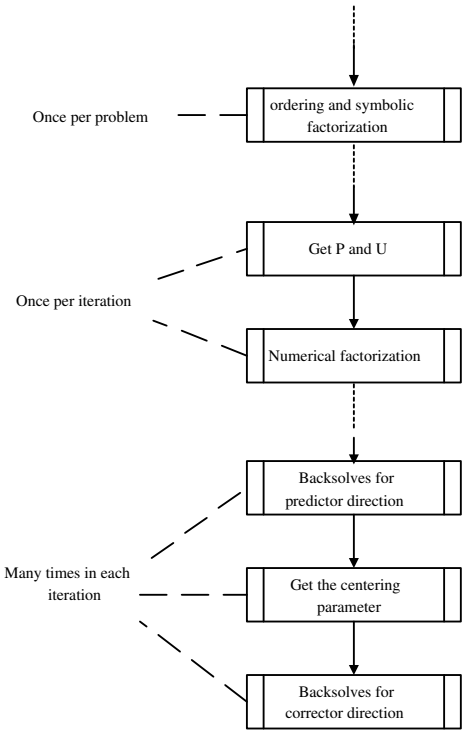


Figure 5.4: The structure of the search direction block

Section 4.3.4 is needed. The matrix that goes to WSMP for symbolic factorization is $\bar{P}$ which is defined in (4.3.25).

Ordering and symbolic factorization: This does the following:

1. According to the CSC-LT input format of WSMP that we discussed in Section 5.3.1, one generates the arrays IA, JA.
2. Create the shared memory *shm_IA_JA* for arrays IA and JA.
3. Put IA and JA in *shm_IA_JA*.
4. Create the message queue and send a message to WSMP to pick up IA and JA.
5. WSMP gets the array and does ordering and symbolic factorization.

Numerical factorization: This does the following:

1. If it is the first iteration, create a shared memory *shm_AVALA* for array *AVALS*.
2. Generate the double precision array *AVALS* for P according to the input format used in WSMP (see Section 5.3.1 for details).
3. Put *AVALS* in *shm_AVALS*.
4. Send the message to WSMP to pick up *AVALS*.
5. WSMP does the numerical factorization.

Back solves for the predictor direction: This does the following:

1. If it is the first iteration, create a shared memory region *shm_B* to store the multiple right hand vector of the linear system of equations; the same shared memory region will be used to pass the solution back.
2. Put $B = [\xi \;\; R]$, where ξ and R are defined by (4.3.17), into *shm_B*.
3. Send the message to WSMP to pick up B.
4. WSMP does the back solve and then puts the solution into *shm_B*.
5. Send the message to MATLAB to pick up the solution.
6. Use MATLAB to solve the small dense system (4.3.23).
7. Calculate the directions by using (4.3.9), (4.3.11), (4.3.14), and (4.3.16).

Find the centering parameter: The centering parameter μ can be calculated as follows:

1. Find the maximum step length as given by (4.4.1) and the predicted complementarity gap by using (4.5.1).
2. Find μ according to formula (4.5.2).

Back solve for correct direction: Same as back solve for predictor direction except that it does not need to have step 1.

5.4 More Details about the SR-IPM Solver

During the implementation, many problems need attention and handling. For example: if WSMP fails or it does not give a good solution; how to chose a better starting point for each problem; when and how to terminate the program; how to find a better step length in each iteration than that given by a fixed fraction of the maximum step length. We will discuss these below.

5.4.1 Numerical Problems

In a primal-dual IPM the Newton system to be solved at every iteration is

$$(P + RS)\Delta y = \text{RHS}.$$

The matrix P, the sparse portion of AD^2A^T, can be severely ill-conditioned or actually singular. How to handle the singularity of P has been discussed in Section 4.3.4. Still, during the Cholesky factorization, very small pivots may occurs. WSMP provides a mechanism to handle this. The parameters *IPARM*(11:12) instructs *wssmp* how to handle very small, zero, or negative (in case of LL^T factorization) pivots. The parameter *DPARM*(11:12) and *DPARM*(21:22) are four special values that can be set by the user:

DAPRM(11): *bad pivot threshold*, default $DAPRM(11) = 4 \times 10^{-16}$,

DAPRM(12): *small pivot threshold*, default $DAPRM(12) = 10^{-10}$,

DAPRM(21): *bad pivot substitute*, default $DAPRM(21) = 10^{200}$,

DAPRM(22): *small pivot substitute*, default $DAPRM(22) = 10^{-7}$,

Using these parameters, WSMP works in the following way:

- If the pivot value is less than or equal to the *bad pivot threshold*, then the pivot element will be substituted by the *bad pivot substitute*.
- If the pivot value is less than or equal to the *small pivot threshold*, then the pivot element will be substituted by the *small pivot substitute*.

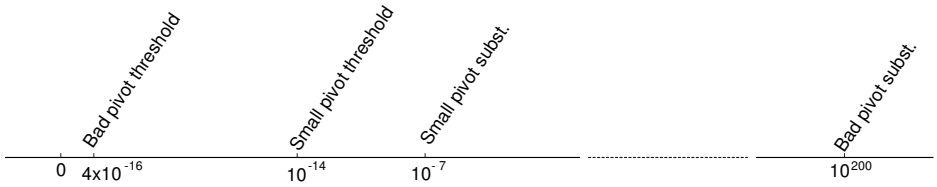


Figure 5.5: Handling bad pivots

WSMP may still give us a bad result e.g. *NAN* (not a number). Even in this case, the program should then stop and give a reasonable message. In our implementation, we use *optimal* to keep track of the stop status. If *NAN* occurs, then the *optimal* is set to -2 to indicate the numerical difficulty encountered. For normal termination, the value of *optimal* is set to 1 and 0, for an optimal solution found or infeasibility detected respectively. If the maximum iteration is exceeded during the programming, then it will be stopped with $optimal = -1$. This can be seen in more detail in the flow chart in Figure 5.7.

5.4.2 Starting Point

The simple starting point

$$(y, w, x, \tau, \nu, z, s, \kappa) := (0, e, e, 1, 1, e, e, 1), \tag{5.4.1}$$

provides reasonable performance in most cases. However, the performance of the algorithm can be improved by using the following starting point, suggested by E.D.Anderson [2].

The algorithm for starting point:

1. Choose an arbitrary initial point $(y, w, x, \tau, \nu, z, s, \kappa)$ that satisfying IPC,

   for example the point (5.4.1).

2. Solve system (4.3.2) with right hand side (4.3.3) and force $\Delta\nu = 1$;

   Let $(d_y^1, d_w^1, d_x^1, d_\tau^1, d_\nu^1, d_z^1, d_s^1, d_\kappa^1)$ be the resulting direction.

3. Get the maximal possible step size α in the direction $(d_y^1, d_w^1, d_x^1, d_\tau^1, d_\nu^1, d_z^1, d_s^1, d_\kappa^1)$.

4. Solve system (4.3.2) with $\bar{\mu} = 10\mu$,

   where the last three equations are replaced by

$$\begin{aligned} Wd_s^2 + Sd_w^2 &= -Ws + (1-\alpha)\bar{\mu}e - \alpha^2 d_w^1 d_s^1, \\ Xd_z^2 + Zd_x^2 &= -Xz + (1-\alpha)\bar{\mu}e - \alpha^2 d_x^1 d_z^1, \\ \kappa d_\tau^2 + \tau d_\kappa^2 &= -\kappa\tau + (1-\alpha)\bar{\mu}e - \alpha^2 d_\tau^1 d_\kappa^1. \end{aligned} \tag{5.4.2}$$

   Let $(d_y^2, d_w^2, d_x^2, d_\tau^2, d_\nu^2, d_z^2, d_s^2, d_\kappa^2)$ be the resulting direction.

5. The initial point is obtained by

$$\begin{aligned} y^0 &:= y + d_y^2, \\ w^0 &:= \max(1, w + d_w^2), \\ x^0 &:= \max(1, x + d_x^2), \\ \tau^0 &:= \max(1, \tau + d_\tau^2), \\ \nu^0 &:= \nu + d_\nu^2, \\ s^0 &:= \max(1, s + d_s^2), \\ \kappa^0 &:= \max(1, \kappa + d_\kappa^2). \end{aligned} \tag{5.4.3}$$

6. If this point is not in the required neighborhood, it is modified to stay in the neighborhood.

Observe that the computational cost of obtaining this starting point is equivalent to the cost of one interior-point iteration.

5.4.3 Stopping Criteria

An important issue is when to terminate the algorithm and how to give the conclusion solution based on the information obtained from the last iterate. Clearly, the algorithm cannot be terminated before a feasible optimal solution of the homogeneous model has been obtained. Feasibility for the embedding model is preserved during the algorithm, therefore we define only the following measures:

tol_emd: The tolerance of the duality gap of the embedding problem.

tol_ori: The tolerance of the duality gap of the original problem.

tol_kt : Due to the embedding structure, Theorems 4.2.1 and 4.2.2 say that one of τ and κ must be zero and the other one must be positive. So *tol_kt* is the tolerance to determine which one is zero in the solution of (SP).

tol_end: The end tolerance of the gap of the embedding problem. Due to the numerical reasons, this is the smallest value we want to reach for duality gap of the embedding model.

Define

$$\begin{array}{lll} fea\_p & = & \frac{\|Ax-b\tau\|+\|Fx+s-u\tau\|}{\tau+\|x\|}, \\ fea\_d & = & \frac{\|A^Ty+z-c\tau-F^Tw\|}{\tau+\|z\|}, \\ gap\_ori & = & \frac{\|c^T-b^Ty+u^Tw\|}{\tau+\|b^Ty-u^Tw\|}, \end{array} \tag{5.4.4}$$

and

$$error = \max\{fea\_p,\ fea\_d,\ gap\_ori\}.$$

Due to the limitation of machine precision, we introduce firm optimal, non-firm optimal, firm infeasible, and non-firm infeasible outcomes. The

algorithm will terminate with firm optimal if $error < tol\_ori$ and $\kappa < \tau * tol\_kt$. In this case a solution

$$(x^*, z^*, y^*, w^*, s^*) = (x^k, z^k, y^k, w^k, s^k)/\tau^k$$

is reported to be the optimal solution for (P). The algorithm will also be terminate if $\mu < tol\_emd$ and $\tau < \kappa * tol\_kt$. In this case a feasible solution to the homogeneous model with a small τ has been computed. Therefore the problem is primal or dual infeasible. Moreover, the algorithm is terminated if $\mu < tol\_end$, and reports the result up to certain precision. Figure 5.6 presents the complete structure of our subroutine *stopping.m*.

