IMPLEMENTATION OF THE NEW INTERIOR-POINT METHODS

IMPLEMENTATION OF THE NEW INTERIOR-POINT METHODS

By

Qirong Miao, M.Sc.

A Thesis

Submitted to the School of Graduate Studies

in Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2002) McMaster University

COMPUTING & SOFTWARE Hamilton, Ontario

TITLE: Implementation of The New Interior-Point Methods

AUTHOR: Qirong Miao, M.Sc.

SUPERVISOR: Dr. Tamás Terlaky

NUMBER OF PAGES: XV, 86

To my wife, Liping Wei,

and son, Wei Miao.

Abstract

Interior Point Methods have been the largest and most active area of research in optimization for the past 18 years. We discuss a new Interior point method: a Self-Regular proximity function based Predictor-Corrector Interior Point method that is combined with the Homogeneous self-dual embedding strategy. This family of Interior Point Methods was first introduced by Peng, Roos and Terlaky. The goal of this research was to develop a high quality implementation of this method and test the power of the new methods in practice.

Our implementation is based on IBM's OSL, ESSL and WSMP packages. The computations are performed on an IBM RS-6000 workstation. Various implementation issues are discussed and their effects are evaluated. We perform two kinds of comparison:

(1) we compare our testing results with the results obtained from OSL;

(2) we compare the computational performance by evaluating the effect of the barrier parameter q in the Self-Regular proximity function.

Computational results confirm that our implementation is reliable, robust and the computational results are encouraging. In particular, our implementation compares favorably with the commercial solver package OSL when detecting the infeasibility of the testing problem.

Acknowledgements

I gratefully acknowledge my supervisor Dr. T. Terlaky, the leader of the Advanced Optimization Laboratory in the Department of Computing and Software at McMaster University. He spent much time on advising and teaching me. He is always ready to lend a hand whenever I meet some trouble in my study. In particular, he has a thorough understanding of the most recent achievements in the field of Interior-Point Methods and a smart insight for the software developing and debugging, these are really the endless help for my research.

I would like to thank all the members of my defence committee, Dr. T. Terlaky, Dr. M. von Morenschildt and Dr. J. Peng for their careful examination of this thesis and for their helpful advice and suggestions.

I thank all the members of the optimization group for their kind help and for the enjoyable time they gave to me. I also appreciate the generous help from my classmates and friends in Hamilton.

Last but not least, I want to express my warmest thanks to my wife and son. They also contributed substantially to this thesis by their love, support, and by forgiving my absence as husband and father for two years. Without their continuous support, this thesis would not have been possible.

Contents

List of Notation

A_d	:	the dense part of the matrix A
A_s	:	the sparse part of the matrix A
A^T	:	transpose of the matrix A
C^2	:	the set of twice continuously differentiable functions
D^2	:	the diagonal matrix associated with the simplified (HLP) model
e	:	the vector with all components equal to one
F	:	the matrix associated with the variable upper bounds in the LO problem
$\mathcal{F}$	:	the feasible set of the LO problem
$\widetilde{\mathcal{F}}$	:	the optimal solution set of primal-dual pair (LP) and (LD)
L	:	the input length of the LO problem
μ	:	the centering parameter
μ^0	:	initial value of the centering parameter
R^n	:	n-dimension Euclidean vector space
$R^n_+(R^n_{++})$	:	nonnegative (positive) orthant in R^n
x_i	:	i-th coordinate of the vector x
x^T	:	transpose of the vector x
Δx	:	the primal part of search direction
Δs	:	the dual part of search direction
X	:	diagonal matrix diag(x) generated by vector x
xs	:	the coordinatewise product of the vectors x and s
v	:	the scaled vector defined by $v = \sqrt{\frac{xs}{\mu}}$

d_x	:	the primal part of the search direction in the scaled v-space: $d_x = \frac{v\Delta x}{x}$
d_s	:	the dual part of the search direction in the scaled v-space: $d_x = \frac{v\Delta s}{s}$
α	:	step size
ϵ	:	accuracy parameter
τ	:	proximity parameter
θ	:	updating factor for μ
$\psi(t)$	:	proximity function in R
$\psi(x)$	:	proximity function in R^n
$\Psi(x)$	:	$\Psi(x) = \sum_{i=1}^{n} \psi(x_i)$
$\mathcal{N}(\mu, \tau)$	:	a neighborhood around the central path
$\Upsilon_{p,q}(t)$	:	a class of self-regular functions with $p, q \geq 1$
Ω_1	:	the set of functions that satisfy condition SR1
Ω_2	:	the set of functions that satisfy condition SR2

List of Figures

List of Tables

List of Abbreviation

LO	:	Linear Optimization
IPM(s)	:	Interior-Point Method(s)
IPC	:	Interior-Point Condition
SR	:	Self-Regular
OSL	:	Optimization Subroutine Library
ESSL	:	Engineering and Scientific Subroutine Library
WSMP	:	Watson Sparse Matrix Package

Preface

Interior-Point Methods have been the largest and most dramatic area of research in optimization for the past 18 years. These methods were theoretically proven to converge in polynomial time. Moreover, Interior-Point Methods for linear optimization have become quite mature in practice as well. Today, Interior-Point Methods have permanently changed the landscape of linear programming theory, practice and computation.

In this thesis, we discuss a new Interior-Point method, which is the Self-Regular proximity function based Predictor-Corrector Interior-Point method. Then we implement this new method and analyze the computational results.

The thesis consists of six chapters. In Chapter 1, we briefly introduce some basic concepts and important results of linear optimization, such as the standard and canonical forms of linear optimization problems, duality and complementarity, and so on.

In Chapter 2, a brief introduction to Interior-Point Methods is presented. First, the basic features of Interior-Point Methods are described, then some fundamental concepts (interior-point condition, central path, analytical center, etc.) and their properties are summarized. Then, the Primal-Dual Path-Following algorithm, one of the most important Interior-Point Methods, is discussed. Further, we consider how to choose step length and stopping criteria. Finally, we turn our attention to the homogeneous embedding model that acts as a pivotal role in our implementation.

In Chapter 3, We discuss a new class of proximity measures called *Self-Regular* Proximities, which was first introduced by J. Peng, C. Roos, and T. Terlaky.

In Chapter 4, we first consider the Newton system of the homogeneous self-dual embedding model for the standard linear optimization problem, and compute the Newton direction by using a simplified method. Then we extend these to the Newton system of the homogeneous self-dual embedding model with variable upper bounds. Preprocessing, efficient handling of dense columns and predictor-corrector methods are discussed as well. A concise description of our implementation illustrated by some flowcharts of the program is presented in the last part of this chapter.

In Chapter 5, we present our computational results using the Self-Regular based predictor-corrector IPM combining with the homogeneous self-dual embedding strategy. We perform two kinds of comparisons: 1) we compare our testing results with the results obtained from OSL package; 2) we compare the computational performance by evaluating the effect of the barrier parameter q in the Self-Regular proximity function. Finally, we get our conclusions based on these testing results and comparisons.

In the last chapter, we give some suggestions for further work.

Chapter 1

Introduction to Linear Optimization

Linear Optimization (LO) is one of the most widely taught and fast developing techniques in applied mathematics, it has been used successfully to develop a large number of applications in many areas of science, engineering, commerce and industry. It might mean running a business to maximize profit, minimize loss, maximize efficiency, or minimize risk. It might mean selecting a flight plan for an airline to minimize time or fuel use. It might be the practical problems such as investing strategy, bridge design, circuit design, and so on [28].

Linear Optimization deals with a simple mathematical model: find the optimal (minimal or maximal) value of a *linear function* subject to *linear constraints* imposed on the variables. The constraints may be either equality or inequality constraints. LO exhibits a wonderful combination of both continuous algebraic and combinatorial properties. The wide applicability of LO models, the rich mathematical theory underlying these models and the methods developed to solve them have been the driving forces behind the rapid and continuing evolution of the subject.

There are many different ways to represent an LO problem. The two most popular and widely used representations are the standard and the canonical forms[1]. The *standard form* LO Problem can be described as follows:

$$\begin{array}{lll} & \min & c^T x \\ (\text{LP}) & \text{s.t.} & Ax = b, \\ & & x \geq 0, \end{array} \tag{1.1}$$

where $A \in R^{m \times n}$, $b \in R^m$ and $c \in R^n$. A vector $x \in R^n$ is called feasible if it satisfies $Ax = b$ and $x \geq 0$.

The *canonical form* LO problem is given as

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & Ax \leq b, \\ & x \geq 0, \end{array} \tag{1.2}$$

[1]It is well known [29] that any LO problem can be converted to either standard or canonical form.

where $A \in R^{m\times n}$, $b \in R^m$ and $c \in R^n$.

1.1 History and Developments of LO

Linear Optimization models have been used for centuries, the field of Linear Optimization has been given the name *Linear Programming* originally, but nowadays the word 'programming' usually refers to the activity of writing computer programs, and as a consequence its use instead of the more natural word 'optimization' gives rise to confusion. Therefore, we prefer to use the name *Linear Optimization*, and this terminology has already become generally accepted.

LO models originated from studying systems of linear inequalities. During 1940's, many LO problems related to World War II were presented, it became clear that a computable method for solving linear optimization problems was needed.

The Simplex Method was invented by George B. Dantzig [8] in 1947. It explicitly explores its combinatorial structure to identify the solution by moving from a vertex to an adjacent one of the feasible set with increasing values of the objective function for maximization problems or decreasing values of the objective function for minimization problems. Since the initial formulation in 1947, Dantzig's Simplex Method has been successfully improved both from the theoretical and practical side by many scholars and researchers. It is generally recognized to be very robust, efficient and it has achieved tremendous success in both theory and application, and still remains to be one of the efficient methods for solving LO problems. In an effort to explain the remarkable efficiency of the Simplex Method, optimization people strived to prove that an LO problem is solvable in polynomial time[2] by the Simplex Method. This topic belongs to the area of *complexity theory*. So far this question has not been solved yet, but it stimulated the development of two important new algorithm classes for LO: Ellipsoid algorithms and Interior-Point algorithms.

In 1979, L.G. Khachiyan [18] invented the ellipsoid algorithm for LO . It is based on the ellipsoid method that was originally developed for nonlinear optimization problems [44]. With this technique, Khachiyan proved that LO belongs to the class of polynomially solvable problems and the ellipsoid algorithm has a polynomial $O(n^2L)$ iteration complexity with a total of $O(n^4L)$ bit operations, where L is the input size of a problem instance[3] and n is the number of variables. Khachiyan's results were immediately cheered in the international press, and the ellipsoid algorithm was subsequently the subject of wide and

[2]A problem is called *solvable in polynomial time* (or simply *polynomial* or *easy*), if there exists an algorithm that solves each instance of the problem in a time that is bounded above by a polynomial of the input size of the instance of the problem; otherwise the problem is considered to be *non-polynomial* or *hard*.

[3]In general, the input size L of an instance is defined as the length of a binary string that is needed to encode the data of the instance. For example, the binary encoding of an integer α needs a string of length $L = 1 + \lceil \log_2(1 + |\alpha|)\rceil$, where the first 1 serves to encode the sign of the number. Similarly, the binary encoding of an integer vector a with dimension n needs a string of length $L = \sum_{j=1}^{n}(1 + \lceil \log_2(1 + |a_j|)\rceil)$;

intensive investigation by various scholars both in the theoretical and computational side. Although Khachiyan's results have had a great theoretical impact, unfortunately, in spite of high expectations, the new algorithm failed to deliver its promises in practical computational efficiency. Even the best implementations of the ellipsoid method are far from being competitive with existing Simplex solvers.

In 1984, N. Karmarkar proposed his new algorithm for LO [17], which started the new era of Interior-Point Methods. It is also a polynomial time algorithm and has a polynomial $O(nL)$ iteration complexity with a total of $O(n^{3.5}L)$ bit operations. Karmarkar also announced that his algorithm could solve large-scale linear programming problems much faster than the existing implementations of the Simplex Method. At that time the name of Karmarkar and his algorithm reached the front-page of New York Times, it attracted many optimization scholars and researchers into this new flourishing field. Hundreds of papers related to this new algorithm were published subsequently. On the other hand, at that time Karmarkar just show the efficiency of his algorithm theoretically, there were no sufficient verifiable computational results, thus his claims about computational efficiency were received with much doubt and scepticism by some experts in the field of mathematical programming. Some years later, a number of efficient solvers were implemented[4], and Interior-Point Methods were proved to be efficient both in theory and practice. In 1987, N. Megiddo [23] presented a framework for primal-dual Interior-Point algorithms. The primal-dual viewpoint proved to be extremly productive. Another notable contribution was Renegar's algorithm [33]. In 1989, Mehrotra described an efficient algorithm for LO [25]. Nowadays, it is clear that Interior-Point Methods provide competitive solvers for large classes of optimization problems.

The rapid development of LO is due not only to the efficient algorithms, but also due to developments in computer technology. In the days before digital computers, only very simple optimization models involving few variables and few constraints were solvable by some special tools and methods. Therefore, in those days, a great deal of emphasis was placed on keeping the model very small and compact, otherwise it could not have been solved. The invention of the digital computer has changed incredibly the whole world, and thus the area of optimization as well. In the last 50 years, by combining computers with efficient iterative methods, a large number of algorithms to solve a great variety of optimization problems have become widely available. We can now solve many large and complex optimization problems that were out of reach before the advent of computers. For example, linear models of airline crew scheduling problems with as many as 13 million variables have been solved within three minutes on a four-processor Silicon Graphics Power Challenge workstation [34]. The achieved acceleration in the last 50 years

---

the binary encoding of a integer matrix A with dimension $m \times n$ needs a string of length $L = \sum_{i=1}^{m}\sum_{j=1}^{n}(1 + \lceil \log_2(1 + |A_{ij}|) \rceil)$.

[4]These solvers include PCx (by Czyzyk, Mehrotra, and Wright [7]), HOPDM (by Gondzio et al. [14]), BPMPD (by Mészáros [36]), OSL (IBM), CPLEX (CPLEX Optimization Inc. [6]), XPRESS (Dash Associates [10]), LOQO (by R. Vanderbei [38]), and LIPSOL (by Y. Zhang [45]). They can be found on the World Wide Web page

http://www.mcs.anl.gov/home/wright/IPPD/.

is due partly to advances in computer technology and for a significant part also to the developments in the field of IPMs for LO.

The following table describes some features of the three famous methods [34]:

Method	Inventor	Year Invented	Iteration Complexity	Bit Operations	Efficient In Practice
Simplex	Dantzig	1947	not−polynomial	not−polynomial	yes
Ellipsoid	Khachiyan	1979	$O(n^2L)$	$O(n^4L)$	not
IPM	Karmarkar	1984	$O(nL)$	$O(n^{3.5}L)$	yes
IPM today	Renegar/Roos, Vial	1988	$O(n^{0.5}L)$	$O(n^3L)$	yes

Table 1.1 Performance Comparison of Three Methods.

The following table describes the speed improvement of solving LO problems in the last half century [5, 9, 12].

Year	Methods	Platform	Size of solvable problem	Time
1950's	Simplex	Mainframe	100	
1960's	Simplex	Mainframe	1000	
1970's	Simplex	Mainframe	10000	
1980's	IPMs, Simplex	Mainframe	100000	
		PC	5000	
2000	IPMs, Simplex	Mainframe	12000000	
		PC	100000	

Table 1.2 The Speed Improvement of Solving LO Problems.

Remark: Bixby claimed that the last 10 years have seen an estimated 10^6 speed improvement [34], 10^3 is due to new algorithms and 10^3 is due to computer technology.

1.2 Duality and Complementarity

For every LO problem there is a companion problem, called the *dual* LO problem. For every variable in the original or *primal* LO problem there is a constraint in the dual LO problem, and for every constraint in the primal there is a variable in the dual. Duality is not only very important but also very useful. It provides the basis to develop efficient algorithms to solve LO problems.

We consider the standard LO problem called the primal problem (LP) as given by (1.1), and its dual problem (LD)

$$\begin{array}{lll} & \max & b^T y \\ (\text{LD}) & \text{s.t.} & A^T y \leq c, \\ & & y \in R^m \text{ is free.} \end{array}$$

By introducing a slack vector $s \in R^n$, the dual problem can be stated as follow:

$$\begin{array}{lll} & \max & b^T y \\ (\text{LD}) & \text{s.t.} & A^T y + s = c, \\ & & s \geq 0, \ y \text{ is free.} \end{array}$$

Vectors y and s are called dual feasible if they satisfy $A^T y + s = c$ and $s \geq 0$.

It is easy to get the following result.

Lemma 1.1 The dual of the dual problem (LD) is equivalent to the primal problem (LP).

There are some very important results about the primal and dual problems.

Theorem 1.2 (Weak Duality) Let x be a feasible solution for the primal problem (LP), and let y, s be a feasible solution for the dual problem (LD). Then

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

Proof: Since $Ax = b$ then $y^T Ax = y^T b$ for any $y \in R^n$. Furthermore, if $A^T y \leq c$ and $x \geq 0$, then $y^T Ax \leq c^T x$. Combining these two results concludes the proof. □

Remark: The value $c^T x - b^T y$ is called the *duality gap* between the primal and dual problems. It is always nonnegative for primal and dual feasible solutions.

This theorem states that the objective value corresponding to a primal (dual) feasible solution provides an upper (lower) bound for the objective value for any feasible solution, including optimal solutions for the other problem.

Definition 1.1 An LO problem is unbounded if there is a sequence of points $\{x^k\}_{k=1}^{\infty}$ in the feasible region such that the objective values converge to minus infinity, i.e., $c^T x^k \to -\infty$.

Definition 1.2 A feasible point x^* (or y^*) is called primal (or dual) optimal if for any feasible point x (or y), we have $c^T x^* \leq c^T x$ (or $b^T y^* \geq b^T y$).

Two immediate consequences are:

Corollary 1.3 If the primal problem is unbounded then the dual problem is infeasible. If the dual problem is unbounded then the primal problem is infeasible.

Corollary 1.4 If x is a feasible solution to the primal problem and y is a feasible solution to the dual problem and $c^T x = b^T y$, then both x and y are optimal for their respective problems.

We note that the converse of Corollary 1.3 is not necessarily true. If either the primal or the dual problem is infeasible then it does *not* follow that the other problem is unbounded; both can be infeasible.

The most important result is the so-called *Strong* Duality Theorem. It states that the optimal values of the primal and dual problems are equal, if they exist [30].

Theorem 1.5 (Strong Duality) Consider a pair of primal and dual LO problems. If one of the problems has a finite optimal solution, then so does the other and the optimal objective values are equal, i.e., $c^T x = b^T y$, where x is primal optimal and y is dual optimal.

The following theorem of *alternatives* for systems of linear equations and inequalities is well-known [11, 29] and plays a crucial role in the theory of LO, it is equivalent to the Strong Duality Theorem 1.5.

Theorem 1.6 (Farkas Lemma) The system

$$\text{(I)} \qquad Ax = b, \qquad x \geq 0$$

is *unsolvable* if and only if the system

$$\text{(II)} \qquad A^T y \leq 0, \quad b^T y > 0$$

is *solvable*.

Farkas' Lemma [11, 29] predates the development of LO and is often used to prove the Strong Duality Theorem, rather than the other way round. Geometrically it states that exactly one of the following two statements is true: (I) b is in the convex cone $\mathcal{C}$ generated by the columns of A; or (II) there is a vector y that makes an acute angle with b but not with any vector in $\mathcal{C}$.

So far we have discussed some properties of duality and presented a pair of primal and dual problems (LP) and (LD). Now we discuss a further relationship between them. At optimality there is an interdependence between the nonnegativity constraints $x \geq 0$ in

the primal and the inequality constraints $A^T y \leq c$ (or nonnegativity of the slack variables $s = c - A^T y \geq 0$) in the dual. For optimal solutions it is not possible to have both $x_j > 0$ and $s_j = c_j - (A^T y)_j > 0$ for any index j. At least one of these constraints must be binding, i.e., either x_j is zero or the j^{th} dual slack variables s_j is zero. This property is the so-called *complementarity* condition and we summarize it in the following equation

$$x_j s_j = x_j(c_j - (A^T y)_j) = 0, \ \forall \ j.$$

Furthermore, we can rewrite the complementarity condition as $\sum_j x_j s_j = 0$. Since the primal and dual constraints ensure that each of the terms in the summation are nonnegative, i.e., $x_j s_j \geq 0$, if the entire sum is zero then every term $x_j s_j$ must be zero. The next theorem [29] is the theoretical basis of the complementarity condition for LO.

Theorem 1.7 (Complementary Slackness) Consider a pair of primal and dual LO problems (LP) and (LD). The primal feasible vector x is optimal for the primal problem and the dual feasible vector y is optimal for the dual problem *if and only if* $x^T(c - A^T y) = x^T s = 0$.

The theorem shows that at least one of x_j and s_j is zero, however, it is possible to have both $x_j = 0$ and $s_j = c_j - (A^T y)_j = 0$ for any $1 \leq j \leq n$. If exactly one of these two quantities is zero for all j, i.e., $x + s > 0$, then the optimal solution is said to be a *strictly complementary* solution and the pair (LP) and (LD) possesses the strict complementarity property.

The following theorem ensures the existence of a strictly complementary solution proved by Goldman and Tucker [34] in 1956. The Goldman-Tucker theorem plays a fundamental role in Interior-Point Methods.

Theorem 1.8 (Goldman-Tucker Theorem) If (LP) and (LD) are feasible then there exists a strictly complementary pair of optimal solutions, that is an optimal solution pair (x, s) satisfying $x + s > 0$.

At the end of this section, we introduce some notations for (LP) and (LD). Let

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

denote the optimal solution set of the primal-dual pair (LP) and (LD). Let

$$\mathcal{B}_{\widetilde{\mathcal{F}}} = \{j : \exists (x, s) \in \widetilde{\mathcal{F}} \ \text{ such that } \ x_j > 0\},$$

$$\mathcal{N}_{\widetilde{\mathcal{F}}} = \{j : \exists (x, s) \in \widetilde{\mathcal{F}} \ \text{ such that } \ s_j > 0\}.$$

Due to the Goldman-Tucker Theorem we have

$$\mathcal{B}_{\widetilde{\mathcal{F}}} \bigcap \mathcal{N}_{\widetilde{\mathcal{F}}} = \emptyset$$

and

$$\mathcal{B}_{\widetilde{\mathcal{F}}} \bigcup \mathcal{N}_{\widetilde{\mathcal{F}}} = \{1, \ldots, n\}.$$

In this chapter we have presented the duality results for a primal-dual pair of LO problems in the standard form. By using standard transformations, it is easy to see that the Strong Duality Theorem and also the Complementary Slackness Theorem hold between *any* pair of primal and dual problems.

Chapter 2

Interior-Point Methods for LO

Interior-Point Methods (IPMs) are currently one of the most active research areas in optimization. They are called *interior-point* methods because the points generated by the algorithms lie in the interior of the feasible region. This is in contrast with the simplex method that moves along the boundary of the feasible region from one extreme point to another. Today, IPMs for LO have become quite mature in theory, and have been applied to practical LO problems with extraordinary success as well.

In this chapter, a brief introduction to IPMs is presented. First, the basic features of IPMs are described, then some fundamental terms (Interior-point condition, central path, analytical center, etc.) and their properties are summarized. After these, the Primal-Dual Path-Following algorithm, which is one of the most important IPM, is discussed. Then we discuss step length and stopping criteria. Finally, we turn our attention to the homogeneous embedding model that acts as a pivotal role in our implementation.

2.1 Prelude

The modern era of interior-point methods has been come in 1984, when Karmarkar proposed his algorithm for LO [17]. In the years since then, much research was invested in IPMs, many algorithmic variants were developed, thousands of papers have been written on the subject, and several high-quality solvers based on IPMs for LO were implemented efficiently. Existing IPMs can be divided into four main classes [36]:

- Affine-Scaling Methods [36], which transform the problem via an "affine-scaling" transformation. They are known as a class of the simplest and still efficient IPMs. At each iteration (i) they construct an ellipsoid, called "The Dikin ellipsoid" in the interior of the feasible set, and then (ii) take a step in the direction which minimizes the objective function over the ellipsoid to obtain the next iterate.

- Projective Methods [36], such as Karmarkar's algorithm, use a more elaborate projective transformation. They start at an interior feasible point. At each iteration (i)

the LO problem is transformed via a *projective transformation*, to obtain an equivalent problem in a projected space, then (ii) a projected steepest-descent direction is computed, (iii) a step is taken along this direction, and (iv) the resulting point is mapped back to the original space.

- Path-Following Methods [36], are the most successful IPMs. They attempt to stay close to the barrier trajectory, the so-called *central path*, while approaching the solution set of the LO problem.
- Potential-Reduction Methods [36], attempt to obtain a reduction in some merit or potential function. There are various potential-reduction algorithms in the literature, such as the affine potential-reduction algorithm, the primal-dual potential-reduction algorithm, etc.

In this thesis we focus on primal-dual path-following IPMs and introduce the necessary terminology in the next section.

2.2 Fundamental Concepts and Results

In this section, we introduce some basic terms, concepts and some important results relevant to path-following IPMs.

Definition 2.1 For LO problems (LP) and (LD), a point triple (x^0, s^0, y^0) is said to satisfy the *interior point condition* (IPC), if (x^0, s^0, y^0) satisfy the following conditions:

$$\begin{aligned} Ax^0 &= b, \quad x^0 > 0, \\ A^T y^0 + s^0 &= c, \quad s^0 > 0. \end{aligned}$$

From the Strong Duality Theorem 1.5 and Complementarity Theorem 1.7, one can see that finding an optimal solution of (LP) and (LD) is equivalent to solving the following system[1]:

$$\begin{aligned} Ax &= b, \quad x > 0, \\ A^T y + s &= c, \quad s > 0, \\ xs &= 0. \end{aligned} \tag{2.1}$$

The basic idea of Primal-Dual IPMs is to replace the third equation in (2.1), the complementarity condition for (LP) and (LD), by the parameterized equation $xs = \mu e$, where e denotes the all-one vector[2] and $\mu > 0$. Thus we consider the system

$$\begin{aligned} Ax &= b, \quad x > 0, \\ A^T y + s &= c, \quad s > 0, \\ xs &= \mu e. \end{aligned} \tag{2.2}$$

[1]Here xs denotes the coordinatewise product of the vectors x and s, i.e., if $x = (x_1, x_2, \ldots, x_n)^T$ and $s = (s_1, s_2, \ldots, s_n)^T$, then $xs = (x_1 s_1, x_2 s_2, \ldots, x_n s_n)^T$.

[2]We denote $e = (1, 1, \ldots, 1)^T$.

The existence of a unique solution to system (2.2) is warranteed by the following Theorem[3]. We denote this unique solution by $(x(\mu), y(\mu), s(\mu))$.

Theorem 2.1 If the IPC holds, then for each $\mu > 0$, the parameterized system (2.2) has a unique solution.

Definition 2.2 If $(x(\mu), y(\mu), s(\mu))$ is the unique solution of (2.2), then $x(\mu)$ is called the *μ-center* of (LP) (or *primal μ-center*) and $(y(\mu), s(\mu))$ is called the *μ-center* of (LD) (or *dual μ-center*).

Definition 2.3 With μ running through all positive real numbers, the sets of primal μ-centers $\{x(\mu) : \mu > 0\}$ and dual μ-centers $\{(y(\mu), s(\mu)) : \mu > 0\}$ are called the *central path* of (LP) (or *primal central path*) and the *central path* of (LD) (or *dual central path*) respectively[4].

There is an interesting observation about the primal objective value, the dual objective value and the duality gap along the central paths. We describe it in the following proposition.

Proposition 2.2 [34] Let μ monotonically decrease to 0. Then along the primal central path the primal objective value $c^T x(\mu)$ is monotonically decreasing to the optimal objective value; along the dual central path the dual objective value $b^T y(\mu)$ is monotonically increasing to the optimal objective value; and the duality gap $c^T x(\mu) - b^T y(\mu)$ is monotonically decreasing to 0.

Moreover, we can have an obvious conclusion: if one of the LO problems (LP) and (LD) is infeasible, then the IPC can not be satisfied, thus the central paths do not exist. On the other hand, feasibility of both (LP) and (LD) is not enough for the existence of the central paths. In fact, the following theorem states the existence of the central paths accurately.

Theorem 2.3 [34] The central paths exist *if and only if* both the primal-dual LO problems (LP) and (LD) are feasible, the IPC holds and $\text{rank}(A) = m$.

The central path plays a key role both in the development of the theory and in the design of algorithms for LO problems. Figure 2.1 shows the central path of a small two dimensional problem.

Definition 2.4 The *analytic center* of $\widetilde{\mathcal{F}}$ is given by[5]

$$(x^*, s^*) = \arg \max_{(x,s)\in\widetilde{\mathcal{F}}} \prod_{j\in\mathcal{B}_{\widetilde{\mathcal{F}}}} x_j \prod_{j\in\mathcal{N}_{\widetilde{\mathcal{F}}}} s_j.$$

[3]This result was proved by McLinden [22], Kojima et al. [21]. See also Güler [15].

[4]The term *central path* was introduced by Sonnevend [35] and Megiddo [23], the limiting behavior of the central path as μ goes to zero has been a hot topic for some time. The properties of the central path for LO were first considered by Megiddo [23], and later by Güler and Ye [16].

[5]Here $\widetilde{\mathcal{F}}$, $\mathcal{B}_{\widetilde{\mathcal{F}}}$ and $\mathcal{N}_{\widetilde{\mathcal{F}}}$ are defined in Chapter 1.

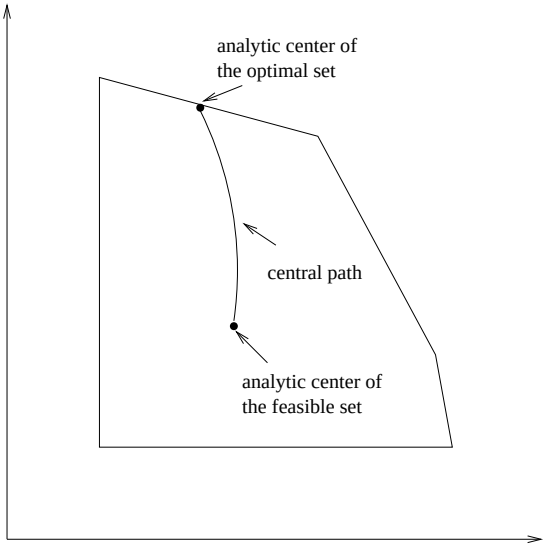

Figure 2.1 The central path and the analytic center

A nice property of the central path is that it starts at the analytic center[6] of the feasible set, and ends at the analytic center of the set of optimal solutions [34]. Therefore, we have the following theorem.

Theorem 2.4 The *central path* converges to the *analytic center* of the optimal set $\widetilde{\mathcal{F}}$ as $\mu \to 0$.

2.3 Primal-Dual Path-Following Algorithms

The basic idea of all primal-dual path-following algorithms is to trace the central path approximately by an iterative method. They use Newton's method to obtain a search directions at each iteration. There are a variety of primal-dual path-following algorithms in the literature. In this section we briefly investigate the classical primal-dual path-following Newton algorithm for LO.