5.4.4 Line Search and Dynamic SR-IPM

At the beginning of our algorithm, due to the use of the embedding model, the initial point can be chosen as well centered and even on the central path. Therefore the Newton direction is the best choice for the start of the algorithm. Later, a point might get closer to the boundary, thus one might need a better barrier property when calculating the search direction. Therefore we decided to use $q = 1$ in the SR-search direction at the beginning of the iteration process, and then dynamically increase q if the position of the current iterate is near the boundary. An inexact line search is used here, called "backtracking". It is implemented by *backtracking.m*. If too many coordinates of v are very close to zero then we cut back the step length by the fractions [0.9975, 0.95, 0.90, 0.75, 0.50] until there are not too many very small coordinates for the next iterate. We only use backtracking if the step length is satisfactory and $q = 1$. In the case that the step length is very small, e.g. $\alpha < 0.005$, the search direction will change to use the SR-proximity function with higher q, which will lead to a centering direction. Another line search, *linesearch.m*, is used for the direction with higher q to ensure the current iterate stays in a certain proximity neighborhood or not too close to the boundary. After a successful higher order step the value of q is reset to 1, and the iteration is continued. Figure 5.7 gives a complete flow chart of this dynamic SR-IPM.

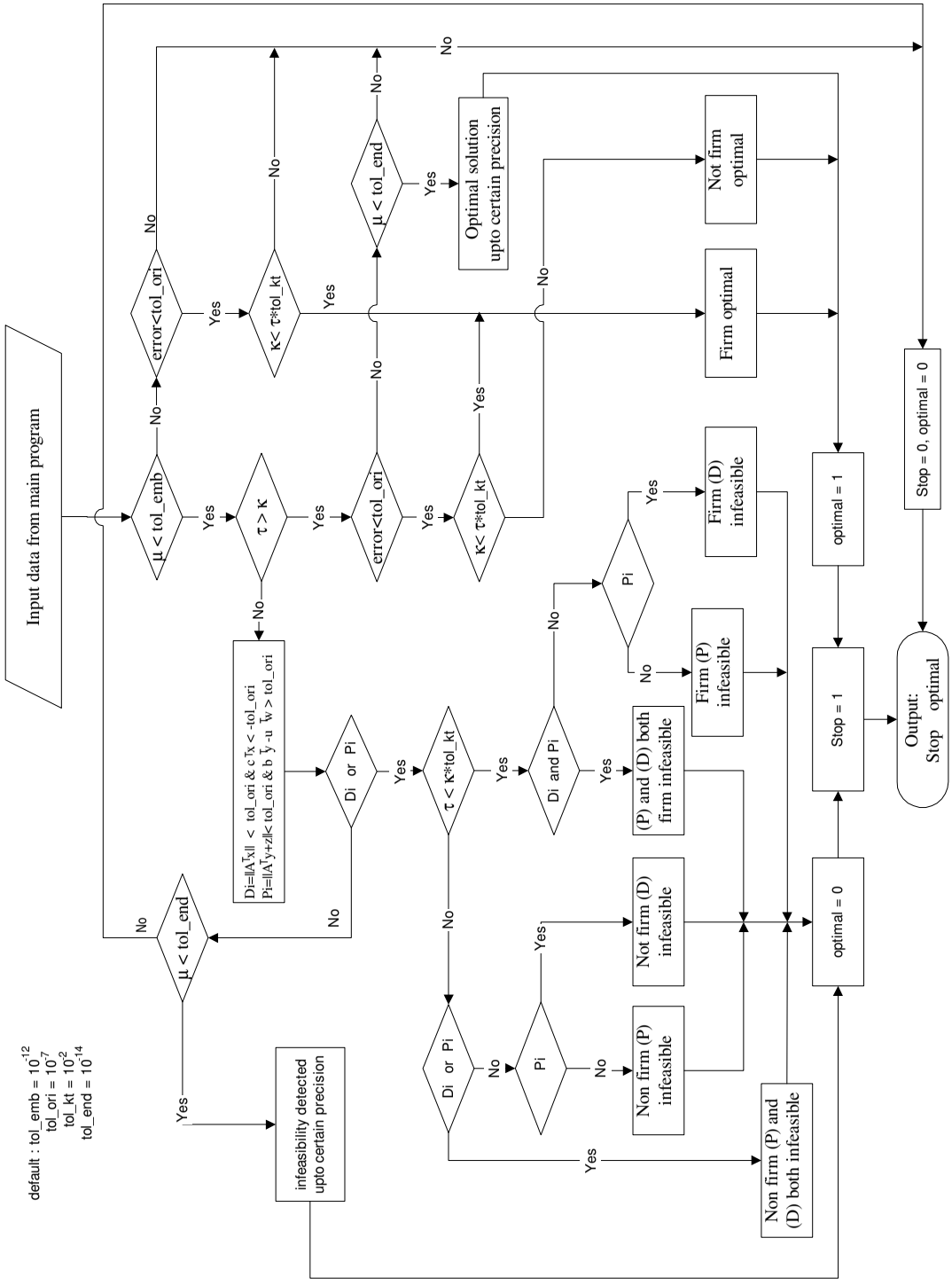


Figure 5.6: Flow chart of stopping.m

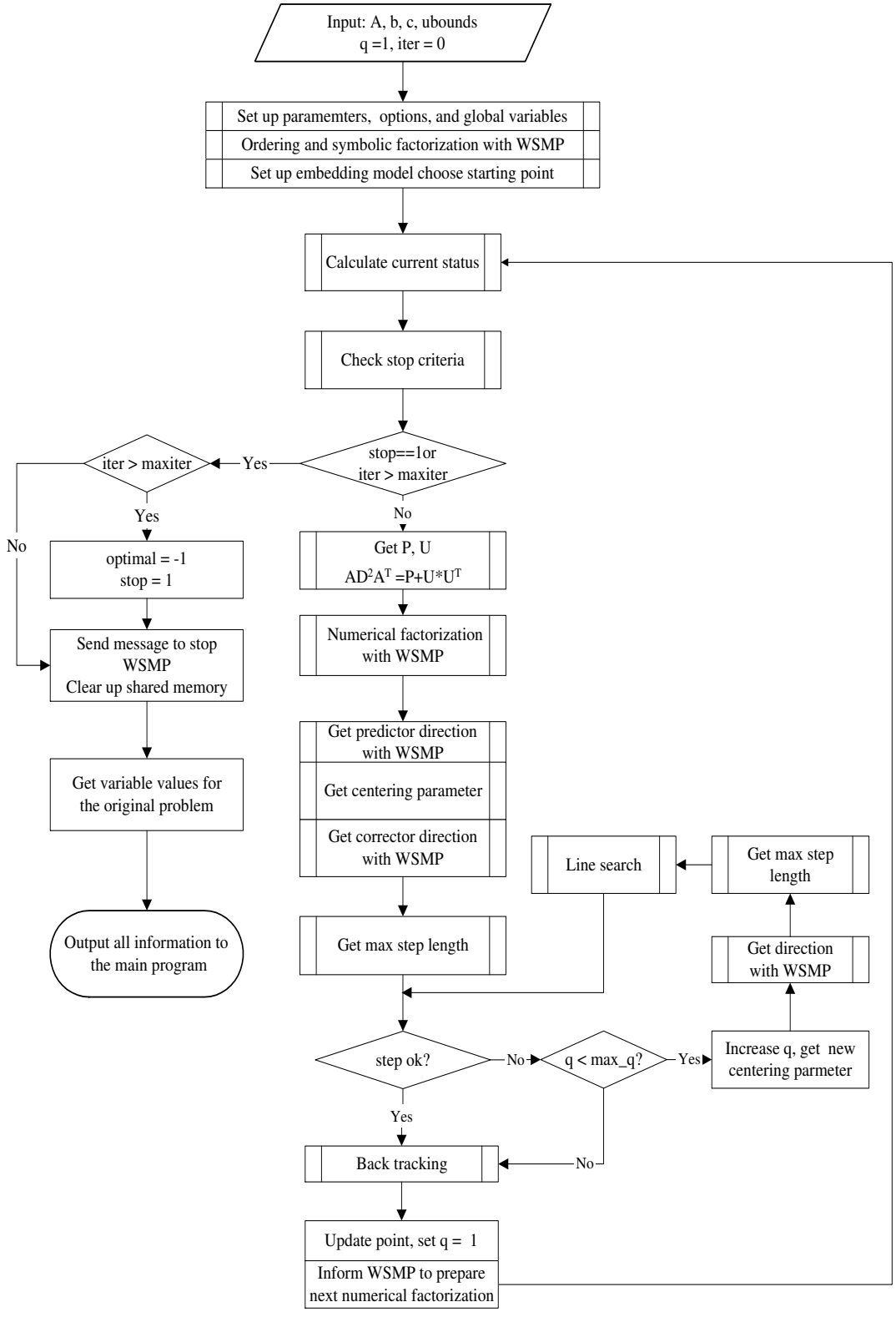


Figure 5.7: Dynamic SR-IPMs

5.5 How to Use McIPM

Type: *mcipm* in the *mcipm* directory at the system prompt. That will let you enter the MATLAB environment. At the MATLAB prompt ">>" once again enter *mcipm*. McIPM will prompt you for a problem name and then proceed to solve the problem.