[6]The notion of analytic center of a polyhedron was introduced by G. Sonnevend [35]. It can be defined as [34] :

Definition (Analytic Center) Let the nonempty and bounded set $\mathcal{T}$ be the intersection of an affine space in R^p with the nonnegative orthant of R^p. We define the support $\sigma(\mathcal{T})$ of $\mathcal{T}$ as the subset of the full index set $\{1, 2, \ldots, p\}$ given by

$$\sigma(\mathcal{T}) = \{j : \exists\, x \in \mathcal{T} \quad \text{such that} \quad x_j > 0\}.$$

The analytic center of $\mathcal{T}$ is defined as the zero vector if $\sigma(\mathcal{T}))$ is empty; otherwise it is the vector in $\mathcal{T}$ that maximizes the product

$$\prod_{j \in \sigma(\mathcal{T})} x_j, \quad x \in \mathcal{T}.$$

We start with a discussion of proximity functions. As observed by many researchers, the choice of the proximity function is crucial not only for the quality and elegance of the analysis, but also for the performance of the algorithm. Several proximities have been introduced and used in the IPM literature [34, 39, 42] and these proximities are usually defined in a scaled primal-dual space. For simplicity, let us first introduce the notation v and v^{-1}. For any strictly feasible primal-dual pair (x, s) and any positive number μ, we define

$$v := \sqrt{\frac{xs}{\mu}}, \quad v^{-1} := \sqrt{\frac{\mu e}{xs}}. \tag{2.3}$$

The i^{th} component of the vector v is $\sqrt{\frac{x_i s_i}{\mu}}$ and the i^{th} component of the vector v^{-1} is $\sqrt{\frac{\mu}{x_i s_i}}$. Note that by using these notations, we can rewrite the centrality condition as an equation of v:

$$xs = \mu e \Leftrightarrow v^2 = e \Leftrightarrow v = e.$$

The following are two popular proximity functions for primal-dual IPMs [34, 39, 42]:

$$\delta(xs, \mu) := \frac{1}{\sqrt{2}} \|v - v^{-1}\|, \tag{2.4}$$

$$\Phi(xs, \mu) := \sum_{i=1}^{n} \phi(v_i) = \frac{x^T s}{2\mu} - \frac{n}{2} + \frac{n \log \mu}{2} - \frac{1}{2} \sum_{i=1}^{m} \log(x_i s_i), \tag{2.5}$$

where

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

It is easy to see that both measures vanish if $v = e$, and go to infinity if v approaches the boundary of the nonnegative orthant. This is the *barrier property* of the proximities. The measure Φ is the primal-dual *logarithmic barrier function* with *barrier parameter* μ. We will discuss a new class of proximity functions called *self-regular proximities* in Chapter 3.

In order to build our iterative process, we should solve the system (2.2) approximately. Generally speaking, for any $\mu > 0$, the exact unique solution for the system (2.2) can not be computed easily. We usually make *Newton steps* to get "close to" the solutions. Given (x, s, y) that satisfy the IPC, for a particular $\mu > 0$, we want to solve

$$\begin{aligned} A(x + \Delta x) &= b, \\ A^T(y + \Delta y) + (s + \Delta s) &= c, \\ (x + \Delta x)(s + \Delta s) &= \mu e, \end{aligned} \tag{2.6}$$

and get the so-called *search direction* $(\Delta x,\ \Delta s,\ \Delta y)$.

This system is still nonlinear in Δx and Δs, thus we neglect the second order term $\Delta x \Delta s$, and we get the *Primal-Dual Newton System*

$$\begin{aligned} A\Delta x &= 0, \\ A^T \Delta y + \Delta s &= 0, \\ s\Delta x + x\Delta s &= \mu e - xs. \end{aligned} \tag{2.7}$$

When $x > 0$, $s > 0$ and $\text{rank}(A) = m$, the system (2.7) has a unique solution $(\Delta x,\ \Delta s,\ \Delta y)$.

Let $\Psi(xs, \mu)$ be a proximity measure, now we can describe the Classical Primal-Dual Newton Method for LO as follows.

Classical Primal-Dual Newton Method for LO

Input:

A proximity parameter τ;

an accuracy parameter $\epsilon > 0$;

a fixed barrier update parameter θ, $0 \leq \theta \leq 1$;

$(x^0, s^0) > 0$, y^0, and $\mu^0 = 1$ such that $\Psi(x^0 s^0, \mu^0) \leq \tau$.

begin

$x := x^0$; $s := s^0$; $y := y^0$; $\mu := \mu^0$;

while $n\mu \geq \epsilon$ **do**

begin

$\mu := (1 - \theta)\mu$;

while $\Psi(xs, \mu) \geq \tau$ **do**

begin

Solve system (2.7) for $\Delta x,\ \Delta s,\ \Delta y$;

Determine step size α by some rules;

$x := x + \alpha \Delta x$;

$s := s + \alpha \Delta s$;

$y := y + \alpha \Delta y$;

end

end

end

Now we will briefly investigate how this algorithm goes. We first need to define a *neighborhood* of the central path:

$$\mathcal{N}(\tau,\mu) = \{(x,s) > 0 : Ax = b,\ A^T y + s = c,\ \Psi(xs,\mu) \leq \tau\},$$

where $\Psi(xs,\mu)$ is the proximity function to measure the distance from the point (x,s) to $(x(\mu), s(\mu))$, it can be defined as (2.4), (2.5) or others; τ is the *radius of the neighborhood*, $\tau > 0$, and it determines the size of the neighborhood. Figure 2.2 illustrates a neighborhood of the central path in the v-space.

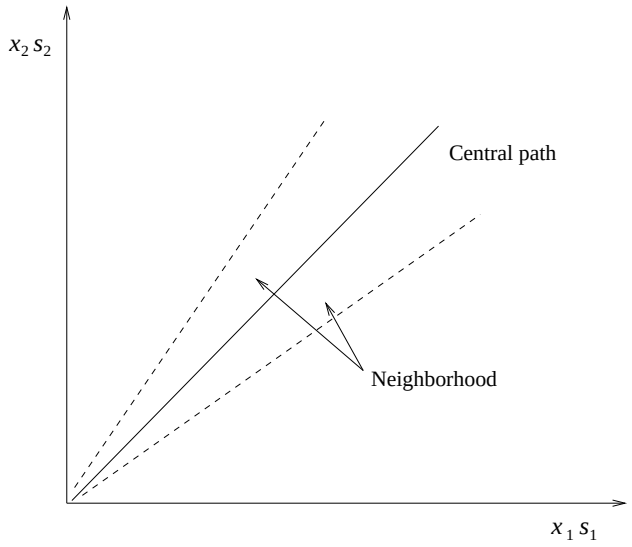

Figure 2.2 Neighborhood of the central path

In classical primal-dual algorithms, the proximity parameter τ and the accuracy parameter ϵ are constants, they can be chosen without much difficulty[7].

Another important parameter in the algorithm is the choice of the parameter θ. Sometimes the parameter θ is not fixed but chosen as a number in the interval $[0,1]$ corresponding to the present iterate[8]. Usually, if θ is a constant independent of n the dimension of the problem, for instance $\theta = 0.5$, then we call the algorithm a *large-update* (or long-step) method. If θ depends on the problem dimension such as $\theta = \frac{1}{2\sqrt{n}}$, then the algorithm is named a *small-update* (or short-step) method. At present, there is still a gap between the practical performance and the theoretical worst-case complexity of these two classes of IPMs in the literature. Small-update methods have the best theoretical performance while they are hopelessly slow in practice; on the other hand, large-update methods have worse theoretical complexity bounds while they are the most efficient methods in practice. Table 2.1 summarize the gap between theory and practice.

[7]For example, we can choose $\tau = 0.5$ [34], and $\epsilon = 10^{-8}$.

[8]In the case $\theta = 1$, the solution in the iteration is called the *predictor direction*. If $\theta = 0$, we obtain the so-called *corrector direction*.

Table 2.1 Comparison of Large-update and small-update methods

Method	θ	Iteration bound	Performance in practice
Large update	$1/2$	$O(n \log n/\epsilon)$	Efficient
Small update	$1/\sqrt{n}$	$O(\sqrt{n} \log n/\epsilon)$	Very poor

Moreover, Figures 2.3 and 2.4 demonstrate the practical performance of IPMs with large-update and small-update for a specific two-dimension LO problem. These figures are drawn in the so-called v-space.

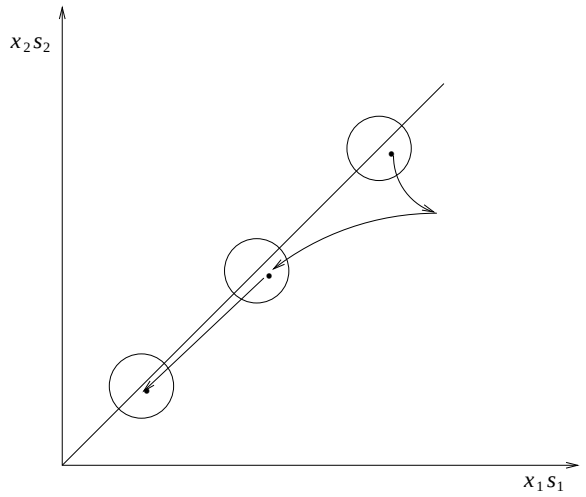

Figure 2.3 Performance of a large-update IPM.

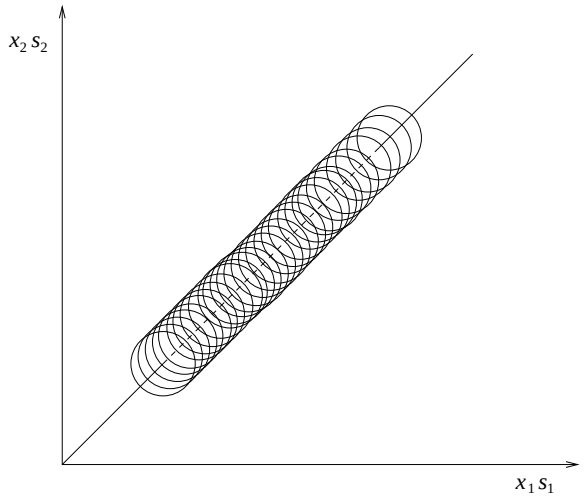

Figure 2.4 Performance of a small-update IPM.

By solving the Newton system (2.7), we can get search direction $(\Delta x,\ \Delta s,\ \Delta y)$. Then we want to take a step along the search direction to get a new point (x, s, y). Thus, we

should compute the *step size* α by some line search rules, then construct a new point (x, s, y) with $x := x + \alpha\Delta x$, $s := s + \alpha\Delta s$ and $y := y + \alpha\Delta y$. For some IPMs, the step size is chosen very carefully so that it will always keep the iterate strictly feasible while reducing the duality gap as much as possible. A simple and practical way to deal with the step size α is that, in each iteration, we compute the maximal feasible step size (α_{max}), then damp it by a certain ratio (for instance $0.99\alpha_{max}$) and accept the result as a step size. In fact, the idea of utilizing a fixed factor of the maximal step size as the working step size has been widely employed in most IPM packages and works extremely well [2]. We will discuss this in more detail in the next section.

One can observe that we just use a simple formula $\mu = (1-\theta)\mu$ to update parameter μ in each outer iteration in the classical primal-dual algorithm proposed in this section. Then we solve system (2.7) in the inner iteration, and μ acts as an independent parameter in the inner process. This model has been employed for most algorithms in the book by Roos, Terlaky and Vial [34]. We elaborate a little further on this topic. In the IPM literature, there are a certain number of rules to update μ. For example, in some IPMs, μ is updated with respect to the present duality gap, i.e., $\mu = \frac{x^T s}{n}$. After the update of μ, then we want to check whether the current iterate is in the neighborhood $\mathcal{N}(\tau, \mu)$ of the new center $(x(\mu), s(\mu), y(\mu))$. If the answer is 'yes', then go to the outer process and update μ by a certain rule. Otherwise, we need to run the inner process until the iterate enters the neighborhood $\mathcal{N}(\tau, \mu)$ again. Then we go back to an outer iteration, and so on. This process is repeated until an approximate solution to the problem is obtained. Moreover, most practical algorithms then construct an exact solution by resorting to a rounding procedure as described by Ye [41].

After we update the parameter μ at each outer iteration, we need to check if the stopping criteria is met, i.e., when will the loop stop? Thus we need to give some stopping criteria, we leave this discussion in Section 2.5.

So far we have described most parts of the classical primal-dual algorithm, and understand how this algorithm goes. Last, but not least, we still need to discuss two important issues: 1) how to find a strictly feasible initial point (x^0, s^0, y^0); 2) how to detect the infeasibility or unboundedness of the underlying LO problem. We leave this to Section 2.6.

2.4 Step-length

The basic mechanism in IPM algorithms is iterative. In each iteration, after we get a search direction, we need to compute the step size. In practice we choose the step size as follows:

First, compute the maximal step size to the boundary given by

$$\alpha_{max} := \arg\max_{\alpha \geq 0} \{(x,\ \tau,\ s,\ \kappa) + \alpha(\Delta x,\ \Delta\tau,\ \Delta s,\ \Delta\kappa) \geq 0\}.$$

Then the step size is slightly reduced with a factor $\alpha_0 = 0.99995$ to ensure that the new

point is strictly positive, i.e., $\alpha = \alpha_0 \alpha_{max}$. Some codes use smaller α_0 in those iterations in case that 0.99995 is too aggressive. However, in most cases this aggressive choice of α_0 seems to be the best.

In general, the algorithm cannot be guaranteed to be globally convergent with the choice $\alpha_0 = 0.99995$. However, Kojima, Megiddo and Mizuno [19] has proved global convergence of a variant of the primal-dual method that allows the aggressive choice of α_0 in most iterations.

2.5 Stopping Criteria

Interior point algorithms terminate when the duality gap is small enough and the current solution is feasible for the original problems (LP) and (LD), or when the infeasibility is small enough. The practical tolerances are larger than the theoretical bounds that guarantee identification of an exact solution; this is a common drawback of all numerical algorithms for solving LO problems. To obtain a sensible solution the duality gap and the measure of infeasibility should be related to the problem data. Relative primal infeasibility is related to the length of the vector b, dual infeasibility is related to the length of the vector c, and the duality gap is related to the actual objective value. A solution with p digits relative accuracy is guaranteed by the stopping criteria presented here:

$$\frac{\| Ax - b \|}{1 + \| b \|} \leq 10^{-r},$$
$$\frac{\| A^T y - c \|}{1 + \| c \|} \leq 10^{-r},$$
$$\frac{| c^T x - b^T y |}{1 + | b^T y |} \leq 10^{-r},$$

where r is the number of digits accurate in the solution. An 8-digit exact solution ($r = 8$) is typically required in the literature.

2.6 Homogeneous Self-Dual Embedding Strategy

Most interior-point algorithms need a strictly feasible point as the starting point. However, the complexity of obtaining such a point is as difficult as solving the LO problem itself. Moreover, a complete LO algorithm should accomplish the following two tasks:

(1) Identify the optimal solution if the LO problem is solvable;

(2) Detect the infeasibility or unboundedness when the LO problem is infeasible or unbounded.

There are various approaches to resolve the above issues in the LO literature. These include 1) the big M method which adds one or more artificial column(s) and/or row(s) and

a huge penalty parameter M to force solutions to become feasible during the algorithm; 2) combining the primal and dual problem into a single linear feasible problem (2.1), then applying LO algorithms to find a feasible point of the problem; 3) Phase I–then–Phase II method, i.e., first try to find a feasible point (and possibly one for the dual problem), and then start to look for an optimal solution if the problem is feasible and bounded [42].

In this section, we discuss the most robust and efficient way: the so-called *homogeneous self-dual embedding* model, which is based on the construction of a homogeneous and self-dual LO model related to (LP) and (LD). The homogeneous self-dual embedding model was firstly introduced by Ye, Todd and Mizuno [43]. This model originated from the homogeneous model of Goldman and Tucker [13, 37]. Its main idea is: first embedding the LO problem into a slightly larger problem that always has a strictly feasible initial point, then we can obtain the optimal solution of the original LO problem easily from the solution of the embedded problem if the original LO problem has an optimal solution. Otherwise we get a certificate for the infeasibility of the original (LP) or (LD) problems. The homogeneous self-dual embedding algorithm has the following advantages and appealing features:

- It solves the LO problem without any assumption;
- It can be solved in $O(\sqrt{n}\log\frac{n}{\epsilon})$ iterations, i.e., it keeps the currently known best polynomial complexity result;
- In each iteration, it needs the solution of a Newton system whose size is nearly the same as the Newton system solved in the original problem, i.e., it retains the complexity of bit operations;
- It can efficiently detect the infeasibility or unboundedness for either (LP) or (LD).

We start with the definition of Self-duality.

Definition 2.5 LO problem is called *Self-dual* if the dual of the problem is equivalent to the primal. In other words, if the LO problem can be written as

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Qx \geq -c \\ & x \geq 0 \end{array}$$

with[9] $Q^T = -Q$, then it is *self-dual*.

An advantage of self-duality is that we can apply a primal-dual interior-point algorithm to solve the self-dual problem without doubling the dimension of the linear system solved at each iteration.

[9]If $Q^T = -Q$, the matrix Q is called *skew-symmetric*.

Before we discuss the homogeneous self-dual embedding model, let us first introduce the following homogeneous system:

$$\begin{array}{rrrl} Ax & -b\tau & = 0, & \\ -A^T y & +c\tau & \geq 0, & \\ b^T y - c^T x & & \geq 0, & \\ y \in R^m, \ x \geq 0, & \tau \geq 0. & & \end{array} \tag{2.8}$$

Remark: It is worthwhile to note that in system (2.8), variable τ is called a homogenizing variable, since the system is a homogeneous system obtained by homogenizing the optimality conditions by using the variable τ.

One can easily check that system (2.8) has the following features:

- System (2.8) is a homogeneous linear equality and inequality system, and it has zero as its trivial solution.
- System (2.8) can be regarded as an LO problem with zero objective function and a zero right hand side.
- Considered as an LO problem, system (2.8) is self-dual, i.e., its dual is equivalent to itself[10].
- Considered as an LO problem, system (2.8) does not satisfy the interior-point condition.

We need to modify system (2.8) to keep self-duality and have interior points. We can do that by choosing any $x^0 > 0,\ s^0 > 0$, and $y^0 \in R^m$ and modify the problem by introducing the error with respect to x^0, s^0, y^0. In fact, system (2.8) can be transformed into the following homogeneous[11] self-dual LO problem (HLP):

$$\begin{array}{lll} (\mathbf{HLP}) & \min & \beta\nu \\ & \text{s.t.} & \begin{array}{rrrrl} Ax & -b\tau & -r_p\nu & = 0, & \\ -A^T y & +c\tau & -r_d\nu & \geq 0, & \\ b^T y - c^T x & & -r_g\nu & \geq 0, & \\ r_p^T y + r_d^T x & +r_g\tau & & = -\beta, & \\ y \in R^m, \ x \geq 0, & \tau \geq 0, & \nu \in R, & & \end{array} \end{array} \tag{2.9}$$

where r_p, r_d and r_g represent the *infeasibility* of the initial primal and dual points, and the *duality gap*, respectively. For given $x^0 > 0$, $s^0 > 0$, y^0, $\tau^0 > 0$, $\kappa^0 > 0$, and $\nu^0 > 0$,

[10] One can verify this conclusion by using the rules proposed in the scheme for dualizing in Chapter 1.

[11] For a homogeneous LO, we do not mean that all constraints must be homogeneous, or equivalently all right-hand sides be zero. We allow a single inhomogeneous constraint, often called a *normalizing constraint*.

we have

$$\begin{aligned} r_p &= \frac{Ax^0 - b\tau^0}{\nu^0}, \\ r_d &= \frac{c\tau^0 - A^T y^0 - s^0}{\nu^0}, \\ r_g &= \frac{-c^T x^0 - \kappa^0 + b^T y^0}{\nu^0}, \\ \beta &= -r_p^T y^0 - r_d^T x^0 - r_g \tau^0. \end{aligned} \tag{2.10}$$

Remark: In system (2.9), we introduce the artificial variable ν with appropriate coefficients to the system, and also add the fourth constraint to get a skew-symmetric coefficient matrix and achieve self-duality.

By introducing the slack variables s and κ, the (HLP) problem can be written as follows:

$$\begin{array}{lll} (\mathbf{HLP}) & \min & \beta\nu \\ & \text{s.t.} & \begin{array}{rrrrrr} Ax & -b\tau & -r_p\nu & & & = 0, \\ -A^T y & +c\tau & -r_d\nu & -s & & = 0, \\ b^T y - c^T x & & -r_g\nu & & -\kappa & = 0, \\ r_p^T y + r_d^T x & +r_g\tau & & & & = -\beta, \end{array} \\ & & y \text{ is free},\ x \geq 0,\ \tau \geq 0,\ \nu \in R,\ s \geq 0,\ \kappa \geq 0, \end{array} \tag{2.11}$$

where r_p, r_d, r_g and β are defined by (2.10).

Observe that the first three constraints in (HLP), with $\tau = 1$ and $\nu = 0$, represent primal and dual feasibility (with $x \geq 0$) and reversed weak duality, therefore, if such a solution is known, then they define primal and dual optimal solutions. Also note that the constraints of (HLP) form a skew-symmetric system, thus it ensure that the system (2.11) is a self-dual LO problem.

Remark: Observe that if we simply choose

$$x^0 = e, \quad s^0 = e, \ \text{ and } \ y^0 = 0,$$

then, the corresponding homogeneous LO problem has a strictly feasible solution on the central path with $\mu^0 = 1$, $\beta = n + 1$.

Now let us denote by (HLD) the dual of (HLP). Denote by y' the dual multiplier vector for the first constraint, by x' the dual multiplier vector for the second constraint, by τ' the dual multiplier for the third constraint, by ν' the dual multiplier for the fourth constraint. Furthermore, let us denote by $\mathcal{F}_h$ the set of all points $(y, x, \tau, \nu, s, \kappa)$ that are feasible for (HLP), denote by $\mathcal{F}_h^0$ the set of strictly feasible points with $(x, \tau, s, \kappa) > 0$ in $\mathcal{F}_h$. Then we have the following result [42].

Theorem 2.5 Consider problems (HLP) and (HLD).

(i) (HLD) has an equivalent form to (HLP), i.e., (HLD) is simply (HLP) with $(y,\ x,\ \tau,\ \nu)$ being replaced by $(y',\ x',\ \tau',\ \nu')$.

(ii) (HLP) has a strictly feasible point

$$y = y^0,\ x = x^0 > 0,\ \tau = 1,\ \nu = 1,\ s = s^0 > 0,\ \kappa = 1.$$

(iii) (HLP) has an optimal solution and its optimal solution set is bounded.

(iv) The optimal value of (HLP) is zero, and

$$(y,\ x,\ \tau,\ \nu,\ s,\ \kappa) \in \mathcal{F}_h \quad \text{implies that} \quad \beta\nu = x^T s + \tau\kappa.$$

(v) There is an optimal solution $(y^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ s^*,\ \kappa^*) \in \mathcal{F}_h$ such that

$$\begin{pmatrix} x^* + s^* \\ \tau^* + \kappa^* \end{pmatrix} > 0.$$

Let us consider systems (2.11) and (2.10) again, without loss of generality, suppose a certain point $(x^0,\ y^0,\ s^0)$ is selected, one can easily verify that: if x^0 is feasible in (LP), then by (2.10) r_p is zero, and thus for any feasible solution x to (HLP) with $\tau > 0$, x/τ is feasible for (LP); if $(y^0,\ s^0)$ is feasible for (LD), then by (2.10) r_d is zero, and then for any feasible solution $(y^0,\ s^0)$ to (HLP) with $\tau > 0$, $(y^0,\ s^0)/\tau$ is feasible for (LD)[12]; Moreover, the following theorem demonstrates the relationship between the optimal solutions of (HLP) and those of (LP) and (LD), here we just reference it without proof [42].

Theorem 2.6 Let $(y^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ s^*,\ \kappa^*)$ be a strictly self complementary solution[13] for (HLP).

(i) (LP) has an optimal solution if and only if $\tau^* > 0$. In this case, $\frac{x^*}{\tau^*}$ is an optimal solution for (LP) and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau^*})$ is an optimal solution for (LD).

(ii) (LP) has no optimal solution if and only if $\kappa^* > 0$. In this situation, if $c^T x^* < 0$ then (LD) is infeasible; if $-b^T y^* < 0$ then (LP) is infeasible; and if both $c^T x^* < 0$ and $-b^T y^* < 0$ hold, then both (LP) and (LD) are infeasible.

[12]If $r_g > 0$, then for any feasible solution $(x^0,\ y^0,\ s^0)$ to (HLP) with $\nu > 0$ and $\tau > 0$, we have

$$c^T x - b^T y \le -r_g \nu < 0,$$

thus by the Weak Duality Theorem 1.2, either x/τ or $(y^0,\ s^0)/\tau$ must be infeasible.

[13]If $(y^*,\ x^*,\ \tau^*,\ \nu^* = 0,\ s^*,\ \kappa^*)$ is an optimal solution of a self-dual problem (HLP) with

$$\begin{pmatrix} x^* + s^* \\ \tau^* + \kappa^* \end{pmatrix} > 0,$$

then it is called a *strictly self complementary solution* for (HLP).

At the end of this section, we mention that we can easily obtain a strictly feasible initial point for the embedding problem (HLP), but the computational cost of a Newton step for the new problem (HLP) is slightly bigger than calculating a Newton step for the original problem (LP). For the problem (HLP), we need to make two additional rank-1 updates (see Section 4.1 for details). In practice, one use a so-called *simplified model* [34] which requires only one additional rank-1 update at each iteration. The simplified model is essentially equivalent to the original model. This is first observed by Roos, Terlaky and Vial [34]. We will discuss these issues in more detail in Chapter 4.

Chapter 3

New IPMs Based on Self-Regular Proximities

Proximity measures or potential functions serve a crucial role in both the theoretical study and the practical implementation of IPMs. We introduced two popular proximity measures used in the literature for primal-dual IPMs in Section 2.3. It is worthwhile to note that other variants of the proximity measure $\Psi(xs, \mu)$ were used in the literature (see [20, 26, 34, 39, 42]). In this chapter we discuss a new class of proximity measures that was first introduced by Peng, Roos, and Terlaky, the class of so-called *Self-Regular* Proximity measures. All material in this chapter is extracted from the paper [31] and the book [32].

3.1 Prelude

From the discussions in the previous chapters, one may notice that we need to keep control on the *distance* of the current iterates to the current μ-center in both the analysis and implementation of IPMs. This means that we need to quantify the *distance* from the vector xs to the vector μe by using some proximity measures. In fact, the selection of the proximity measure is very important for both the performance and elegance of the analysis of IPMs. Before describing the proximity measure, for ease of reference, let us introduce the following notations:

$$d_x := \frac{v\Delta x}{x}, \quad d_s := \frac{v\Delta s}{s}, \tag{3.1}$$

$$\bar{d}_x := \frac{\Delta x}{x}, \quad \bar{d}_s := \frac{\Delta s}{s}. \tag{3.2}$$

Here v and v^{-1} are defined in (2.3).

As we mentioned in Section 2.3, by using the so-called v-space, we can rewrite the centrality condition as $v = e$ or, equivalently coordinatewise, $v_i = 1$ for all $i = 1,\ 2,\ ...,\ n$.

Moreover, by using the above notations, one can rewrite the last equation $s\Delta x + x\Delta s = \mu e - xs$ of system (2.7) as follows (just multiply both sides by $\frac{v}{xs}$):

$$d_x + d_s = v^{-1} - v.$$

Furthermore, let us denote by $\bar{A} = \frac{1}{\mu}AV^{-1}X,\ V = \text{diag}(v),\ X = \text{diag}(x)$. Then we can rewrite the original Newton system (2.7) for the LO problem as the following new Newton system in the scaled v-space.

$$\begin{aligned} \bar{A}d_x &= 0, \\ \bar{A}^T\Delta y + d_s &= 0, \\ d_x + d_s &= v^{-1} - v. \end{aligned} \tag{3.3}$$

Let us denote the search direction in the scaled v-space by $d_v = d_x + d_s = v^{-1} - v$. From (3.3) we can observe that d_x and d_s are the orthogonal decomposition of the direction d_v. Moreover, one can see that the vector d_v is the steepest descent direction with respect to the proximity $\Phi(xs, \mu)$ (see Section 2.5 for the definition of $\Phi(xs, \mu)$) in the $v-$space.

3.2 Univariate Self-Regular Functions

In this section our primary goal is to introduce a univariate function defined on R_{++} that can be used to define a proximity measure in the $v-$space. Thus, the function should satisfy two basic requirements: 1) it attains its global minimum at 1; 2) it can be used to measure the distance from any point in R_{++} to 1. Moreover, it is also desirable for the function to enjoy certain barrier property that prevents the argument from moving to the boundary of R_{++}. Let us first give the definition of a univariate *Self-Regular* function.

Definition 3.1 The function[1] $\psi(t) \in C^2 :\ (0,\ \infty) \to R$ 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

$$\nu_1(t^{p-1} + t^{-1-q}) \leq \psi''(t) \leq \nu_2(t^{p-1} + t^{-1-q}),\ \ \forall\, t\ \in\ (0,\ \infty). \tag{3.4}$$

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

$$\psi(t_1^r t_2^{1-r}) \leq r\psi(t_1) + (1-r)\psi(t_2),\ \ r \in [0,\ 1]. \tag{3.5}$$

If a univariate function $\psi(t)$ is Self-Regular, then the parameters q and p are called the *barrier degree* and the *growth degree* of the function, respectively.

Remark: It is worthwhile to point out that $\psi(t)$ is Self-Regular *if and only if* $\psi(t)/\nu_1$ is Self-Regular. Therefore, without loss of generality, we may assume $\nu_1 = 1$.

[1] In this thesis, we denote the set of twice continuously differentiable functions by C^2.

In SR1, there is a condition $\psi(1) = \psi'(1) = 0$, by simply using integration, one can get an immediate consequence as follows:

$$\psi(t) = \int_1^t \int_1^\xi \psi''(\zeta) d\zeta d\xi \,. \tag{3.6}$$

Let us now consider the special case $\nu_1 = \nu_2 = 1$. Then (3.4) implies to

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

hence, we know that $\psi(t)$ is uniquely determined by p and q. The unique self-regular function obtained in this way plays an important role in this chapter. For convenience, we denote it by $\Upsilon_{p,q}(t)$. With $p \geq 1$, one may easily get the following equations by simply integrating twice.

$$\Upsilon_{p,1}(t) = \frac{t^{p+1} - 1}{p(p+1)} - \frac{\log t}{q} + \frac{p-1}{p}(t-1) \tag{3.7}$$

$$\Upsilon_{p,q}(t) = \frac{t^{p+1} - 1}{p(p+1)} + \frac{t^{1-q} - 1}{q(q-1)} + \frac{p-q}{pq}(t-1), \quad q > 1. \tag{3.8}$$

Since

$$\lim_{q \to 1} \frac{t^{1-q} - 1}{q-1} = -\log t,$$

one can easily verify that

$$\lim_{q \to 1} \Upsilon_{p,q}(t) = \Upsilon_{p,1}(t).$$

In the special case $p = q = 1$, one obtain[2]:

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

Before we discuss more results about the functions $\Upsilon_{p,q}(t)$, we first observe some of their characteristics. Figure 3.1 demonstrates the growth behaviors and barrier behaviors of two functions $\Upsilon_{1,1}(t)$ and $\Upsilon_{1,3}(t)$. From this figure, it is clear that when $t \to \infty$, the growth behaviors of these two functions are very similar; on the other hand, as $t \to 0$, the barrier behavior of the function $\Upsilon_{1,3}(t)$ is much stronger than that of $\Upsilon_{1,1}(t)$.