For example, to run the problem "afiro", you should do

```
>> mcipm
Enter problem name: afiro
```

Alternatively, you could enter *mcipm('afiro')* or *mcipm afiro* at once. Note that a *problem name* is not a file name, and should not contain any dot. In the above example, the actual file storing the problem "afiro" may be *afiro.mps.gz*, or *afrio.mps*, or *afiro.mat*.

McIPM supports two LO input formats: MAT and MPS. There are thus two directories for storing problems: *matdir* and *mpsdir*. In general, a MAT-file should have a file extension *.mat* and be stored in the directory *matdir*; similarly for MPS-files. However, one need not use these default directories and can store problems in any directory as described in the following paragraphs.

McIPM can also read problems from directories other than the above two, as long as they are specified in the configuration function *ipmconfig.m* in the directory *mcipm/solvers/command* as entries of the string vector *PROBDIR*. The first entry of *PROBDIR* is an empty string defined as:

```
PROBDIR=str2mat([]);
```

Do not alter it. If you want McIPM to access problems stored in the directory, e.g., "/usr/local/mpsdir", then add an entry to *PROBDIR* following the first entry:

```
PROBDIR=str2mat(PROBDIR,'/usr/local/mpsdir');
```

(You can add more than one directory, incidentally). To ensure proper access, the full path of a directory should be specified the same way as in the above example. If multiple files exist with the same problem name, for example, *agg.mat* and *agg.mps*, McIPM will choose *agg.mat* to solve that is the first according to the order of MAT and MPS. All problem files can be either uncompressed, or compressed by *gzip* or by the UNIX *compress*.

Chapter 6

Computational Experience

In this section, we describe our numerical experiments and present our computational results. Our numerical experiments were performed on an IBM RS/6000 44P Model 270 Workstation. It is a symmetric multiprocessor (SMP) workstation with up to four processors and 8GB memory.

6.1 Testing Results

The test problems used in our experiments come from the Netlib set of LO problems. This is an electronically distributed set of LO problems collected and maintained by David Gay [6] at AT&T Bell Laboratories. As of February 2002, the collection consists of three test sets:

- The standard test set, Netlib-Standard, containing 95 problems, all of which have primal-dual optimal solutions.

- The infeasible test set, Netlib-Infeasible, containing 28 problems for which there are of course no solutions.

- The Kennington test set, Netlib-Kennington, containing 16 larger scale problems arising from military aircraft applications.

The size of the problems in the Netlib set range from a few dozen variables to over fifteen thousand. Over the years, the Netlib set has become the standard set of test problems for testing and comparing algorithms and software for linear programming.

All three problem test sets from Netlib were tested. The results are shown in Tables 6.1 to 6.6. The parameter setting used to get those solutions is[1] as follows:

```
% PARAMETERS  - Algorithmic parameters for McIPM
p          = 1;
q          = 1;
MAXITER    = 99;           % Maximum number of iteration
tol_emb    = 4.0e-13;      % an accuracy parameter for nu
tol_ori    = 1.0e-9;       % an accuracy parameter for gap_ori
tol_end    = 1.e-18;       % an accuracy parameter for nu
tol_bad    = 1.e-20;
tol_kt     = 1.e-2;        % an accuracy parameter for tau/ka
damp       = 0.995;        % a variable damping parameter
bad_step   = 0.1;
worst_step= 0.05;
MAXQ       = 3;
startp     = 0.5;
```

In each table, seven columns are given, the problem name: the number of rows and columns in matrix A after preprocessing[2], the number of iterations required for convergence, the maximin relative error (p: primal infeasibility, d: dual infeasibility, g: the primal-dual gap), the primal objective value, and the CPU seconds, respectively.

All problems in the Netlib-Standard test set have been solved. The results show that our algorithm exhibits high precision solutions for most of the problems. For some of them, however, e.g., *fit2d* and *fit2p* in Table **??**, we still have difficulty in getting highly precise solutions.

There are 28 problems in the Netlib-infeasible set. Two of them, *gran* and *woodinfe*, were solved in the preprocessing stage. The problem *ceria3d*

[1]The parameter *tol_ori* is chosen from *4.0e-6* to *1.0e-11* for each problem to get highly accurate solutions.

[2]For the number of constraints and the number of variables in the unprocessed model, please see Appendix B

Table 6.1: Results for the Netlib-Standard Problems (I)

Problem	Rows	Cols	Iter	Residual	Primal objective	CPU sec.
25fv47	798	1854	30	2.04363e-10 d	5.5018458985e+03	10.53
80bau3b	2235	11516	43	3.97010e-09 p	9.8722419247e+05	27.00
adlittle	55	137	15	5.78593e-10 p	2.2549496305e+05	0.19
afiro	27	51	10	3.33772e-12 p	-4.6475314285e+02	0.59
agg	488	615	26	4.36317e-11 p	-3.5991767285e+07	7.70
agg2	516	758	19	8.97628e-12 p	-2.0239252356e+07	8.77
agg3	516	758	25	8.77039e-11 p	1.0312115935e+07	8.52
bandm	269	436	18	8.43110e-12 g	-1.5862801840e+02	2.59
beaconfd	148	270	15	1.23708e-12 d	3.3592485808e+04	1.59
blend	74	114	13	6.44693e-12 d	-3.0812149844e+01	1.12
bnl1	632	1576	39	1.13651e-12 p	1.9776295615e+03	5.79
bnl2	2268	4430	45	4.23242e-11 p	1.8112365422e+03	23.64
boeing1	347	722	26	1.84524e-08 g	-3.3521349467e+02	4.42
boeing2	140	279	20	1.29064e-08 g	-3.1501828336e+02	2.29
bore3d	199	300	20	1.11974e-10 g	1.3730803948e+03	2.20
brandy	149	259	19	6.28859e-12 g	1.5185098967e+03	2.21
capri	267	476	18	5.37504e-09 p	2.6900133717e+03	2.69
cycle	1801	3305	42	4.86760e-10 g	-5.2263930157e+00	31.68
czprob	737	3141	34	6.25627e-10 g	2.1851966963e+06	42.61
d2q06c	2171	5831	51	1.23888e-10 d	1.2278421120e+05	33.87
d6cube	404	6184	19	3.06025e-09 p	3.1549166672e+02	10.19
degen2	444	757	14	7.81046e-12 p	-1.4351780000e+03	2.84
degen3	1503	2604	15	8.39681e-12 p	-9.8729400000e+02	21.56
dfl001	6071	12230	53	4.65262e-09 g	1.1266396083e+07	92.06
e226	220	469	22	7.44573e-12 d	-1.8751929063e+01	2.88
etamacro	357	692	31	1.88848e-09 g	-7.5571523329e+02	3.85
fffff800	501	1005	36	2.75467e-11 p	5.5567956517e+05	6.77
finnis	492	1014	31	4.27625e-10 d	1.7279106561e+05	3.29
fit1d	24	1049	24	9.66457e-11 g	-9.1463780916e+03	3.38
fit1p	627	1677	15	2.59566e-06 p	9.1463781202e+03	73.90
fit2d	25	10524	21	3.00026e-10 g	-6.8464293279e+04	14.86

Table 6.2: Results for the Netlib-Standard Problems (II)

Problem	Rows	Cols	Iter	Residual	Primal objective	CPU sec.
fit2p	627	1677	19	3.65069e-06 p	6.8464295930e+04	22.56
forplan	135	463	34	2.20971e-10 g	-6.6421886812e+02	3.87
ganges	1137	1534	24	4.19426e-09 g	-1.0958573531e+05	8.09
gfrd-pnc	600	1144	17	2.00079e-10 d	6.9022365407e+06	2.17
greenbea	2318	5424	55	7.34298e-10 g	-7.2552727625e+07	50.07
greenbeb	2317	5415	45	1.04160e-08 p	-4.3022601934e+06	39.68
grow15	300	645	22	7.51468e-11 g	-1.0687094128e+08	2.59
grow22	440	946	21	4.59065e-11 g	-1.6083402847e+08	3.33
grow7	140	301	20	7.33895e-11 g	-4.7787811811e+07	1.42
israel	174	316	26	6.21945e-11 d	-8.9664482176e+05	5.51
kb2	43	68	21	6.36373e-10 d	-1.7499001295e+03	0.83
lotfi	151	364	27	1.46431e-10 g	-2.5264705311e+01	1.36
maros	835	1921	27	1.01829e-10 g	-5.8063737934e+04	9.07
maros-r7	3136	9408	20	1.85557e-10 p	1.4971851666e+06	190.44
modszk1	686	1622	33	8.05082e-11 g	3.2061975749e+02	7.07
nesm	654	2922	33	2.51576e-08 p	1.4076036203e+07	8.05
perold	625	1530	45	1.52835e-08 g	-9.3807550575e+03	10.08
pilot	1441	4657	56	1.42740e-08 g	-5.5748972003e+02	80.01
pilotja	924	2044	46	1.23465e-08 g	-6.1131363663e+03	18.94
pilotwe	722	2930	43	1.45902e-08 p	-2.7201075178e+06	8.15
pilot4	402	1173	40	7.52342e-10 g	-2.5811392560e+03	8.74
pilot87	2030	6460	71	9.32361e-08 g	3.0171046778e+02	196.99
pilotnov	951	2242	29	1.83604e-08 g	-4.4972761820e+03	11.21
recipe	85	177	12	5.34138e-08 g	-2.6661599644e+02	1.18
sc105	105	163	16	6.48185e-12 g	-5.2202061186e+01	0.65
sc205	205	317	17	6.32452e-12 g	-5.2202061170e+01	1.41
sc50a	49	77	11	1.20036e-11 d	-6.4575077043e+01	0.87
sc50b	48	76	10	1.24306e-11 d	-6.9999999984e+01	0.79
scagr25	471	671	17	8.09252e-10 p	-1.4753433020e+07	2.68
scagr7	129	185	13	2.33609e-09 p	-2.3313898207e+06	1.17
scfxm1	322	592	27	1.29445e-11 p	1.8416759031e+04	3.52
scfxm2	644	1184	29	9.47906e-13 p	3.6660261565e+04	6.36
scfxm3	966	1776	28	1.06417e-11 p	5.4901254556e+04	7.32