[2]This is the well-known univariate logarithmic barrier function [34].

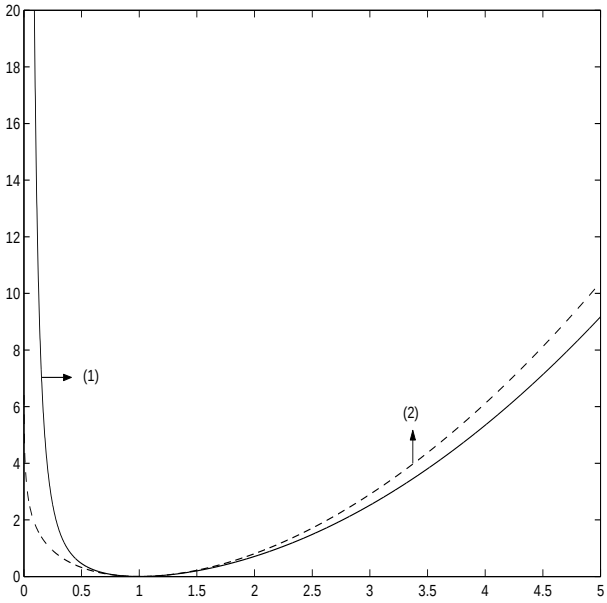

Figure 3.1 Demonstration of Self-Regular functions, (1)-$\Upsilon_{1,3}(t)$, (2)-$\Upsilon_{1,1}(t)$.

After these definitions, an immediate question appears: is the function $\Upsilon_{p,q}(t)$ Self-Regular or not? From now we will focus on this topic. For convenience of reference, let us first denote by Ω_1 and Ω_2 the sets of functions whose elements satisfy conditions SR1 and SR2, respectively.

In fact, one can easily verify that: 1) $\Upsilon_{p,q}(1) = \Upsilon'_{p,q}(1) = 0$; 2) $\Upsilon''_{p,q}(t)$ satisfy (3.4) with $\nu_1 = \nu_2 = 1$, since $\Upsilon''_{p,q}(t) = t^{p-1} + t^{-1-q}$. Thus we can conclude that $\Upsilon_{p,q}(t)$ satisfy condition SR1, i.e., $\Upsilon_{p,q}(t) \in \Omega_1$.

At this moment, we still do not know if $\Upsilon_{p,q}(t)$ satisfy condition SR2. To get to this conclusion, let us cite some results from [32].

Lemma 3.1 A function $\psi(t) \in C^2 : \ (0, \ \infty) \to R$ belongs to Ω_2 if and only if the function $\psi(\exp(\zeta)) : \ R \to R$ is convex in ζ, or equivalently $\psi'(t) + t\psi''(t) \geq 0$ for $t > 0$.

Lemma 3.2 Suppose that $\psi(t) \in \Omega_2$. Then for any $\alpha \in R$, $\psi(t^\alpha) \in \Omega_2$.

Lemma 3.3 Let N be any positive integer, $\beta_0 \in R$, $\beta_i \geq 0$, $\rho_i \in R$, $i = 1, 2, ..., N$ and

$$\psi(t) = \beta_0 \log t + \sum_{i=1}^{N} \beta_i (t^{\rho_i} - 1),$$

then $\psi(t) \in \Omega_2$.

Theorem 3.4 establishes the direct result.

Theorem 3.4 The function $\Upsilon_{p,q}(t)$ is self-regular when $p, \ q \geq 1$.

Proof: We want to show that $\Upsilon_{p,q}(t) \in \Omega_2$. By simple calculation, we get

$$\Upsilon'_{p,q}(t) + t\Upsilon''_{p,q} = \frac{p+1}{p}t^p + \frac{q-1}{q}t^{-q} + \frac{1}{q} - \frac{1}{p}.$$

Hence, for any $p \geq q \geq 1,\ t > 0$, we have

$$\Upsilon'_{p,q}(t) + t\Upsilon''_{p,q} > \frac{1}{q} - \frac{1}{p} \geq 0.$$

On the other hand, when $q > p \geq 1$, one has

$$1 - \frac{1}{q} \geq \frac{1}{p} - \frac{1}{q}.$$

Furthermore, for any $t > 0$, we have

$$\begin{aligned}\Upsilon'_{p,q}(t) + t\Upsilon''_{p,q} &= \frac{p+1}{p}t^p + \frac{q-1}{q}t^{-q} + \frac{1}{q} - \frac{1}{p} \\ &\geq \frac{p+1}{p}t^p + \frac{q-p}{pq}t^{-q} + \frac{1}{q} - \frac{1}{p} \\ &> (\frac{1}{p} - \frac{1}{q})(t^p + t^{-q} - 1) > 0,\end{aligned}$$

i.e., $\forall\ q > p \geq 1,\ t > 0$, we have

$$\Upsilon'_{p,q}(t) + t\Upsilon''_{p,q} \geq 0.$$

Combining the above two parts, one has that for $p,\ q \geq 1,\ t > 0$

$$\Upsilon'_{p,q}(t) + t\Upsilon''_{p,q} \geq 0.$$

Thus by Lemma 3.1, we have: $\Upsilon_{p,q}(t) \in \Omega_2$, and by the previous discussions we also know that $\Upsilon_{p,q}(t) \in \Omega_1$. Therefore the proof is completed. □

The next two Lemmas provide us two ways to generate self-regular functions.

Lemma 3.5 If the function $\psi_1(t),\ \psi_2(t)$ are self-regular, then any combination $\beta_1\psi_1 + \beta_2\psi_2$ with $\beta_1,\ \beta_2 \geq 0,\ \ \beta_1 + \beta_2 > 0$ is self-regular.

Lemma 3.6 If $\psi(t) = \psi(t^{-1})$ and $\psi(t) \in \Omega_1$, then $\psi(t)$ is self-regular.

So far, we have proved that $\Upsilon_{p,q}(t)$ provides a family of Self-Regular functions. One may ask: is this the only family of Self-Regular functions? The answer is 'No'. Let us proceed with other examples.

Example 1 Consider the function

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

Then we have

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

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

Obviously, $\psi(t) \in \Omega_1$, i.e., it satisfies condition SR1 (with $\nu_1 = \min(p, q)$, $\nu_2 = \max(p, q)$). On the other hand, we have:

$$\psi'(t) + t\psi''(t) = (p+1)t^p + (q-1)t^{-q} \geq 0.$$

By Lemma 3.1, this shows that: $\psi(t) \in \Omega_2$, thus $\psi(t)$ is self-regular. Note that if $p = q$ then $\psi(t) = p\Upsilon_{p,p}(t)$, otherwise the functions $\psi(t)$ and $\Upsilon_{p,q}(t)$ are linearly independent.

Example 2 Consider the function

$$\psi(t) = \frac{1}{2}(t - t^{-1})^2.$$

It is easy to verify that this is a special case of the function introduced in Example 1 where $p = 1$ and $q = 3$. Hence, self-regularity follows.

When we verify if a function is Self-Regular, we need to check both SR1 and SR2. Otherwise the conclusion may not be correct. Actually, some functions satisfy SR1 but not SR2, while others satisfy SR2 but not SR1. Now we give some examples to show that SR1 and SR2 are two independent conditions.

Example 3 Consider the function

$$\psi(t) = t^2 - t - \log t.$$

One can easily check that it satisfies SR1, with $p = \nu_1 = 1$ and $q = 1$, $\nu_2 = 2$. On the other hand, by simple computation, we have

$$\psi'(t) + t\psi''(t) = 4t - 1.$$

When $t < \frac{1}{4}$, we get

$$\psi'(t) + t\psi''(t) < 0,$$

therefore, $\psi(t)$ does not satisfy SR2.

Example 4 Consider the function

$$\psi(t) = t^2.$$

We can check easily that it satisfy SR2 but not SR1.

3.3 Basic Properties of Univariate Self-Regular Functions

In this section, we proceed with a discussion about the relations between a function $\psi(t)$ and its first and second derivatives when $\psi(t) \in \Omega_1$. We take some results from [32] without proof.

Lemma 3.7 Suppose that $\psi(t) \in \Omega_1$. Then

$$| \frac{1}{t}\psi'(t) | \leq \frac{\nu_2}{\nu_1}\psi''(t), \quad t > 0.$$

Lemma 3.8 Suppose that the function $\psi(t) \in \Omega_1$ with $q > 1$. Then there is a constant C_ν depending on the constants ν_1, ν_2, p and q such that

$$\psi(t)\psi''(t) \leq C_\nu \psi'(t)^2, \quad \forall\, t > 0.$$

Now we give some bounds on a function $\psi(t) \in \Omega_1$ in terms of t and $\psi'(t)$.

Lemma 3.9 Suppose that $\psi(t) \in \Omega_1$. Then

$$\begin{aligned}
\frac{1}{2}(t-1)^2 &\leq \frac{\psi(t)}{\nu_1}; \\
\frac{t^{p+1}-1}{p(p+1)} - \frac{t-1}{p} &\leq \frac{\psi(t)}{\nu_1}; \\
\frac{t^{1-q}-1}{q(q-1)} + \frac{t-1}{q} &\leq \frac{\psi(t)}{\nu_1}; \\
\psi(t) &\leq \frac{1}{2\nu_1}\psi'(t)^2.
\end{aligned}$$

Our next lemma presents some lower bounds for $| \psi'(t) |$ that can be viewed as a characterization of the barrier behaviors of the functions $\psi'(t)$ and $\psi(t)$ as well.

Lemma 3.10 Suppose that $\psi(t) \in \Omega_1$. Then

$$| \psi'(t) | \geq \frac{\nu_1}{q}(t^{-q} - 1), \quad \forall\, t < 1,$$

$$| \psi'(t) | \geq \frac{\nu_1}{p}(t^{p} - 1), \quad \forall\, t > 1.$$

Our next lemma provides the means to bound the increase of $\psi(t)$ with respect to t.

Lemma 3.11 Suppose that $\psi(t) \in \Omega_1$. Then, for any $\vartheta > 1$, we have

$$\psi(\vartheta t) \leq \frac{\nu_2}{\nu_1}(\vartheta^{p+1}\psi(t) + \vartheta \Upsilon'_{p,q}(\vartheta)\sqrt{2\nu_1\psi(t)} + \nu_1 \Upsilon_{p,q}(\vartheta)).$$

An immediate consequence of the above lemma is the following.

Lemma 3.12 Suppose that $\psi(t) \in \Omega_1$ and $\vartheta > 1$. There exist two constants $\nu_3,\ \nu_4 > 0$ dependent on p and q such that for all $\vartheta \in [1,\ 1+\nu_3]$, we have

$$\psi(\vartheta t) \leq \frac{\nu_2\nu_4}{\nu_1}(\psi(t) + (\vartheta - 1)\sqrt{2\nu_1\psi(t)} + \nu_1(\vartheta - 1)^2).$$

3.4 Self-Regular Functions in R^n_{++}

We defined univariate Self-Regular functions and discussed their properties in the previous sections. In this section, we extend our discussion to the multi-dimension space, we start with the definition of Self-Regular functions on R^n_{++}. Then we recall some results from [32] without proof.

Definition 3.3 A function $\psi(x) :\ R^n_{++} \to R^n_{++}$ defined by

$$\psi(x) = (\psi(x_1), ..., \psi(x_n))^T \tag{3.9}$$

is said to be Self-Regular if the kernel function $\psi(t) :\ R_{++} \to R$ is Self-Regular.

Definition 3.4 We also denote by $\Psi(x)$ the function

$$\Psi(x) := \sum_{i=1}^{n} \psi(x_i). \tag{3.10}$$

From the above definitions, one can easily see that Self-Regular functions in R^n_{++} also have the following attractive features.

Lemma 3.13 Let the functions $\psi(x)$ and $\Psi(x)$ be defined by (3.9) and (3.10), 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) $\Psi(x)$ is strictly convex with respect to $x \in R^n_{++}$, and vanishes at its global minimal point the all 1 vector $x = e$, i.e., $\Psi(e) = 0$ and $\psi(e) = \psi'(e) = 0$. Further[3],

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

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

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

For ease of reference, for any vector function $x(t) : R \to R^n_{++}$ we denote by $x'(t), x''(t)$ the first and second derivatives of $x(t)$ with respect to t, i.e.,

$$x'(t) = (x_1'(t), ..., x_n'(t))^T; \quad x''(t) = (x_1''(t), ..., x_n''(t))^T.$$

[3] x^p denotes the coordinatewise power of the vector x.

Lemma 3.14 Suppose that $x(t) : R \rightarrow R^n_{++}$ is a vector function. If $x(t)$ is twice differentiable with respect to t for all $t \in (l_t,\ u_t)$ and $\psi(t)$ is also a twice continuously differentiable function in a suitable domain that contains all the components of $x(t)$, then

$$\frac{d}{dt}\Psi(x(t)) = \psi'(x(t))^T x'(t), \quad t \in (l_t,\ u_t),$$

$$\frac{d^2}{dt^2}\Psi(x(t)) \leq \varpi \parallel x'(t) \parallel^2 + \psi'(x(t))^T x''(t),$$

where

$$\varpi = \max\{|\psi''(x_i(t))|, i = 1, 2, ..., n\}$$

is a number depending on $x(t)$ and $\psi(t)$.

Lemma 3.14 can be easily proved by simple calculus. Based on this lemma, one can estimate the first and second differentials of $\Psi(x(t))$ with respect to t without much difficulty.

Having discussed Self-Regular functions, now we can move to the main theme, the new proximity based on Self-Regular functions. The Self-Regular proximity suggested in [32] is defined as:

$$\Psi(x,\ s,\ \mu) := \Psi(v) = \sum_{i=1}^{n} \psi(v_i), \tag{3.11}$$

where v is defined by (2.3) and $\psi(\cdot)$ is a univariate Self-Regular function.

If the above proximity $\Psi(x,\ s,\ \mu)$ is *Self-Regular*, then the function $\psi(\cdot)$ is differentiable. Therefore, by simple differential, we can get

$$\nabla\Psi(v) = \left(\frac{d\psi(v_1)}{dv_1}, \frac{d\psi(v_2)}{dv_2}, \ldots, \frac{d\psi(v_n)}{dv_n}\right)^T.$$

Similar to (3.11), we can define the functions $\Psi(x(\mu)^{-1}x)$ and $\Psi(s(\mu)^{-1}s)$ as proximities in the primal and dual spaces. Based on the above definitions and results, we have the following Lemma 3.15, which provides us a means to bound the proximity measure in the scaled v-space according to the values of its counterparts in the primal x and dual s spaces.