Table 6.3: Results for the Netlib-Standard Problems (III)

Problem	Rows	Cols	Iter	Residual	Primal objective	CPU sec.
scorpion	375	453	14	2.19173e-10 g	1.8781248224e+03	1.43
scrs8	485	1270	24	8.45000e-10 p	9.0429721807e+02	2.77
scsd1	77	760	9	3.52144e-09 g	8.6666667066e+00	0.79
scsd6	147	1350	12	2.18289e-09 g	5.0500000176e+01	1.25
scsd8	397	2750	9	1.24882e-08 g	9.0500000920e+02	1.88
sctap1	300	660	19	1.25850e-08 g	1.4122500116e+03	1.63
sctap2	1090	2500	13	2.83764e-08 g	1.7248071880e+03	3.75
sctap3	1480	3340	13	1.94000e-09 g	1.4240001057e+03	5.46
seba	515	1036	29	2.18827e-09 g	1.5711600036e+04	33.35
share1b	112	248	30	1.31246e-09 d	-7.6589304082e+04	1.50
share2b	96	162	12	1.44747e-09 d	-4.1573222002e+02	0.77
shell	496	1487	24	5.06524e-11 p	1.2088253460e+09	3.11
ship04l	356	2162	16	5.14199e-11 d	1.7933245380e+06	2.88
ship04s	268	1414	17	9.45476e-09 d	1.7987146920e+06	2.00
ship08l	688	4339	19	1.19372e-08 g	1.9090552093e+06	6.12
ship08s	416	2171	19	1.36799e-09 p	1.9200982085e+06	4.31
ship12l	838	5329	28	2.93064e-11 g	1.4701879193e+06	9.93
ship12s	466	2293	22	1.27323e-09 d	1.4892361304e+06	3.85
sierra	1222	2715	20	9.81134e-10 g	1.5394362175e+07	5.57
stair	356	538	15	2.07812e-10 p	-2.5126694659e+02	2.96
standata	359	1258	15	2.72321e-10 p	1.2576995001e+03	1.77
standgub	361	1366	15	3.15581e-12 p	1.2576995000e+03	1.79
standmps	467	1258	20	7.17416e-11 p	1.4060175000e+03	3.34
stocfor1	109	157	16	4.95938e-10 p	-4.1131976203e+04	0.81
stocfor2	2157	3045	31	2.34807e-09 d	-3.9024406453e+04	14.16
stocfor3	16675	23541	50	3.11401e-10 d	-3.9976783296e+04	254.87
truss	1000	8806	17	1.24089e-09 d	4.5881584641e+05	8.98
tuff	292	617	25	7.46744e-10 g	2.9214777306e-01	3.16
vtpbase	194	325	17	2.07140e-10 p	1.2983146244e+05	1.52
wood1p	244	2595	18	1.58995e-11 p	1.4429024116e+00	9.29
woodw	1098	8418	22	3.09570e-07 p	1.3044768851e+00	17.44

Table 6.4: Results for the Netlib-Infeasible Problems

Problem	Rows	Cols	nonzero	Iter	feasibility	Residual	CPU sec.
bgdbg1	348	629	1662	8	(LP) infeasible	4.60204e-10	1.69
bgetam	357	692	2004	7	(LP) infeasible	4.59429e-10	1.74
bgprtr	20	40	70	10	(LP) infeasible	1.57589e-11	0.93
box1	231	261	651	6	(LP) infeasible	5.69929e-10	1.12
chemcom	288	744	1590	6	(LP) is infeasible	1.99079e-10	1.40
cplex1	3005	5224	10947	16	(LP) is infeasible	2.43980e-11	11.28
cplex2	224	378	1215	52	optimal*	2.41438e-10	2.18
ex72a	197	215	682	6	(LP) is infeasible	2.71528e-10	1.05
ex73a	193	211	457	6	(LP) is infeasible	1.26073e-10	1.12
forest6	66	131	246	7	(LP) is infeasible	7.14199e-09	0.98
galenet	8	14	22	5	(LP) is infeasible	8.24488e-10	0.80
gosh	3718	13625	100566	37	(LP) is infeasible	1.26896e-09	109.28
gran	infeasibility detected at preprocessor						
greenbea	2319	5419	30425	22	(LP) is infeasible	6.47042e-14	24.97
itest2	9	13	26	5	(LP) is infeasible	1.65205e-10	0.77
itest6	11	17	29	5	(LP) is infeasible	1.55372e-09	0.81
klein1	54	108	750	22	(LP) is infeasible	2.88358e-13	1.80
klein2	477	531	5062	15	(LP) is infeasible	4.22242e-10	24.93
klein3	994	1082	13101	17	(LP) is infeasible	3.05186e-10	125.79
mondou2	259	467	934	11	(LP) is infeasible	5.03555e-12	1.57
pang	357	727	2932	34	(LP) is infeasible	1.00058e-13	5.59
pilot4i	402	1173	7226	16	(LP) is infeasible	3.24427e-11	4.39
qual	323	459	1633	38	(LP) is infeasible	3.88660e-13	4.96
reactor	318	806	2589	20	Both are infeasible	1.76218e-12	3.07
refinery	319	464	1600	13	(LP) is infeasible	1.61327e-09	2.31
vol1	323	459	1633	30	(LP) is infeasible	3.22592e-13	4.16
woodinfe	infeasibility detected at preprocessor						

* McIPM gives an optimal solution to this problem, see Table 6.5 for more details.

Table 6.5: Results for the Netlib-Infeasible problem *cplex2*

	fea-p	fea-d	gap	Primal objective
LIPSOL	1.08e-06	1.56e-13	5.55e-17	6.5513713512e-01
McIPM	6.90652e-08	2.21169e-16	1.51820e-09	6.5680048168e-01

Table 6.6: Results for the Netlib-Kennington Problems

Problem	Rows	Cols	nonzero	Iter	Residual	Primal objective	CPU sec.
cre-a	3428	7248	18168	27	2.55023e-08 d	2.3595407173e+07	26.15
cre-b	7240	77137	260785	35	2.39033e-08 d	2.3129640016e+07	288.66
cre-c	2986	6411	15977	31	2.82112e-08 d	2.5275116200e+07	25.21
cre-d	6476	73948	246614	33	2.22400e-08 d	2.4454969999e+07	254.79
ken-07	1691	2867	6640	16	4.08777e-09 p	-6.7952044125e+08	7.91
ken-11	11548	18203	42161	20	1.37126e-08	-6.9723821803e+09	89.50
ken-13	23393	37420	84909	32	1.12311e-08 p	-1.0257394782e+10	316.78
ken-18	MPS reader mps2mat error						
osa-07	1118	25067	144812	31	8.07157e-09 g	5.3572252146e+05	66.18
osa-14	2337	54797	317097	39	1.52717e-08	1.1064628610e+06	179.11
osa-30	4350	104374	604488	48	3.47687e-09 g	2.1421398804e+06	417.14
osa-60	MPS reader mps2mat error						
pds-02	2788	7551	16230	41	6.73e-8 p	2.8857860783e+10	24.43
pds-06	9617	29087	62582	60	7.69e-08	2.7761036410e+10	171.72
pds-10	16239	49613	106802	72	6.66e-9 p	2.6727094856e+10	364.44
pds-20	33250	107627	231155	get NAN from WSMP at iteration 39			

is still under testing. LIPSOL failed at iteration 9. For problem *cplex2*, our algorithm failed to report infeasibility; instead it gave an optimal solution in the same way as LIPSOL. The details of our results on *cplex2* including primal infeasibility, dual infeasibility, and duality gap are reported in Table 6.5. For the remaining 24 problems, we get an infeasibility certificate, while LIPSOL gives an infeasibility certificate only for 12 problems[3].

In the Netlib-Kennington set, the MPS reader *mps2mat* has difficulty with problems *ken-18* and *osa-60*, thus both LIPSOL and McIPM are unable to solve them. This means we need a better MPS file reader in the future. Problem *pds-20* has numerical difficulty at iteration 39 when the search direction was calculated by WSSMP. All the other thirteen problems were solved with reasonable iteration numbers and precisions. The results are shown in Table 6.6.

6.2 Testing Different SR-functions

We have tested four ways to choose the best SR-proximity function. One is called "dynamic update" where the value of q changes during the iterations. This variant is discussed in Section 5.4.4. The other three variants work with fixed q values: q= 1, 2, or 3. Tables 6.7 - 6.11 show the test results. In the dynamic case, the column *Flag-q* indicates if $q > 1$ was used or not. The column "digits" shows how many digits agree with the the standard solution[4] from Netlib. The parameter setting used to get these solutions is[5] as follows

```
% PARAMETERS  - Algorithmic parameters for McIPM
p          = 1;
q          = 1, 2, or 3;
MAXITER    = 99;
tol_emb    = 4.0e-13;    % an accuracy parameter for nu
tol_ori    = 1.0e-7;     % an accuracy parameter for gap_ori
```

[3]See Table 6.15

[4]For each problem you can find the standard solution in Appendix B

[5]The parameter *tol_ori* is chosen from *4.0e-6* to *1.0e-11* for each problem to get the most accurate solutions.

```
tol_end   = 1.e-18;     % an accuracy parameter for nu
tol_bad   = 1.e-20;
tol_kt    = 1.e-2;      % an accuracy parameter for tau/ka
damp      = 0.995;      % a variable damping parameter
bad_step  = 0.1;
worst_step= 0.05;
MAXQ      = 1, 2, or 3;
startp    =0.5;
```

For evaluating the effect of $q > 1$, we present all problems with $q > 1$ in the dynamic algorithm in Table 6.12.

Based on the results in Table 6.12 we can conclude that our dynamic algorithm needs fewer iterations for problems *bnl1*, *pds-06*, and *shell* than with the classical IPM algorithm ($q = 1$), while the precision of the solution remains the same. We get higher precision for problem *ship12l* with the same iteration number. Problems *bored3d*, *pilot*, *pilot87*, *gosh*, *osa-14*, and *ship08l* have the same performance as when $q = 1$. For other problems, our iteration number is slightly higher than the classical IPM algorithm.

In the case $q = 2$. (compared with the classical algorithm $q = 1$), we can say that our algorithm still works well and requires fewer iterations for the eight problems *greenbea, perold, pilot, shell, pds-02, pds-06*, and *pds-10*. With three problems, *bnl1, dfl001,* and *osa-30*, the number of iterations and the precision are the same. Problem *ship12s* has one more iteration than for the case $q = 1$, but it get two more correct digits. Some problems, e.g., *bore3d*, have comparable results.

For $q = 3$, our algorithm is still stable and solves all problems that can be solved by others. Problem *pds-06* was solved by 56 iterations, which is better than the classical algorithm. Two problems keep the same number of iterations and precision. All others require more iterations than for the case $q = 1$.

The classical algorithm ($q = 1$) failed to solve problem *pds-06*, but the algorithm with higher q or the dynamic case, generates an optimal solution in a reasonable number of iterations.

Based on these results we may conclude that a higher value of q may improve the performance of IPMs and can even help to solve some problems.

Table 6.7: Experiences with Different SR-Proximity Functions: The Netlib-Standard Problems (I)

	q=1		**q=2**		**q=3**		**Dynamic**		
Problem	iter	digits	iter	digits	iter	digits	Flag-q	iter	digits
25fv47	29	8	32	8	37	7	0	29	8
80bau3b	42	9	37	6	53	9	0	42	9
adlittle	14	9	14	8	15	6	0	14	9
afiro	9	7	10	8	10	8	0	9	7
agg	25	7	27	7	28	7	0	25	7
agg2	21	8	22	8	23	8	0	21	8
agg3	22	8	24	8	24	8	0	22	8
bandm	17	6	17	6	17	6	0	17	6
beaconfd	12	7	13	8	14	8	0	12	7
blend	10	7	11	8	11	7	0	10	7
bnl1	35	7	35	7	45	7	1	34	7
bnl2	44	6	40	7	50	6	0	44	6
boeing1	28	6	24	6	30	6	0	28	6
boeing2	19	6	20	6	24	6	0	19	6
bore3d	20	8	20	7	23	8	1	20	8
brandy	17	6	17	6	21	7	0	17	6
capri	18	7	18	6	20	7	0	18	7
cycle	42	7	44	7	60	7	1	47	7
czprob	32	9	33	9	35	8	1	32	8
d2q06c	46	6	49	7	51	6	0	46	6
d6cube	19	9	21	0	22	8	0	19	9
degen2	13	0	13	0	15	10	0	13	10
degen3	14	9	14	8	15	9	0	14	9
dfl001	51	8	51	8	59	8	1	52	8
e226	20	7	21	8	25	7	0	20	7
etamacro	26	7	24	7	30	7	0	26	7
fffff800	29	6	28	6	29	6	0	29	6
finnis	27	9	31	9	27	9	1	27	8
fit1d	26	7	24	9	24	8	0	26	7
fit2d	20	7	22	8	21	8	0	20	7

Table 6.8: Experiences with Different SR-Proximate Functions: The Netlib-Standard Problems(II)

	q=1		**q=2**		**q=3**		**Dynamic**		
Problem	iter	digits	iter	digits	iter	digits	Flag-q	iter	digits
forplan	36	6	38	6	39	6	0	36	6
ganges	21	6	22	6	23	6	0	21	6
gfrd-pnc	18	8	18	8	19	8	0	18	8
greenbea	61	6	60	6	79	6	1	69	6
greenbeb	43	4	50	4	74	4	0	43	4
grow15	17	8	20	7	21	8	0	17	8
grow22	18	7	21	7	21	7	0	18	7
grow7	17	7	20	7	21	7	0	17	7
israel	23	6	25	6	31	6	0	23	6
kb2	17	9	18	9	20	8	0	17	9
lotfi	25	6	26	5	27	5	0	25	6
maros	29	4	32	4	38	4	0	29	4
maros-r7	16	8	16	9	16	8	0	16	8
modszk1	32	5	33	5	40	5	0	32	5
nesm	33	6	34	6	41	6	0	33	6
perold	47	6	46	6	52	6	1	48	6
pilot	52	4	50	4	56	4	1	52	4
pilotja	47	6	48	6	51	6	0	47	6
pilotwe	43	6	45	6	48	6	0	43	6
pilot4	39	8	39	7	43	7	0	39	8
pilot87	71	6	72	6	87	6	1	71	6
pilotnov	30	9	31	8	35	8	0	30	9
recipe	12	9	12	8	12	8	0	12	9
sc105	12	7	11	6	11	6	0	12	7
sc205	12	6	12	6	12	6	0	12	6
sc50a	10	7	10	7	10	7	0	10	7
sc50b	9	7	9	8	9	7	0	9	7
scagr25	17	9	17	8	17	8	0	17	9
scagr7	13	7	14	7	15	7	0	13	7
scfxm1	25	8	24	6	29	6	0	25	8
scfxm2	26	6	27	7	27	8	0	26	6
scfxm3	27	8	26	7	28	7	0	27	8

Table 6.9: Experiences with Different SR-Proximity Functions: The Netlib-Standard Problems (III)

	q=1		**q=2**		**q=3**		**Dynamic**		
Problem	iter	digits	iter	digits	iter	digits	Flag-q	iter	digits
scorpion	13	8	13	7	14	7	0	13	8
scrs8	25	5	25	5	27	5	0	25	5
scsd1	10	8	11	7	12	8	0	10	8
scsd6	12	8	17	8	23	9	0	12	8
scsd8	10	7	10	7	10	7	0	10	7
sctap1	18	9	19	9	21	8	0	18	9
sctap2	13	8	13	8	14	8	0	13	8
sctap3	13	7	14	9	14	7	0	13	7
seba	27	9	33	8	31	8	1	31	8
share1b	29	5	30	5	32	5	0	29	5
share2b	12	7	12	7	12	6	0	12	7
shell	26	8	25	8	26	8	1	23	8
ship04l	17	9	17	8	18	8	0	17	9
ship04s	17	9	17	8	27	8	0	17	9
ship08l	20	9	21	9	22	8	1	20	9
ship08s	18	8	19	9	20	9	1	19	9
ship12l	28	8	30	8	31	8	1	28	9
ship12s	22	7	23	9	25	9	1	23	7
sierra	18	8	20	8	22	8	0	18	8
stair	17	6	17	5	19	6	0	17	6
standata	16	10	17	9	17	8	0	16	10
standgub	16	8	17	9	17	8	0	16	8
standmps	19	8	19	9	23	10	0	19	8
stocfor1	14	8	14	8	15	9	0	14	8
stocfor2	30	6	31	7	35	6	0	30	6
stocfor3	48	6	49	6	56	6	0	48	6
truss	18	7	19	7	22	7	0	18	7
tuff	23	7	26	7	26	7	0	23	7
vtpbase	18	9	19	9	20	8	0	18	9
wood1p	17	9	17	8	18	8	0	17	9
woodw	23	7	24	7	24	8	0	23	7

Table 6.10: Experiences with Different SR-Proximity Functions: The Netlib-Infeasible Problems

	q=1	**q=2**	**q=3**	**Dynamic**	
Problem	iter	iter	iter	Flag-q	iter
bgdbg1	8	8	8	0	8
bgetam	7	7	7	0	7
bgprtr	10	11	11	0	10
box1	6	6	6	0	6
chemcom	6	6	6	0	6
cplex1	12	12	13	0	12
ex72a	6	6	6	0	6
ex73a	6	6	6	0	6
forest6	7	8	8	0	7
galenet	5	5	5	0	5
gosh	37	37	37	1	37
greenbeainf	24	27	28	0	24
itest2	5	5	5	0	5
itest6	5	5	5	0	5
klein1	17	19	22	0	17
klein2	16	16	18	0	16
klein3	18	18	20	0	18
mondou2	11	11	11	0	11
pang	37	34	39	0	37
pilot4i	16	17	19	0	16
qual	38	42	43	1	38
reactor	20	21	22	0	20
refinery	14	14	15	0	14
vol1	29	30	35	0	29

Table 6.11: Experiences with Different SR-Proximity Functions: Netlib-Kennington Problems

	q=1		**q=2**		**q=3**		**Dynamic**		
Problem	iter	digits	iter	digits	iter	digits	Flag-q	iter	digits
cre-a	28	8	28	8	38	8	0	28	8
cre-b	35	7	37	7	43	7	0	35	7
cre-c	31	8	35	8	40	8	0	31	8
cre-d	33	7	34	7	40	7	0	33	7
ken-07	17	8	18	8	19	7	0	17	8
ken-11	20	6	21	6	20	6	0	20	6
ken-13	31	10	34	9	36	8	1	32	10
osa-07	30	7	35	7	45	7	1	31	7
osa-14	43	8	44	8	49	8	1	43	8
osa-30	43	7	43	7	45	7	1	48	7
pds-02	36	7	35	7	35*	5	1	41	7
pds-06	99	6	55	7	56	7	1	60	7
pds-10	67	8	65	8	70	8	1	72	8

* get NAN from WSMP at iteration 36.