Lemma 3.15 Let the proximity $\Psi(v)$ be defined by (3.11). If it is Self-Regular, then

$$\Psi(v) \leq \frac{1}{2}(\Psi(x(\mu)^{-1}x) + \Psi(s(\mu)^{-1}s)).$$

3.5 New IPMs Based on Self-Regular Proximities

As we mentioned in Chapter 2, we need to solve a Newton system to get the search direction in each iteration. Based on Self-Regular proximities, we have the following

system, which is a slight modification of the system (3.3):

$$\begin{aligned} \bar{A}d_x &= 0, \\ \bar{A}^T\Delta y + d_s &= 0, \\ d_x + d_s &= -\nabla\Psi(v). \end{aligned} \tag{3.12}$$

Here d_x, Δy, d_s denote the new search directions for LO problem, $-\nabla\Psi(v)$ represents the projected steepest descent direction w.r.t. $\Psi(v)$ in the v-space. Equivalently, in the original space system (3.12) can be written as

$$\begin{aligned} A\Delta x &= 0, \\ A^T\Delta y + \Delta s &= 0, \\ s\Delta x + x\Delta s &= -\mu v\nabla\Psi(v). \end{aligned} \tag{3.13}$$

As we know, if $\text{rank}(A) = m$, then for any $\mu > 0$, system (3.13) has a unique solution Δx, Δy, Δs. The result of a damped Newton step with damping factor α is given by

$$\begin{aligned} x &:= x + \alpha\Delta x, \\ s &:= s + \alpha\Delta s, \\ y &:= y + \alpha\Delta y. \end{aligned} \tag{3.14}$$

So far, we have introduced Self-Regular proximities, and built the Self-Regular based Newton system as well. By solving system (3.13), we can get the new Newton direction. Moreover, by using formula (3.14), we can obtain the new iterate. Based on these results, for a given Self-Regular proximity we can describe the new primal-dual algorithm as follows:

The Primal-Dual Algorithm Based on Self-Regular Proximities for LO

Input:

A proximity parameter $\tau > \nu_1^{-1}$;

an accuracy parameter $\epsilon > 0$;

a fixed update parameter θ, $0 < \theta < 1$;

(x^0, s^0), y^0, and $\mu^0 = 1$ such that $\Psi(x^0s^0, \mu^0) \leq \tau$.

begin

$x := x^0$; $s := s^0$; $y := y^0$; $\mu := \mu^0$;

while $n\mu \geq \epsilon$ **do**

begin

$\mu := (1-\theta)\mu$;

while $\Psi(xs, \mu) \geq \tau$ **do**

begin

Solve system (3.13) for $\Delta x,\ \Delta s,\ \Delta y$;

Determine step size α by some rules;

$x := x + \alpha\Delta x$;

$s := s + \alpha\Delta s$;

$y := y + \alpha\Delta y$;

end

end

end

In the algorithm, we use a value $\tau > \nu_1^{-1}$ for the proximity and we assume that we are given a triple $(x^0,\ s^0,\ y^0)$ such that $\Psi(x^0 s^0,\ \mu^0) \leq \tau$ for $\mu^0 = 1$. This can be done without loss of generality [34] by applying the Self-Dual embedding model.

For the current iterates (x, s, y) and centering parameter value μ, if the proximity $\Psi(xs,\ \mu)$ exceeds τ, then we use one or more damped Newton steps to re-center; otherwise μ is reduced by the factor $1-\theta$, where $0 < \theta < 1$. This is repeated until $n\mu < \epsilon$, where ϵ is the accuracy parameter, e.g., $\epsilon = 10^{-8}$.

With large-update, the algorithm has a polynomial complexity with $O(n^{\frac{q+1}{2q}} \log \frac{n}{\epsilon})$ iteration bound, where $q \geq 1$. When q increases, the iteration bound decreases and with $q = \log n$ it reaches the to date best bound $O(\sqrt{n} \log n \log \frac{n}{\epsilon})$. For the small-update method, the algorithm enjoys a polynomial complexity with $O(\sqrt{n} \log \frac{n}{\epsilon})$ iteration bound [32].

Chapter 4

Implementation of the New IPMs

In the previous chapters, we have discussed the fundamentals of path-following interior point methods, the homogeneous self-dual embedding model, the basic properties of self-regular proximities and a new family of interior point algorithms. The aim of this chapter is to convert the new algorithms into an efficient implementation. To reach this goal several problems have to be dealt with. Some of these problems have been at least partially discussed earlier (e.g., embedding model, etc.), but these need further elaboration. Some other topics (e.g., preprocessing, numerical linear algebra, etc.) have not yet been touched.

In this chapter, we first discuss the Newton system of the homogeneous self-dual embedding model for the standard LO problem, and compute the Newton direction by using a simplified method. Then we extend these to the Newton system of the homogeneous self-dual embedding model with variable upper bounds. Preprocessing, efficient handling of dense columns and Predictor-Corrector methods are discussed as well. A concise description of our implementation illustrated by some flowcharts of the program is presented in the last part of this chapter.

4.1 Solving the Newton System of (HLO) for the Standard LO Problem

We have introduced the homogeneous self-dual embedding model (HLO) in Section 2.6. In this section, we discuss how to solve the Newton System of (HLO) for the standard form LO problem. Let us first consider the standard LO problem:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0, \end{aligned} \tag{4.1}$$

where $A \in R^{m \times n},\ b \in R^m,\ c \in R^n$ and $x \in R^n$.

Recall the (HLO) system (2.9) in Section 2.6 (see also [34]), and let us assume that $(y^0,\ x^0,\ \tau^0,\ \nu^0,\ s^0,\ \kappa^0)$ is the given initial point with $x^0 > 0,\ \tau^0 > 0,\ \nu^0 > 0,\ s^0 > 0$ and

$\kappa^0 > 0$, then we have:

$$
\begin{array}{lll}
(\textbf{HLO}) & \min & \beta\nu \\
& \text{s.t.} & Ax - b\tau - r_p\nu = 0, \\
& & -A^Ty + c\tau - r_d\nu - s = 0, \\
& & b^Ty - c^Tx - r_g\nu - \kappa = 0, \\
& & r_p^Ty + r_d^Tx + r_g\tau = -\beta, \\
& & y \text{ is free}, \ x \geq 0, \ \tau \geq 0, \nu \in R, \ s \geq 0, \ \kappa \geq 0,
\end{array}
\tag{4.2}
$$

where

$$
\begin{aligned}
r_p &= \frac{Ax^0 - b\tau^0}{\nu^0}, \\
r_d &= \frac{c\tau^0 - A^Ty^0 - s^0}{\nu^0}, \\
r_g &= \frac{b^Ty^0 - c^Tx^0 - \kappa}{\nu^0}, \\
\beta &= x^{0T}s^0 + \tau^0\kappa^0.
\end{aligned}
\tag{4.3}
$$

Suppose that we have the current iterate and we want to compute the Newton direction (Δy, Δx, $\Delta\tau$, $\Delta\nu$, Δs, $\Delta\kappa$). The Newton system of the Homogeneous Self-dual embedding model is the following:

$$
\begin{array}{rrrrrr}
A\Delta x & - b\Delta\tau & - r_p\Delta\nu & & & = 0, \\
-A^T\Delta y & + c\Delta\tau & - r_d\Delta\nu & - \Delta s & & = 0, \\
b^T\Delta y - c^T\Delta x & & - r_g\Delta\nu & & - \Delta\kappa & = 0, \\
r_p^T\Delta y + r_d^T\Delta x & + r_g\Delta\tau & & & & = 0, \\
S\Delta x & & & + X\Delta s & & = r_{xs}, \\
& \kappa\Delta\tau & & & + \tau\Delta\kappa & = r_{\tau\kappa},
\end{array}
\tag{4.4}
$$

where[1] $X = \text{diag}(x)$, $S = \text{diag}(s)$ and

$$
\begin{aligned}
&\mu_+ = (1-\theta)\mu, \\
&r_{xs} = -\mu_+ \cdot v_{xs} \cdot \nabla\Psi(v_{xs}), \\
&r_{\tau\kappa} = -\mu_+ \cdot v_{\tau\kappa} \cdot \psi'(v_{\tau\kappa}), \\
&v_{xs} = \sqrt{\frac{xs}{\mu_+}}, \\
&v_{\tau\kappa} = \sqrt{\frac{\tau\kappa}{\mu_+}}, \\
&r_{xs} \in R^n, \ r_{\tau\kappa} \in R, \ v_{xs} \in R^n, \text{ and } v_{\tau\kappa} \in R.
\end{aligned}
$$

[1]Here $v, \Psi(v), \psi(v), \nabla\Psi(v), \nabla\psi(v)$ and $\psi'(v)$ are defined in Chapter 3 .

We can write (4.4) in matrix form as follows:

$$\begin{pmatrix} 0 & A & -b & -r_p & 0 & 0 \\ -A^T & 0 & c & -r_d & -I & 0 \\ b^T & -c^T & 0 & -r_g & 0 & -I \\ {r_p}^T & {r_d}^T & {r_g}^T & 0 & 0 & 0 \\ 0 & S & 0 & 0 & X & 0 \\ 0 & 0 & \kappa & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta x \\ \Delta\tau \\ \Delta\nu \\ \Delta s \\ \Delta\kappa \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ r_{xs} \\ r_{\tau\kappa} \end{pmatrix}. \tag{4.5}$$

By multiplying the first, second, third and last equality in (4.2) by y^T, x^T, τ and ν, respectively, and summing them up, we get:

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

When we apply this equation to our new point $(y + \Delta y,\ x + \Delta x,\ \tau + \Delta\tau,\ \nu + \Delta\nu,\ s + \Delta s,\ \kappa + \Delta\kappa)$, then we have:

$$\beta(\nu + \Delta\nu) = (x + \Delta x)^T(s + \Delta s) + (\tau + \Delta\tau)(\kappa + \Delta\kappa).$$

Using[2] $\Delta x^T \Delta s + \Delta\tau\Delta\kappa = 0$, one can easily get

$$\beta\Delta\nu = (x^T\Delta s + s^T\Delta x) + (\tau\Delta\kappa + \kappa\Delta\tau).$$

Combining this with (4.4), we have

$$\Delta\nu = \frac{-\mu_+}{\beta}(v_{xs}^T\nabla\Psi(v_{xs}) + v_{\tau\kappa}\psi'(v_{\tau\kappa})). \tag{4.6}$$

From the fifth and the last row of (4.5), we have

$$\Delta s = X^{-1}(r_{xs} - S\Delta x), \tag{4.7}$$

and

$$\Delta\kappa = \tau^{-1}(r_{\tau\kappa} - \kappa\Delta\tau). \tag{4.8}$$

Considering (4.6), (4.7), and (4.8), system (4.5) simplifies into the following system,

$$\begin{pmatrix} 0 & A & -b \\ -A^T & X^{-1}S & c \\ b^T & -c^T & \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta x \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} \bar{r}_p \\ \bar{r}_d \\ \bar{r}_g \end{pmatrix}, \tag{4.9}$$

where

$$\begin{aligned} \bar{r}_p &= r_p\Delta\nu, \\ \bar{r}_d &= r_d\Delta\nu + X^{-1}r_{xs}, \\ \bar{r}_g &= r_g\Delta\nu + \tau^{-1}r_{\tau\kappa}. \end{aligned}$$

[2]This can be obtained by the following steps: multiply the first four equations of (4.5) by Δy^T, Δx^T, $\Delta\tau$ and $\Delta\nu$, respectively, then sum the equations up.

Let us introduce the notation

$$D^2 = (X^{-1}S)^{-1},$$

then from the second row of (4.9) we get

$$\Delta x = D^2\bar{r}_d + D^2A^T\Delta y - D^2c\Delta\tau. \tag{4.10}$$

Now (4.9) can be simplified into the system,

$$\begin{pmatrix} AD^2A^T & a \\ \hat{a}^T & \gamma \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} \hat{r}_p \\ \hat{r}_g \end{pmatrix}, \tag{4.11}$$

where

$$\begin{aligned} a &= -b - AD^2c, \\ \hat{a} &= (b^T - c^TD^2A^T)^T, \\ \gamma &= \frac{\kappa}{\tau} + c^TD^2c, \\ \hat{r}_p &= \bar{r}_p - AD^2\bar{r}_d, \\ \hat{r}_g &= \bar{r}_g + c^TD^2\bar{r}_d. \end{aligned}$$

From (4.11), we get

$$\Delta\tau = \frac{1}{\gamma}(\hat{r}_g - \hat{a}^T\Delta y), \tag{4.12}$$

and

$$(AD^2A^T - \tilde{a}\hat{a}^T)\Delta y = \tilde{r}, \tag{4.13}$$

where

$$\begin{aligned} \tilde{a} &= \frac{a}{\gamma}, \\ \tilde{r} &= \hat{r}_p - \hat{r}_g\tilde{a}. \end{aligned}$$

If we solve (4.13) straightly, we must realize that the matrix $(AD^2A^T - \bar{a}_1a_2^T)$ is not a Positive-Definite Symmetric matrix, and due to the rank-one factor $\tilde{a}\hat{a}^T$, it is typically a fully dense matrix, thus we can not use sparse Cholesky factorization to it.

In order to solve system (4.13) efficiently, we can use the following steps:

First, we solve the following simpler systems:

$$\begin{aligned} AD^2A^T\eta &= \tilde{r} \\ AD^2A^T\tilde{\eta} &= -\tilde{a}. \end{aligned} \tag{4.14}$$

Having η and $\tilde{\eta}$, we can use the formula

$$\Delta y = \frac{\eta + \eta\hat{a}^T\tilde{\eta} - \hat{a}^T\eta\tilde{\eta}}{1 + \hat{a}^T\tilde{\eta}} = \eta - \tilde{\eta}\cdot\frac{\hat{a}^T\eta}{1 + \hat{a}^T\tilde{\eta}} \tag{4.15}$$

to compute Δy.

To solve (4.14) and (4.15), we just need one Cholesky factorization of the matrix AD^2A^T and four back substitutions. These steps are derived from the Sherman-Morrison-Woodbury formula [3].

Having Δy and $\Delta\nu$, one can calculate $\Delta\tau$, Δx, $\Delta\kappa$ and Δs easily by (4.15), (4.12), (4.10), (4.8) and (4.7).

Remark: The reader can easily verify that the size of the system (4.1) is $(m \times n)$; the size of the system (4.2) is $(m + n + 2) \times (m + 2n + 3)$; the size of the system (4.5) is $(m + 2n + 3) \times (m + 2n + 3)$; and finally the size of the system (4.14) is $m \times m$. The total cost to get the Newton steps for (HLO) include forming the matrix AD^2A^T with the cost of $O(mn^2)$ arithmetic operations, one Cholesky factorization of AD^2A^T with the cost of $O(m^3)$ arithmetic operations, and four back substitutions with the cost of $O(mn)$ arithmetic operations. On the other hand, if we solve the Newton system for the original LO problem, we need to solve the system $AD^2A^T\Delta y = \hat{\xi}$, the size of the coefficient matrix is $m \times m$, the total cost to get the Newton steps for the original LO problem include forming the matrix AD^2A^T with the cost of $O(mn^2)$ arithmetic operations, one Cholesky factorization of AD^2A^T with the cost of $O(m^3)$, and two back substitutions with the cost of $O(mn)$ arithmetic operations. From this point of view, we can solve the Newton system of the homogeneous self-dual embedding model with nearly the same cost as solving the Newton system of the original LO problem, the extra cost is two back substitutions.

4.2 Solving the (HLO) Newton System with Variable Upper Bounds

In the previous section, we have discussed how to solve the Newton system of (HLO) for standard LO problems, where all the variables satisfy $x \geq 0$, and they have no upper bounds. However, in many practical LO problems, some variables have upper bounds. These problems have the following form:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & 0 \leq x_i \leq u_i,\ i \in \mathcal{I} \\ & 0 \leq x_j,\ \ j \in \mathcal{J}, \end{array} \tag{4.16}$$

where

$$\begin{array}{l} A \in R^{m\times n},\ \text{rank}(A) = m, \\ |\mathcal{I}| = m_f,\ u \in R^{m_f},\ m_f \leq n, \\ \mathcal{I} \bigcup \mathcal{J} = \{1, 2, ..., n\},\ \mathcal{I} \bigcap \mathcal{J} = \emptyset. \end{array}$$

The last two constraints of (4.16) can be written as

$$\begin{aligned} &Fx + z = u, \\ &x \geq 0, \\ &z \geq 0, \end{aligned}$$

where $F \in R^{m_f \times n}$ and the rows of F are unit vectors corresponding to the upper bounded variables and $z \in R^{m_f}$ is a slack vector.