Table 6.12: All the Problems with $q > 1$ in Dynamic SR-IPM

	q=1		**q=2**		**q=3**		**Dynamic**	
Problem	iter	digits	iter	digits	iter	digits	iter	digits
bnl1	35	7	35	7	45	7	**34**	7
bore3d	**20**	8	20	7	23	8	**20**	8
cycle	**42**	7	44	7	60	7	47	7
czprob	**32**	9	33	9	35	8	32	8
dfl001	**51**	8	**51**	8	59	8	52	8
finnis	**27**	9	31	9	**27**	9	27	8
greenbea	61	6	**60**	6	79	6	69	6
perold	47	6	**46**	6	52	6	48	6
pilot	52	4	**50**	4	56	4	52	4
pilot87	**71**	6	72	6	87	6	**71**	6
seba	**27**	9	33	8	31	8	31	8
shell	26	8	25	8	26	8	**23**	8
ship08l	**20**	9	21	9	22	8	**20**	9
ship08s	**18**	8	19	9	20	9	19	9
ship12l	28	8	30	8	31	8	**28**	9
ship12s	**22**	7	23	9	25	9	23	7
gosh	37	infeasible	37	infeasible	37	infeasible	37	infeasible
ken-13	**31**	10	34	9	36	8	32	10
osa-07	**30**	7	35	7	45	7	31	7
osa-14	**43**	8	44	8	49	8	**43**	8
osa-30	**43**	7	**43**	7	45	7	48	7
pds-02	36	7	**35**	7	35*	5	41	7
pds-06	99	6	**55**	7	56	7	60	7
pds-10	67	8	**65**	8	70	8	72	8

Our results show that the dynamic version is comparable with the classical one ($q = 1$) and still have some room for improvement.

6.3 Comparing the Results with LIPSOL

In this section we will compare our algorithm with LIPSOL. The results from the current version of McIPM and LIPSOL are shown in Tables 6.13-6.17.

Based on Tables 6.13-6.15, one can see that for the Netlib-Standard problems, in twenty cases McIPM has a better performance than LIPSOL, and an thirty-six problems McIPM has comparable performance. We still need to improve on thirty-nine of the problems. For most of these, the precision of the solution is acceptable (7-9) with reasonably good iteration numbers.

Based on Table 6.16 we conclude that our embedding algorithm produce infeasible certificate for all 24 problems, where as LIPSOL gives infeasibility certificate for only 12 problems.

Table 6.17 shows for five of the Netlib-Kennington problems the performance of our McIPM is better, although with weaker precision, while for the remaining problems McIPM is weaker than LIPSOL. However, McIPM solved all these large problem and gave acceptable results.

Table 6.13: Comparison with LIPSOL: Netlib-Standard (I)

	Results from McIPM			**Results from LIPSOL**		
Problem	Iter	Residual	Digits	Iter	Residual	Digits
25fv47	29	1.10e-09	8	25	1.11e-14	11
80bau3b	42	3.97e-08	9	40	2.79e-09	9
adlittle	14	3.77e-09	9	13	1.70e-13	11
afiro	9	1.85e-08	10	8	1.96e-13	11
agg	25	4.29e-08	10	21	1.02e-11	11
agg2	21	9.67e-08	11	18	4.30e-11	11
agg3	22	9.94e-08	11	17	6.46e-11	11
bandm	17	7.52e-08	10	18	8.47e-11	10
beaconfd	12	1.92e-08	10	13	2.79e-11	11
blend	10	3.49e-08	10	12	9.68e-14	11
bnl1	34	8.13e-08	7	26	2.93e-11	7
bnl2	40	3.47e-08	9	31	4.78e-09	9
boeing1	24	6.02e-08	6	21	3.44e-10	9
boeing2	19	4.90e-08	6	19	5.63e-10	8
bore3d	20	6.80e-09	10	18	1.07e-11	11
brandy	17	3.42e-08	10	17	1.02e-13	11
capri	18	3.59e-09	6	19	2.00e-14	11
cycle	42	4.50e-08	8	25	2.78e-10	6
czprob	32	1.94e-08	9	36	5.88e-09	11
d2q06c	46	4.22e-08	7	32	6.50e-10	7
d6cube	19	6.12e-09	10	23	1.34e-10	8
degen2	13	8.01e-10	11	14	1.97e-10	10
degen3	15	8.12e-09	11	25	1.11e-10	6
dfl001	51	3.85e-08	8	79	1.23e-07	7
e226	20	3.62e-08	10	21	5.58e-14	11
etamacro	26	5.89e-08	7	25	9.18e-09	7
fffff800	27	2.85e-08	6	26	5.58e-09	7
finnis	27	6.56e-08	10	30	3.74e-12	11
fit1d	20	2.16e-08	10	19	2.80e-11	11
fit1p	15	2.59e-06	7	16	1.27e-09	10
fit2d	20	5.70e-08	9	7	9.19e-12	6
fit2p	19	3.65e-06	7	21	2.86e-11	9

Table 6.14: Comparison with LIPSOL: Netlib-Standard (II)

	Results from McIPM			**Results from LIPSOL**		
Problem	Iter	Residual	Digits	Iter	Residual	Digits
forplan	34	2.20e-10	6	22	2.14e-12	6
ganges	21	8.18e-08	6	18	1.87e-10	6
gfrd-pnc	18	3.44e-08	8	21	1.42e-13	11
greenbea	61	4.01e-11	6	43	1.63e-07	4
greenbeb	43	5.15e-08	4	38	3.21e-13	4
grow15	17	5.47e-08	10	17	6.97e-10	11
grow22	18	7.50e-08	7	19	1.01e-09	11
grow7	17	4.27e-08	10	16	4.47e-10	9
israel	23	8.04e-08	9	23	7.81e-09	11
kb2	17	1.13e-08	10	15	2.90e-12	11
lotfi	25	1.63e-08	7	18	2.64e-13	11
maros	29	6.92e-08	6	33	7.33e-13	11
maros-r7	16	2.59e-08	10	15	7.59e-09	11
modszk1	32	7.78e-08	7	24	5.18e-15	10
nesm	33	1.65e-08	6	33	5.38e-09	6
perold	46	1.63e-08	6	31	5.79e-11	6
pilot	50	7.59e-08	4	31	5.76e-09	4
pilotja	47	2.99e-08	6	31	6.84e-10	6
pilotwe	43	1.25e-08	7	37	1.12e-13	6
pilot4	39	2.42e-08	8	30	2.28e-09	8
pilot87	71	8.56e-08	6	38	8.56e-09	6
pilotnov	30	2.70e-08	9	20	2.39e-13	11
recipe	12	5.97e-09	7	9	9.74e-11	11
sc105	11	9.73e-10	8	10	2.34e-15	11
sc205	12	5.73e-09	8	10	9.66e-11	7
sc50a	10	4.28e-09	9	10	2.41e-15	11
sc50b	9	2.81e-09	7	7	5.80e-09	8
scagr25	17	7.84e-10	9	17	8.63e-11	11
scagr7	13	9.40e-08	7	14	2.08e-11	7
scfxm1	24	2.36e-09	9	19	1.06e-14	11
scfxm2	26	5.26e-08	11	21	1.84e-14	11
scfxm3	26	1.03e-09	10	21	1.33e-13	11

Table 6.15: Comparison with LIPSOL: Netlib-Standard (III)

	Results from McIPM			**Results from LIPSOL**		
Problem	Iter	Residual	Digits	Iter	Residual	Digits
scorpion	13	4.36e-08	10	15	2.48e-13	11
scrs8	25	4.68e-09	5	24	4.73e-12	5
scsd1	10	5.48e-09	8	9	8.07e-09	8
scsd6	12	2.44e-09	8	11	1.23e-09	8
scsd8	10	1.56e-08	7	11	1.43e-11	10
sctap1	18	1.37e-09	8	17	2.66e-13	11
sctap2	13	2.47e-08	8	19	5.03e-13	11
sctap3	13	8.04e-08	7	18	2.73e-14	11
seba	27	2.02e-08	9	22	4.31e-10	11
share1b	29	8.50e-08	6	22	5.28e-15	11
share2b	12	1.74e-09	7	13	7.83e-13	11
shell	23	5.49e-08	11	21	8.04e-15	11
ship04l	17	2.59e-09	11	14	2.21e-14	11
ship04s	17	2.71e-09	7	14	9.82e-11	10
ship08l	20	3.16e-08	9	16	1.12e-12	11
ship08s	18	8.49e-10	9	15	2.98e-13	11
ship12l	28	1.64e-09	9	18	6.36e-11	11
ship12s	22	7.61e-08	9	18	1.46e-11	11
sierra	18	7.69e-08	9	17	8.71e-11	11
stair	17	3.07e-08	7	14	1.95e-10	10
standata	16	1.08e-09	10	17	7.96e-15	11
standgub	16	1.76e-07	11	17	4.41e-13	11
standmps	19	8.84e-08	11	24	8.22e-14	11
stocfor1	14	6.56e-09	9	16	2.43e-11	11
stocfor2	30	4.45e-08	7	21	1.51e-10	11
stocfor3	48	9.67e-08	5	32	1.50e-10	5
truss	18	6.07e-08	8	19	3.69e-11	11
tuff	23	8.04e-08	7	20	5.99e-09	4
vtpbase	18	1.12e-09	10	23	2.70e-11	11
wood1p	17	9.04e-09	11	19	3.23e-09	6
woodw	23	7.72e-08	7	28	6.65e-10	7

Table 6.16: Comparison with LIPSOL: Netlib-Infeasible

	Results from McIPM		**Results from LIPSOL**	
Problem	Iter	Feasility	Iter	Feasility
bgdbg1	8	(LP) infeasible	6	(LP) infeasible??
bgetam	7	(LP) infeasible	6	(LP) infeasible??
bgprtr	10	(LP) infeasible	8	(LP) infeasible
box1	5	(LP) infeasible	4	(LP) infeasible
chemcom	6	(LP) infeasible	6	(LP) infeasible
cplex1	12	(LP) infeasible	6	(LP) infeasible
ex72a	6	(LP) infeasible	3	(LP) infeasible
ex73a	6	(LP) infeasible	3	(LP) infeasible
forest6	7	(LP) infeasible	10	(LP) infeasible
galenet	5	(LP) infeasible	5	(LP) infeasible
gosh	37	(LP) infeasible	16	(LP) infeasible??
greenbeainf	24	(LP) infeasible	13	Both infeasible???
itest2	5	(LP) infeasible	5	(LP) infeasible
itest6	5	(LP) infeasible	5	(LP) infeasible??
klein1	17	(LP) infeasible	18	Both infeasible???
klein2	16	(LP) infeasible	13	Both infeasible???
klein3	18	(LP) infeasible	16	Both infeasible???
mondou2	11	(LP) infeasible	5	(LP) infeasible
pang	34	(LP) infeasible	22	Both infeasible???
pilot4i	16	(LP) infeasible	16	Both infeasible???
qual	38	(LP) infeasible	40	(LP) infeasible
reactor	20	Both infeasible	10	(LP) infeasible??
refinery	14	(LP) infeasible	14	(LP) infeasible
vol1	29	(LP) infeasible	23	Both infeasible???

Note: ? means that LIPSOL got non-firm result.

Table 6.17: Comparison with LIPSOL: Netlib-Kennington

	Results from McIPM		**Results from LIPSOL**	
Problem	Iter	Digits	Iter	Digits
cre-a	28	8	30	11
cre-b	35	7	42	11
cre-c	31	8	30	11
cre-d	33	7	38	11
ken-07	17	8	16	11
ken-11	20	6	22	11
ken-13	31	10	27	11
osa-07	30	7	27	11
osa-14	43	8	37	11
osa-30	43	7	36	10
pds-02	35	7	29	11
pds-06	55	7	43	11
pds-10	65	8	53	11

Chapter 7

Conclusions and Future Work

In this thesis, we have summarized the fundamental facts about of LO, IPMs, and SR-IPMs. We also have discussed implementation details for a family of SR-IPMs that currently forms an LO software package: McIPM. Based on the test results, we are very confident that our algorithms are robust for all the test problems. These tests show encouraging computational results, and help us to identify items for future improvement. For some problems, the iteration numbers are less than those of LIPSOL, while the solutions have the same precision.

Due to time limitation we could not cover everything in the current version of McIPM. Thus there is still room for some improvements that should be investigated in the future, in order to make our algorithm and software more efficient and completely independent of third-party software. We list a few of these issues:

- numerical stability in calculating the search direction,
- numerical singularity of the matrix AD^2A^T,
- adaptive choice of the self-regular proximity,
- handling dense columns by Goldfarb-Scheinberg update,

- efficient line-search for SR search directions,
- preprocessing and postprocessing,
- embedding IPM based on OSL, ESSL and WSMP,
- sparse Cholesky and Bunch-Parlelt factorization with iterative refinement.

All these issues relate to the LO implementation. In the future we plan to extend our package to more general nonlinear problems, such as Quadratic Optimization (QO), Second-Order Conic Optimization (SOCO), Semi-Definite Optimization (SDO), Nonlinear Complementarity Problem (NLCP). Moreover, we plan to complete our software package by crossover to simplex and basis-identification procedures, sensitivity analysis, and so on.

Appendix A

Parameters in WSSMP

In what follows we give some explanation of the parameter of *wssmp* which is the primary routine in WSMP.

N is the dimension of the matrix A.

IA JA AVALS are arrays that are used to store matrix E in the lower triangle part in the compressed sparse column (CSC-LT) format.

DIAG is not used for (CSC-LT) format but modified compressed sparse row/column (MSR/MSC).

PERM is the permutation vector.

INVP is the inverse permutation vector.

B is a dense matrix that contains the multiple right hand sides of the system of equations $AX = B$ to be solved.

LDB is leading dimension of B.

NRHS is the number of right hand sides.

AUX is the auxiliary storage.

$NAUX$ is the size of AUX.

MRP is an integer array that is used to set bad pivot flags.

IPARM is an integer array of size 64 that is used to pass various parameters to *wssmp* and return some useful information about the execution of a call to *wssmp*. Now we only describe the relevant entries of *IPARM*.

IPARM(2) **and** *IPARM(3)* starting task and ending task, they control the subset of the task to be performed.

IPARM(5) defines the numbering style. We set *IPARM(5)=0*, then the C-style numbering (starting from 0) is used in our implementation.

IPARM(6) As an input to the iterative refinement step it is the maximum number of the iterative refinement to be performed. On the output, it contains the actual number of iterative refinement steps performed.

IPARM(7) controls how to measure the residual in iterative refinement.

IPARM(10) If *IPARM(10)=0*, then *wsmmp* will not perform any scaling of the input matrix and vectors. If *IPARM(10)=1*, then a scaling is performed. *IPARM(10)=2* instructs $WSMMP$ to perform the scaling only under certain conditions.

IPARM(11:12) Instruct *wssmp* how to handle very small and zero pivots.

IPARM(16:20) Controls the ordering and load-balancing permutation vectors. *IPARM(16:20)=(1,0,1,0,0)* is recommended by WSMP for *interior-point algorithms*.

IPARM(31) It will be read during the numerical factorization phase. You can set *IPARM(31)=0* for LL^T factorization, *IPARM(31)=1* for LDL^T factorization. Also you can set *IPARM(31)=2*, then each diagonal entry of the coefficient matrix A is compared against *DPARM(31)* and if any of the diagonal entries is less than or equal

to *DPARM(31)*, then LDL^T factorization is performed, else LL^T factorization is performed.

DPARM is a double precision parameter array. Unlike *IPARM*, only a few of the first 32 entries of $DPARM$ are used.

DPARM(6) is the relative error limits for iterative refinement. Iterative refinement is stopped if *IPARM(7)* > 0 and the relative error becomes less than *DPARM(6)*. It is not used if *IPARM(7)*$= 0$.

DPARM(11:12) is the bad (small) pivot threshold. It is used to provide user some control over pivoting.

DPARM(21:22) is the bad (small) pivot substitution. It must be non-negative.

DPARM(31) is the factorization option threshold.

Appendix B

Test Problems

This section contains NETLIB test problem summary. In the following tables, the column and nonzero counts exclude slack and surplus columns and the right-hand side vector, but include the cost rows. It omitted other free rows and all but the first right-hand side vector. The BR column indicates whether a problem has bounds or ranges: B stands for "has bounds", R for "has ranges". For more details see [6].

Table B.1: The Netlib-Standard Problems (I)

Name	Rows	Cols	Nonzeros	BR	Optimal Value
25FV47	822	1571	11127		5.5018458883E+03
80BAU3B	2263	9799	29063	B	* 9.8723216072E+05
ADLITTLE	57	97	465		2.2549496316E+05
AFIRO	28	32	88		-4.6475314286E+02
AGG	489	163	2541		-3.5991767287E+07
AGG2	517	302	4515		-2.0239252356E+07
AGG3	517	302	4531		1.0312115935E+07
BANDM	306	472	2659		-1.5862801845E+02
BEACONFD	174	262	3476		3.3592485807E+04
BLEND	75	83	521		-3.0812149846E+01
BNL1	644	1175	6129		1.9776292856E+03
BNL2	2325	3489	16124		1.8112365404E+03
BOEING1	351	384	3865	BR	-3.3521356751E+02
BOEING2	167	143	1339	BR	-3.1501872802E+02
BORE3D	234	315	1525	B	1.3730803942E+03
BRANDY	221	249	2150		1.5185098965E+03
CAPRI	272	353	1786	B	2.6900129138E+03
CYCLE	1904	2857	21322	B	-5.2263930249E+00
CZPROB	930	3523	14173	B	2.1851966989E+06
D2Q06C	2172	5167	35674		1.2278423615E+05
D6CUBE	416	6184	43888	B	3.1549166667E+02
DEGEN2	445	534	4449		-1.4351780000E+03
DEGEN3	1504	1818	26230		-9.8729400000E+02
DFL001	6072	12230	41873	B	* 1.12664E+07
E226	224	282	2767		-1.8751929066E+01
ETAMACRO	401	688	2489	B	-7.5571521774E+02
FFFFF800	525	854	6235		5.5567961165E+05
FINNIS	498	614	2714	B	* 1.7279096547E+05
FIT1D	25	1026	14430	B	-9.1463780924E+03
FIT1P	628	1677	10894	B	9.1463780924E+03
FIT2D	26	10500	138018	B	-6.8464293294E+04
FIT2P	3001	13525	60784	B	6.8464293232E+04

Table B.2: The Netlib-Standard Problems (II)

Name	Rows	Cols	Nonzeros	BR	Optimal Value
FORPLAN	162	421	4916	BR	-6.6421873953E+02
GANGES	1310	1681	7021	B	-1.0958636356E+05
GFRD-PNC	617	1092	3467	B	6.9022359995E+06
GREENBEA	2393	5405	31499	B	* -7.2462405908E+07
GREENBEB	2393	5405	31499	B	-4.3021476065E+06
GROW15	301	645	5665	B	-1.0687094129E+08
GROW22	441	946	8318	B	-1.6083433648E+08
GROW7	141	301	2633	B	-4.7787811815E+07
ISRAEL	175	142	2358		-8.9664482186E+05
KB2	44	41	291	B	-1.7499001299E+03
LOTFI	154	308	1086		-2.5264706062E+01
MAROS	847	1443	10006	B	-5.8063743701E+04
MAROS-R7	3137	9408	151120		1.4971851665E+06
MODSZK1	688	1620	4158	B	3.2061972906E+02
NESM	663	2923	13988	BR	1.4076073035E+07
PEROLD	626	1376	6026	B	-9.3807580773E+03
PILOT	1442	3652	43220	B	-5.5740430007E+02
PILOT.JA	941	1988	14706	B	-6.1131344111E+03
PILOT.WE	723	2789	9218	B	-2.7201027439E+06
PILOT4	411	1000	5145	B	-2.5811392641E+03
PILOT87	2031	4883	73804	B	3.0171072827E+02
PILOTNOV	976	2172	13129	B	-4.4972761882E+03
RECIPE	92	180	752	B	-2.6661600000E+02
SC105	106	103	281		-5.2202061212E+01
SC205	206	203	552		-5.2202061212E+01
SC50A	51	48	131		-6.4575077059E+01
SC50B	51	48	119		-7.0000000000E+01
SCAGR25	472	500	2029		-1.4753433061E+07
SCAGR7	130	140	553		-2.3313892548E+06
SCFXM1	331	457	2612		1.8416759028E+04
SCFXM2	661	914	5229		3.6660261565E+04
SCFXM3	991	1371	7846		5.4901254550E+04

Table B.3: The Netlib-Standard Problems (III)