Suppose that we start with an initial solution (y^0, $w^0 > 0$, $x^0 > 0$, $\tau^0 = 1, \nu^0 = 1$, $z^0 > 0$, $s^0 > 0$, $\kappa^0 > 0$), then we can construct the Homogeneous Self-Dual Embedding model for the LO problem (4.16) with upper bounds as follows:

$$\begin{array}{lrrrrrrr} \min & & & & \beta\nu & & & \\ \text{s.t.} & & Ax & -b\tau & -r_{p1}\nu & & & = 0, \\ & & -Fx & +u\tau & -r_{p2}\nu & -z & & = 0, \\ & -A^T y & +F^T w & +c\tau & -r_d\nu & & -s & = 0, \\ & b^T y & -u^T w & -c^T x & -r_g\nu & & -\kappa & = 0, \\ & r_{p1}^T y & +r_{p2}^T w & +r_d^T x & +r_g\tau & & & = -\beta, \\ & y \in R^m, & w \geq 0, & x \geq 0, & \tau \geq 0, & \nu \in R, & z \geq 0, & s \geq 0, \quad \kappa \geq 0, \end{array} \tag{4.17}$$

where

$$\begin{aligned} r_{p1} &= Ax^0 - b\tau^0, \\ r_{p2} &= -Fx^0 + u\tau^0 - z^0, \\ r_d &= F^T w^0 + c\tau^0 - A^T y^0 - s^0, \\ r_g &= -u^T w^0 + b^T y^0 - c^T x^0 - \kappa^0, \\ \beta &= x^{0T} s^0 + z^{0T} w^0 + \tau^0 \kappa^0. \end{aligned}$$

We can derive the Newton system of problem (4.17) the same way as we did in Section 4.1 for problem (4.2). The Newton system can be written in following matrix form:

$$\begin{pmatrix} 0 & 0 & A & -b & -r_{p1} & 0 & 0 & 0 \\ 0 & 0 & -F & u & -r_{p2} & -I & 0 & 0 \\ -A^T & F^T & 0 & c & -r_d & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & -r_g & 0 & 0 & -I \\ r_{p1}^T & r_{p2}^T & r_d^T & r_g^T & 0 & 0 & 0 & 0 \\ 0 & Z & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & S & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta w \\ \Delta x \\ \Delta\tau \\ \Delta\nu \\ \Delta z \\ \Delta s \\ \Delta\kappa \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ r_{zw} \\ r_{xs} \\ r_{\tau\kappa} \end{pmatrix}, \tag{4.18}$$

where

$$\begin{aligned}
\mu_+ &= (1-\theta)\mu,\\
r_{zw} &= -\mu_+ \cdot v_{zw} \cdot \nabla\Psi(v_{zw}),\\
r_{xs} &= -\mu_+ \cdot v_{xs} \cdot \nabla\Psi(v_{xs}),\\
r_{\tau\kappa} &= -\mu_+ \cdot v_{\tau\kappa} \cdot \psi'(v_{\tau\kappa}),\\
v_{zw} &= \sqrt{\frac{zw}{\mu_+}},\\
v_{xs} &= \sqrt{\frac{xs}{\mu_+}},\\
v_{\tau\kappa} &= \sqrt{\frac{\tau\kappa}{\mu_+}}.
\end{aligned}$$

When we multiply the first, second, third, fourth and last equality of (4.17) by y^T, w^T, x^T, τ and ν, respectively, and summing the resulting equations up, we get:

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

When we apply this equation to our new point $(w+\Delta w,\, x+\Delta x,\, \tau+\Delta\tau,\, z+\Delta z,\, s+\Delta s,\, \kappa+\Delta\kappa)$, then we have:

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

Since[3] $\Delta x^T\Delta s + \Delta z^T\Delta w + \Delta\tau\Delta\kappa = 0$, one can easily get

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

Combining this with (4.18), we have

$$\Delta\nu = \frac{-\mu_+}{\beta}(v_{zw}^T\nabla\Psi(v_{zw}) + v_{xs}^T\nabla\Psi(v_{xs}) + v_{\tau\kappa}\psi'(v_{\tau\kappa})). \tag{4.19}$$

From the last three equations of (4.18), we have

$$\Delta z = W^{-1}(r_{zw} - Z\Delta w), \tag{4.20}$$

$$\Delta s = X^{-1}(r_{xs} - S\Delta x), \tag{4.21}$$

$$\Delta\kappa = \tau^{-1}(r_{\tau\kappa} - \kappa\Delta\tau). \tag{4.22}$$

Using (4.19), (4.20), (4.21) and (4.22), system (4.18) can be reduced to

$$\begin{pmatrix} 0 & 0 & A & -b\\ 0 & W^{-1}Z & -F & u\\ -A^T & F^T & X^{-1}S & c\\ b^T & -u^T & -c^T & \frac{\kappa}{\tau} \end{pmatrix} \begin{pmatrix} \Delta y\\ \Delta w\\ \Delta x\\ \Delta\tau \end{pmatrix} = \begin{pmatrix} \check{r}_{p1}\\ \check{r}_{p2}\\ \check{r}_d\\ \check{r}_g \end{pmatrix}, \tag{4.23}$$

[3]This can be obtained by the following steps: multiply the first five equations of (4.18) by Δy^T, Δw^T, Δx^T, $\Delta\tau$ and $\Delta\nu$, respectively, then sum the equations up.

where

$$\begin{aligned}
\check{r}_{p1} &= r_{p1}\Delta\nu,\\
\check{r}_{p2} &= r_{p2}\Delta\nu + W^{-1}r_{zw},\\
\check{r}_{d} &= r_{d}\Delta\nu + X^{-1}r_{xs},\\
\check{r}_{g} &= r_{g}\Delta\nu + \tau^{-1}r_{\tau\kappa}.
\end{aligned}$$

From the second equation of (4.23) we have

$$\Delta w = WZ^{-1}\check{r}_{p2} + WZ^{-1}F\Delta x - WZ^{-1}u\Delta\tau, \tag{4.24}$$

hence system (4.23) can be reduced to

$$\begin{pmatrix} 0 & A & -b \\ -A^T & X^{-1}S + F^TWZ^{-1}F & c - F^TWZ^{-1}u \\ b^T & -c^T - u^TWZ^{-1}F & \frac{\kappa}{\tau} + u^TWZ^{-1}u \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta x \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} \bar{r}_p \\ \bar{r}_d \\ \bar{r}_g \end{pmatrix}, \tag{4.25}$$

where

$$\begin{aligned}
\bar{r}_p &= \check{r}_{p1},\\
\bar{r}_d &= \check{r}_{d} - F^TWZ^{-1}\check{r}_{p2},\\
\bar{r}_g &= \check{r}_{g} + u^TWZ^{-1}\check{r}_{p2}.
\end{aligned}$$

For convenience, let us recall the notation D^2 defined by (4.10). Here we define D^2 as the diagonal matrix

$$D^2 = (X^{-1}S + F^TWZ^{-1}F)^{-1}.$$

From the second equation of (4.25), we get

$$\Delta x = D^2\bar{r}_d + D^2A^T\Delta y - D^2(c - F^TWZ^{-1}u)\Delta\tau. \tag{4.26}$$

Now system (4.25) can still be simplified to

$$\begin{pmatrix} AD^2A^T & a \\ \hat{a}^T & \gamma \end{pmatrix} \begin{pmatrix} \Delta y \\ \Delta\tau \end{pmatrix} = \begin{pmatrix} \hat{r}_p \\ \hat{r}_g \end{pmatrix}, \tag{4.27}$$

where

$$\begin{aligned}
a &= -b - AD^2(c - F^TWZ^{-1}u),\\
\hat{a} &= b - AD^2(c + F^TWZ^{-1}u),\\
\gamma &= (\frac{\kappa}{\tau} + u^TWZ^{-1}u) + (c + F^TWZ^{-1}u)^TD^2(c - F^TWZ^{-1}u),\\
\hat{r}_p &= \bar{r}_p - AD^2\bar{r}_d,\\
\hat{r}_g &= \bar{r}_g + (c + F^TWZ^{-1}u)^TD^2\bar{r}_d.
\end{aligned}$$

From the last equation of (4.27) we get

$$\Delta\tau = \frac{1}{\gamma}(\hat{r}_g - \hat{a}^T\Delta y). \tag{4.28}$$

Finally, system (4.27) can be reduced to

$$(AD^2A^T - \tilde{a}\hat{a}^T)\Delta y = \tilde{r}, \tag{4.29}$$

where

$$\begin{aligned}\tilde{a} &= \frac{a}{\gamma},\\ \tilde{r} &= \hat{r}_p - \tilde{a}\hat{r}_g.\end{aligned}$$

Because (4.29) has the same structure as (4.13), it can be solved the same way as (4.13):

First, we solve the following simpler system,

$$\begin{aligned}AD^2A^T\eta &= \tilde{r}\\ AD^2A^T\tilde{\eta} &= -\tilde{a}\end{aligned}$$

After we get η and $\tilde{\eta}$, we can use the

$$\Delta y = \frac{\eta + \eta\hat{a}^T\tilde{\eta} - \hat{a}^T\eta\tilde{\eta}}{1 + \hat{a}^T\tilde{\eta}} = \eta - \tilde{\eta}\cdot\frac{\hat{a}^T\eta}{1 + \hat{a}^T\tilde{\eta}} \tag{4.30}$$

formula to compute Δy.

Having Δy and $\Delta\nu$, one can calculate $\Delta\tau$, Δx, Δw, $\Delta\kappa$, Δs and Δz easily by (4.30), (4.28), (4.26), (4.24), (4.22), (4.21), (4.20) and (4.19).

Remark: It is easy to see that the size of system (4.16) is $(m + m_f) \times (n + m_f)$; the size of the system (4.17) is $(m + m_f + n + 2) \times (m + 2m_f + 2n + 3)$; the size of the system (4.18) is $(m + 2m_f + 2n + 3) \times (m + 2m_f + 2n + 3)$; and finally the size of the system (4.29) is $m \times m$. The total cost to get the Newton steps for (HLO) model include forming the matrix AD^2A^T with the cost of $O(mn^2)$, one Cholesky factorization of AD^2A^T with the cost of $O(m^3)$ and four back substitutions with the cost of $O(mn)$ arithmetic operations. On the other hand, if we solve the Newton system for the original LO problem, we need to solve the system $AD^2A^T\Delta y = \hat{\xi}$, the size of the coefficient matrix is $m \times m$, the total cost to get the Newton steps for the original LO problem include forming the matrix AD^2A^T with the cost of $O(mn^2)$, one Cholesky factorization of AD^2A^T with the cost of $O(m^3)$, and two back substitutions with the cost of $O(mn)$ arithmetic operations. From this point of view, we can conclude that to solve the Newton system of the homogeneous self-dual embedding model requires nearly the same cost as solving the Newton system of the original LO problem, the extra cost is two back substitutions.

4.3 Preprocessing after OSL

In our implementation, we use OSL to preprocess the input LO model. After OSL's preprocessing, we get the following general form LO model:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & A^1 x \geq b^1 \\ & A^2 x = b^2 \\ & A^3 x \leq b^3 \\ & \underline{b}^4 \leq A^4 x \leq \overline{b}^4 \\ & \ell \leq x \leq u, \end{array} \tag{4.31}$$

where the lower bounds and upper bounds can be infinite.

System (4.31) allows the constraints to be equality and inequality, the variables may have lower bounds and upper bounds. In the previous two sections, we discussed how to solve problems (4.1) and (4.16). Because problem (4.31) is in a different form, in order to solve problem (4.31), we have to do some modifications and transformations. Now we deal with the problem how to convert problem (4.31) into the form of problems (4.1) or (4.16).

First, we focus on the constraints. We convert all the inequalities into equalities, i.e., the constraints become $Ax = b$. To reach this goal, we need to do the following three transformations:

(1) $a_i^T x \leq b_i \Longrightarrow a_i^T x + x_{n+i} = b_i$, where $x_{n+i} \geq 0$

$$\Longrightarrow \hat{a}_i^T \hat{x} = b_i, \quad \text{where } \hat{a}_i = \begin{pmatrix} a_i \\ 1 \end{pmatrix} \text{ and } \hat{x} = \begin{pmatrix} x \\ x_{n+i} \end{pmatrix};$$

(2) $a_j^T x \geq b_i \Longrightarrow a_j^T x - x_{n+j} = b_j$, where $x_{n+j} \geq 0$

$$\Longrightarrow \hat{a}_j^T \hat{x} = b_j, \quad \text{where } \hat{a}_j = \begin{pmatrix} a_j \\ -1 \end{pmatrix} \text{ and } \hat{x} = \begin{pmatrix} x \\ x_{n+j} \end{pmatrix};$$

(3) $b'_k \leq a_k^T x \leq b''_k \Longrightarrow a_k^T x \leq b''_k$ and $a_k^T x \geq b'_k$

$$\begin{aligned} &\Longrightarrow a_k^T x + \xi = b''_k, a_k^T x - \bar{\xi} = b'_k, \quad \text{where } \xi \geq 0,\ \bar{\xi} \geq 0 \\ &\Longrightarrow a_k^T x + \xi = b''_k,\ \xi + \bar{\xi} = b''_k - b'_k,\ \xi \geq 0,\ \bar{\xi} \geq 0 \\ &\Longrightarrow a_k^T x + \xi = b''_k,\ 0 \leq \xi \leq b''_k - b'_k \\ &\Longrightarrow \hat{a}_k^T \hat{x} = b_k, \quad \text{where } \hat{a}_k = \begin{pmatrix} a_k \\ 1 \end{pmatrix},\ \hat{x} = \begin{pmatrix} x \\ \xi \end{pmatrix} \text{ and } 0 \leq \xi \leq b''_k - b'_k. \end{aligned}$$

So far, we converted all inequalities into equalities, thus we have the constraints in

the standard form $Ax = b$, i.e., we have the following system:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & \ell \leq x \leq u. \end{array} \tag{4.32}$$

Let's denote

$$\begin{aligned} \bar{x} &= x - \ell, \\ \bar{u} &= u - \ell, \\ \bar{b} &= b - A\ell, \end{aligned}$$

then (4.32) becomes

$$\begin{array}{ll} \min & c^T(\bar{x} + \ell) \\ \text{s.t.} & A\bar{x} = \bar{b} \\ & 0 \leq \bar{x} \leq \bar{u}. \end{array} \tag{4.33}$$

Since $c^T\ell$ is a constant, we get the following conclusions:

- If $\bar{u}_i = \infty, \forall\, i$, then (4.44) is equivalent to (4.1).
- Otherwise, (4.44) is in the form of (4.16).

4.4 Dense Columns

The most time consuming part in our algorithm for LO is to solve the linear systems as defined in (4.14):

$$\begin{aligned} AD^2A^T\eta &= \tilde{r} \\ AD^2A^T\tilde{\eta} &= -\tilde{a}. \end{aligned}$$

If A is sparse, we can use the sparse package WSMP[4] to solve it by doing sparse Cholesky factorization and back substitutions. However, if one or more columns of A are dense (see page 49), AD^2A^T will be dense. In this case, let us reconsider equation (4.29):

$$(AD^2A^T + \tilde{a}\hat{a}^T)\Delta y = \xi. \tag{4.34}$$

Let's denote:

$$AD^2A^T = \begin{pmatrix} a_1 & a_2 & \ldots & a_n \end{pmatrix} \begin{pmatrix} d_{11} & & & \\ & d_{22} & & \\ & & \ldots & \\ & & & d_{nn} \end{pmatrix} \begin{pmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_n^T \end{pmatrix}.$$

[4]WSMP: The Watson Sparse Matrix Package, which is a high-performance, robust, and easy to use software package for solving large sparse systems of linear equations using a direct method on IBM RS6000 workstations and IBM Scalable Parallel (SP) systems.