Name	Rows	Cols	Nonzeros	BR	Optimal Value
SCORPION	389	358	1708		1.8781248227E+03
SCRS8	491	1169	4029		9.0429998619E+02
SCSD1	78	760	3148		8.6666666743E+00
SCSD6	148	1350	5666		5.0500000078E+01
SCSD8	398	2750	11334		9.0499999993E+02
SCTAP1	301	480	2052		1.4122500000E+03
SCTAP2	1091	1880	8124		1.7248071429E+03
SCTAP3	1481	2480	10734		1.4240000000E+03
SEBA	516	1028	4874	BR	1.5711600000E+04
SHARE1B	118	225	1182		-7.6589318579E+04
SHARE2B	97	79	730		-4.1573224074E+02
SHELL	537	1775	4900	B	1.2088253460E+09
SHIP04L	403	2118	8450		1.7933245380E+06
SHIP04S	403	1458	5810		1.7987147004E+06
SHIP08L	779	4283	17085		1.9090552114E+06
SHIP08S	779	2387	9501		1.9200982105E+06
SHIP12L	1152	5427	21597		1.4701879193E+06
SHIP12S	1152	2763	10941		1.4892361344E+06
SIERRA	1228	2036	9252	B	1.5394362184E+07
STAIR	357	467	3857	B	-2.5126695119E+02
STANDATA	360	1075	3038	B	1.2576995000E+03
STANDGUB	362	1184	3147	B	1.2576995000E+03
STANDMPS	468	1075	3686	B	1.4060175000E+03
STOCFOR1	118	111	474		-4.1131976219E+04
STOCFOR2	2158	2031	9492		-3.9024408538E+04
STOCFOR3	16676	15695	74004		-3.9976661576E+04
TRUSS	1001	8806	36642		4.5881584719E+05
TUFF	334	587	4523	B	2.9214776509E-01
VTP.BASE	199	203	914	B	1.2983146246E+05
WOOD1P	245	2594	70216		1.4429024116E+00
WOODW	1099	8405	37478		1.3044763331E+00

* Those solutions in Netlib are different from the solutions we get from various IPM solvers.

The solution in this table come from CPLEX as standard solution for our digits comparison.

Problem	80bau3b	dfl001	finnis	greenbea
Netlib	9.8723216072e+05	1.12664E+07	1.7279096547e+05	-7.2462405908e+07
CPLEX	9.8722419241e+05	1.1266396047e+07	1.7279106560e+05	-7.2555248130e+07
LIPSOL	9.8722419241e+05	1.1266395657e+07	1.7279106560e+05	-7.2552052069e+07
OSL	9.8722419250e+05	1.1266397783e+07	1.7279106791e+05	-7.2411834102e+07
McIPM	9.8722419247e+05	1.1266396083e+07	1.7279106561e+05	-7.2555229157e+07

Note: We list the solutions obtained by using CPLEX, LIPSOL, OSL, and McIPM.

Table B.4: The Netlib-Infeasible Problems

Name	Rows	Cols	Nonzeros	BR
bgdbg1	349	407	1485	B
bgetam	401	688	2489	B
bgprtr	3577	824	17604	B
box1	232	261	912	B
ceria3d	3577	824	17604	B
chemcom	289	720	2190	B
cplex1	3006	3221	10664	B
cplex2	225	221	1059	B
ex72a	198	215	682	B
ex73a	194	211	668	B
forest6	67	95	270	B
galenet	9	8	16	B
gosh	3793	10733	97257	B
gran	2569	2520	20151	B
greenbea	2505	5405	35159	B
itest2	10	4	17	
itest6	12	8	23	
klein1	55	54	696	
klein2	478	54	4585	
klein3	995	88	12107	
mondou2	313	604	1623	B
pang	362	460	2666	B
pilot4i	411	1000	5145	B
qual	324	464	1714	B
reactor	319	637	2995	B
refinery	324	464	1714	B
vol1	324	464	1714	B
woodinfe	36	89	209	

Table B.5: The Netlib-Kennington Problems

Name	Rows	Cols	Nonzeros	BR	Solutions *
cre-a	3517	4067	19054		2.3595407061e+07
cre-b	9649	72447	328542		2.3129639887e+07
cre-c	3069	3678	16922		2.5275116141e+07
cre-d	8927	69980	312626		2.4454969765e+07
ken-07	2427	3602	19054	B	-6.7952044338e+08
ken-11	14695	21349	70354	B	-6.9723822625e+09
ken-13	28633	42659	139834	B	-1.0257394789e+10
ken-18	105128	154699	512719	B	-5.2217025287e+10
osa-07	1119	23949	167643		5.3572251730e+05
osa-14	2338	52460	367220		1.1064628447e+06
osa-30	4351	100024	700160		2.1421398732e+06
osa-60‡	10281	232966	1630758		4.0440725032e+06
pds-02	2954	7535	21252	B	2.8857862010e+10
pds-06	9882	28655	82269	B	2.7761037600e+10
pds-10	16559	48763	140063	B	2.6727094976e+10
pds-20	33875	105728	304153	B	2.3821658640e+10

* The solution is obtained by using CPLEX.

‡This problem needs more memory than 1 GB.

Bibliography

[1] E.D. Andersen, C. Roos, T. Terlaky, T. Trafalis, and J.P. Warners. The use of low-rank updates in interior-point methods. Technical Report, Delft University of Technology, The Netherlands, 1996.

[2] E.D. Andersen and K.D. Anderson. The Mosek Interior point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm. In H. Frenk , C. Roos, T. Terlaky and S. Zhang (Editors), *High Performance Optimization*, Kluwer Academic Publishers, Boston, 197–232, 1999.

[3] K.D. Andersen. A Modified Schur Complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Trans. Math. Software*, 22(3), 348–356, 1996.

[4] G.B. Dantzig. *Linear Programming and Extensions.* Princeton Univ. Press, Princton, New Jersey, 1963.

[5] Gy. Farkas. *A Fourier-féle mechanikai elv alkalmazásai.* Mathemtikai és Természettudományi Értesítő 12, 457-472, 1894.

[6] D.M. Gay. Electronic mail distrubuton of linear programming testing problems. *Mathematical Programming Society COAL Newsletter*, No. 13, 10-12, 1985.

[7] A.J. Goldman and A.W. Tucker. Theory of linear programming. In H.W. Kuhn and A.W. Tucker. Editors, *Linear Inequalities and Related Sys-*

tems, Annals of Mathematical Studies, Princeton, New Jersey, No. 38, 53–97, 1956.

[8] G.H. Golub and C.F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore and London, 2nd Edition, 1989.

[9] O. Güler, D.den Hertog, C. Roos, T. Terlaky, and T. Tsuchiya. Degeneracy in iterior point methods for linear programming: A survey. *Annals of Operations Research*, 46, 107–138, 1993.

[10] A. Gupta. WSMP: Watson Sparse Matrix Package (Part I: direct solution of symmetric sparse systems). Technical Report RC 21886 (98462), IBM T.J. Watson Research Center, Yorktown Heights, NY, 2000. *http://www.cs.umn.edu/~agupta/wsmp.html.*

[11] W.W. Hager. Updating the inverse of a matrix. *SIAM Rev.,* 31(2), 221–239, 1989.

[12] F.S. Hillier and G.J. Lieberman. *Introduction to Operation Research*. McGraw Hill, Boston, 7th Edition, 2001.

[13] N. Karmarkar. A ploynomial-time algorithm for linear programming. *Combinatorica,* 4, 373–395, 1984.

[14] L.G. Khachian. Polynomial algorithmn in linear programming (in Russian). *Doklady Akademiia Nauk SSR*, 224, 1039–1096, 1979. English Translation: *Soviet Mathematics Doklady*, Volumne 20, 191–194, 1979.

[15] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior-point algorithm fo linear programming. In N. Megiddo (Editor), *Progress in Mathematical Programming: Interior-Point Algorithms and Related Methods*, Springer Verlag, Berlin, 29–47, 1989.

[16] J.W.-H. Liu. The multifronttal method for sparse matrix solution: Theory and practice. *SIAM Review*, 34, 82–109, 1992.

[17] V. Klee and G. Minty. How good is the simplex algorithm? In O.Shisha (Editor), *Inequalities-III,* Academic Press, New York, 53, 159–175, 1972.

[18] I.J. Lustig, R.E, Marsten, and D.F. Shanno. Interior Point Methods for linear programming problems: Computational State of the Art, *ORSA Journal on Computing,* 6, 1–14, 1994.

[19] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optimization*, 2(4), 575–601, 1992.

[20] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming.* The McGraw-Hill Companies, Inc. New York. Internation Edition, 1996.

[21] L.L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, Oxford, 1987.

[22] J. Peng, C. Roos, and T. Terlaky. Self-Regular proximities and new search directions for linear and semidefinite optimization. To appear in *Mathematical Programming,* 2002.

[23] C. Roos, T. Terlaky, and J.-Ph. Vial. *Theory and Algorithms for Linear Optimization. An Interior Approach.* John Wiley and Sons, Chichester, UK, 1997.

[24] T. Terlaky (Editors). *Interior Point Methods of Mathematical Programming,* volume 5 of *Applied Optimization.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[25] T. Terlaky. An Easy Way to Teach Interior Point Methods, *European Journal of Operation Reasearch,* vol 130, 1, 1–19, 2001.

[26] R. J. Vanderbei. *Linear Programming Foundations and Extensions.* Kluwer Academic Publishers, 1996.

[27] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.* 2, 77-79, 1981.

[28] Y. Ye. *Interior-Point Algorithms, Theory and Analysis.* John Wiley & Sons, Chichester, UK, 1997.

[29] Y. Zhang. Solving large-scale linear programs by interior-point methods under the MATLAB enviroment. *Optimization Methods and Software*, 10, 1-31, 1999.

[30] Y. Zhang. User's Guide to LIPSOL Linear-programming Interior Point Solvers V0.4. *Optimization Methods and Software*, 11 & 12, 385-396, 1999.

Index