Without loss of generality, we may suppose that $A = [A_s\, A_d]$ where A_s is the sparse part of A, and D_s^2 is the corresponding part of D^2; A_d is the dense part of A, and D_d^2 is the corresponding part of D^2. Further we assume that there are only a few columns in A_d. Thus we have:

$$\begin{aligned} AD^2A^T + \tilde{a}\hat{a}^T &= \begin{pmatrix} A_s & A_d \end{pmatrix} \begin{pmatrix} D_s^2 & \\ & D_d^2 \end{pmatrix} \begin{pmatrix} A_s^T \\ \\ A_d^T \end{pmatrix} + \tilde{a}\hat{a}^T \\ &= A_s D_s^2 A_s^T + (A_d D_d^2 A_d^T + \tilde{a}\hat{a}^T) \\ &= A_s D_s^2 A_s^T + (UU^T + \tilde{a}\hat{a}^T) \\ &= P + RS^T, \end{aligned}$$

where A_d is $m \times k_1$, $P = A_s D_s^2 A_s^T$ is the sparse part of matrix AD^2A^T, $UU^T = A_d D_d^2 A_d^T$ is the dense part of matrix AD^2A^T, $R = [U\ \ \tilde{a}]$ and $S = [U\ \ \hat{a}]$. In the sequel we assume that R and S have low-rank.

By using the Sherman-Morrison-Woodbury formula [3], we have[5]:

$$(P + RS^T)^{-1} = P^{-1} - P^{-1}R(I + S^T P^{-1} R)^{-1} S^T P^{-1}.$$

Therefore, for (4.45), we have

$$\begin{aligned} \Delta y &= (P^{-1} - P^{-1}R(I + S^T P^{-1} R)^{-1} S^T P^{-1})\xi. \\ &= P^{-1}\xi - P^{-1}R(I + S^T P^{-1} R)^{-1} S^T P^{-1}\xi. \end{aligned}$$

To get Δy, we could compute the inverse matrix P^{-1}, but this calculation is expensive and numerically instable, thus we use the following more efficient method:

(1) First, we want to calculate $P^{-1}\xi$. In fact, this is equivalent to the solution of the system $Px_0 = \xi$. Since P is sparse, P and ξ are given, thus we can use WSMP to make a Cholesky factorization of P and solve the system $Px_0 = \xi$ efficiently, i.e., we get $x_0 = P^{-1}\xi$.

(2) Then, we need to calculate $P^{-1}R$. This is equivalent to the solution of the system $P\hat{X} = U$, here R has possibly more than one column, hence this system has multiple right-hand sides. Nevertheless we can still use WSMP to solve it inexpensively, and get $\hat{X} = P^{-1}U$. In this step we use the same Cholesky factorization of P that was already calculated in step (1).

(3) Having $\hat{X}$, we need to calculate $(I + S^T\hat{X})^{-1}S^T x_0$. As in step (1) and (2), this is equivalent to the solution of the system $(I + S^T\hat{X})y_0 = S^T x_0$. It is a dense system, but fortunately, it is a low dimensional system, its dimension is equal to $k_1 = \text{rank}(R)$, thus the computational cost is small.

[5]Here we assume that the sparse part P is nonsingular. We discuss how to deal with a singular P in the next section.

(4) Finally, we compute $\Delta y = x_0 - \hat{X} y_0$. Having x_0, $\hat{X}$, and y_0, this is very cheap.

The procedure, not counting vector-vector additions, includes:

- calculate P, R and S;
- one Cholesky factorization of the sparse matrix P;
- one back substitution for x_0;
- one block back substitution for k_1 columns to get $\hat{X}$;
- solve a low-rank dense system of size $k_1 \times k_1$;
- one matrix-vector multiplication for $\hat{X} y_0$.

So far we have discussed how to solve system (4.34) if matrix A has one or more dense columns. We still have to discuss how to identify dense columns. There are many criteria in the literature, the following is a widely used reasonable one [45]:

Suppose $A = (a_1, a_2, ..., a_n)$ is given, let us denote by m the *number of rows* in A, nz_i is the *number of nonzero elements in column* a_i, and nr is the criterion value of *nonzero ratio*[6], then:

- if $\frac{nz_i}{m} > nr$ then a_i is dense;
- if $\frac{nz_i}{m} \leq nr$ then a_i is sparse.

Here nr should depend on m, that is, the bigger m is, the smaller nr should be. We make the following choices:

the number of rows m	nonzero ratio nr
$500 < m < 1000$	0.2
$1000 \leq m \leq 5000$	0.1
$5000 < m$	0.01

Remark: We assumed in this section that the sparse part of AD^2A^T is nonsingular. If the sparse part is singular, we can not apply the above procedure directly and we need to do some extra work on the matrix before we can apply Cholesky decomposition. The next section is devoted to this case.

[6] *nonzero ratio* $= \frac{\text{number of nonzero elements}}{\text{number of elements}}$.

4.5 The Solution of the Newton System When $A_s A_s^T$ is Singular

For convenience, we can rewrite equation (4.34) as follows [3]:

$$(\bar{A}_s \bar{A}_s^T + \bar{A}_d \bar{A}_d^T + \bar{a}\tilde{a}^T)\Delta y = \xi, \tag{4.35}$$

where $\bar{A}_s = A_s D_s$ is the sparse part of AD, i.e., all the columns in $\bar{A}_s$ are sparse; $\bar{A}_d = A_d D_d$ is the dense part of AD, i.e., all the columns in $\bar{A}_d$ are dense, and A_d is $m \times k_2$ with $k_2 << m$ [4]. First we have to check the singularity of A_s.

Check the singularity of matrix $\bar{A}_s \bar{A}_s^T$

Let us denote $\bar{a}_{ij}$ the element of matrix $\bar{A}_s$, where $1 \le i \le m$ and $1 \le j \le (n - k_2)$. We use the following algorithm (pseudo code) to check the singularity of matrix $\bar{A}_s \bar{A}_s^T$.

$k_3 = 0$
begin
  for $i = 1$ *to* m
    if $\bar{a}_{ii} \neq 0$ *then*
      do Gaussian elimination and have $\bar{a}_{ki} = 0,\ \forall\ i + 1 \le k \le m$
    else
      if $\bar{a}_{ik} = 0,\ \forall\ i \le k \le n - k_2$ *then*
        $\bar{A}_s \bar{A}_s^T$ *is singular*
        $k_3 = k_3 + 1;\ i_{k_3} = i;$
      else
        do pivoting on $\bar{A}_s$ *and get* $\bar{a}_{ii} \neq 0$
        do Gaussian elimination and have $\bar{a}_{ki} = 0,\ \forall\ i + 1 \le k \le m$
      endif
    endif
  endfor
end
output $\tilde{A}_s$

Do some modification on the matrix $\bar{A}_s$ if $\bar{A}_s \bar{A}_s^T$ is singular

Suppose $i_1,\ i_2,\ \cdots\ i_{k_3}$ are the row indices of the zero rows in matrix $\tilde{A}_s$. Since k_2 is the number of columns of A_d and $\text{rank}(A_s) = m - k_3$, we have $k_3 \le k_2$.

Let

$$H = (h_{i_1} \;\; h_{i_2} \;\; \cdots \;\; h_{i_{k_3}}),$$

where for all $1 \leq j \leq k_3$, $h_{i_j} = (0 \;\; \cdots \;\; 0 \;\; 1 \;\; 0 \;\; \cdots \;\; 0)^T$, i.e., h_{i_j} is a unit vector with its i_j^{th} element equal to 1, and all the other elements are 0.

Then we apply the following transformation to the system (4.35), and we get

$$[(\bar{A}_s \bar{A}_s^T + HH^T) + (-HH^T + \bar{A}_d \bar{A}_d^T + \bar{a}\tilde{a}^T)]\Delta y = \xi.$$

Let us denote

$$P := \hat{A}_s \hat{A}_s^{\;T} = \bar{A}_s \bar{A}_s^T + HH^T = [\bar{A}_s \;\; H][\bar{A}_s \;\; H]^T,$$

and let

$$R = [-H \;\; \bar{A}_d \;\; \bar{a}],$$
$$S = [H \;\; \bar{A}_d \;\; \tilde{a}],$$

then we have

$$R \cdot S^T = [-H \;\; \bar{A}_d \;\; \bar{a}][H \;\; \bar{A}_d \;\; \tilde{a}]^T = -HH^T + \bar{A}_d \bar{A}_d^T + \bar{a}\tilde{a}^T.$$

Now we have the following simplified form of the Newton system (4.35):

$$(P + R \cdot S^T)\Delta y = \xi, \tag{4.36}$$

where P is sparse and nonsingular; R and S have dimension $m \times (k_2 + k_3 + 1)$, thus RS^T is dense with rank at most $(k_2 + k_3 + 1)$, thus having low-rank.

Using the Sherman-Morrison-Woodbury formula to the above simplified Newton system, we have:

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

then

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

Analogous to the procedure explained in Section 4.4, we do not need to compute any inverse matrices. Instead, we use Cholesky factorization and back substitutions to the following four subproblems:

(1) solve the system $Px_0 = \xi$, and get the solution $x_0 = P^{-1}\xi$;

(2) solve the system $P\hat{X} = R$, and get the solution $\hat{X} = P^{-1}R$;

(3) solve the low-rank dense system $(I + S^T\hat{X})y_0 = S^T x_0$, and get the solution y_0;

(4) compute $\Delta y = x_0 - \hat{X}y_0$.

The computational cost of the above steps include:

- one Cholesky factorization of the sparse matrix P;
- one back substitution for x_0;
- one block back substitution with size $(k_2 + k_3 + 1)$ for $\hat{X}$;
- solve a low-rank dense system of size $(k_2 + k_3 + 1)$ by LU factorization.

Thus the total number of arithmetic operations is still $O(n^3)$.

4.6 The Predictor-Corrector Method

We have discussed in the previous sections how to get the search direction by solving the Newton system of the (HLO) model at each step. All of those methods are first order methods that use the Newton direction as the search direction to move to a new point. In this section we introduce the so-called *Predictor-corrector method*, which is a second order method. The motivation for developing higher order methods comes from the observation that the most expensive step of an iteration in an interior-point method is the factorization of a matrix that changes only numerically at each iteration[7]. The higher order method was proposed by Monterio, Adler and Resende [27]. Motivated by this work, Mehrotra [24] developed a practical second-order method which is called *Predictor-corrector method*. For this kind of method, we compute the Newton direction, then we calculate the second order items of the Newton system by using the first order direction. Then we can calculate the real search direction by using the same Cholesky factorization which was used to compute the Newton direction. As a result, the number of iterations is reduced, and the number of arithmetic operations is reduced as well.

The main procedure of *Predictor-corrector method* is that it computes the following three components at each iteration:

- the *predictor* direction is a primal-dual affine-scaling direction, which is the pure Newton direction for the optimality conditions by solving the Newton system (4.37);
- a centering parameter whose size is determined adaptively;
- the *corrector* direction that attempts to compensate the deviation from the central path in the affine-scaling direction.

Let us first consider the predictor-corrector method for the Newton system of (HLO) without variable upper bounds. Suppose that our current point is $(y, x, \tau, \nu, s, \kappa) > 0$. Recall system (4.5). The predictor (affine-scaling) direction can be obtained by choosing

[7]Fortunately, the nonzero pattern of the matrix remains unchanged, thus we need only once symbolic factorization of the matrix through the algorithm, and one Cholesky factorization at each step.

$r_{xs} = -XSe$, $r_{\tau\kappa} = -\tau\kappa$. Thus the following system:

$$\begin{pmatrix} 0 & A & -b & -r_p & 0 & 0 \\ -A^T & 0 & c & -r_d & -I & 0 \\ b^T & -c^T & 0 & -r_g & 0 & -I \\ {r_p}^T & {r_d}^T & {r_g}^T & 0 & 0 & 0 \\ 0 & S & 0 & 0 & X & 0 \\ 0 & 0 & \kappa & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y^a \\ \Delta x^a \\ \Delta\tau^a \\ \Delta\nu^a \\ \Delta s^a \\ \Delta\kappa^a \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ -XSe \\ -\tau\kappa \end{pmatrix} \tag{4.37}$$

is obtained. Having the predictor direction (Δy^a, Δx^a, $\Delta\tau^a$, $\Delta\nu^a$, Δs^a, $\Delta\kappa^a$), we need to find the maximal feasible step length to the boundary along this direction. This can be done by solving the following systems [24]:

$$\alpha_p = \arg\max_{\alpha}\{\alpha_i | x + \alpha\Delta x^a \geq 0,\, \tau + \alpha\Delta\tau^a \geq 0\}$$

and

$$\alpha_d = \arg\max_{\alpha}\{\alpha_i | s + \alpha\Delta s^a \geq 0,\, \kappa + \alpha\Delta\kappa^a \geq 0\}.$$

Moreover, we calculate

$$\mu_a = \frac{(x + \alpha_p\Delta x^a)^T(s + \alpha_d\Delta s^a) + (\tau + \alpha_p\Delta\tau^a)^T(\kappa + \alpha_d\Delta\kappa^a)}{n+1}$$

and

$$\sigma = \left(\frac{\mu_a}{\mu}\right)^3,$$

where σ determines the update of the centering parameter μ.

To move close to the central path, Mehrotra proposed a strategy which consists of two parts [24]:

(1) The centering component is obtained by solving a linear system with the same coefficient matrix as (4.37) and right-hand side (0, 0, 0, 0, $\sigma\mu e - XSe$, $\sigma\mu - \tau\kappa$);

(2) The corrector component is obtained by solving a linear system with the same coefficient matrix as (4.37) and right-hand side (0, 0, 0, 0, $-\Delta x^a\Delta s^a$, $-\Delta\tau^a\Delta\kappa^a$).

We can combine the calculation of these two parts into one step to obtain the *combined centering-corrector item* by solving the following linear system:

$$\begin{pmatrix} 0 & A & -b & -r_p & 0 & 0 \\ -A^T & 0 & c & -r_d & -I & 0 \\ b^T & -c^T & 0 & -r_g & 0 & -I \\ {r_p}^T & {r_d}^T & {r_g}^T & 0 & 0 & 0 \\ 0 & S & 0 & 0 & X & 0 \\ 0 & 0 & \kappa & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y^{cc} \\ \Delta x^{cc} \\ \Delta\tau^{cc} \\ \Delta\nu^{cc} \\ \Delta s^{cc} \\ \Delta\kappa^{cc} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ r^1 \\ r^2 \end{pmatrix}, \tag{4.38}$$

where

$$r^1 = \sigma\mu e - XSe - \Delta x^a \Delta s^a e, \tag{4.39}$$
$$r^2 = \sigma\mu - \tau\kappa - \Delta\tau^a \Delta s \kappa^a.$$

Then we get the search direction as

$$(\Delta y,\, \Delta x,\, \Delta\tau,\, \Delta\nu,\, \Delta s,\, \Delta\kappa) = (\Delta y^{cc},\, \Delta x^{cc},\, \Delta\tau^{cc},\, \Delta\nu^{cc},\, \Delta s^{cc},\, \Delta\kappa^{cc}).$$

Remark: We use Self-Regular proximities in our implementation, thus with $\mu^+ = \sigma\mu$, we can write (4.39) as follows:

$$\begin{aligned} r^1 &= -\mu^+ v_{xs} \nabla\Psi(v_{xs}) - \Delta x^a \Delta s^a e, \\ r^2 &= -\mu^+ v_{\tau\kappa} \nabla\psi(v_{\tau\kappa}) - \Delta\tau^a \Delta\kappa^a, \end{aligned}$$

where Ψ and ψ are defined in Section 3.2, v_{xs} and $v_{\tau\kappa}$ is defined in Section 4.1.

Now we move to the other case, i.e., the Newton system of (HLO) with variable upper bounds. Suppose our current point is $(y,\, w,\, x,\, \tau,\, \nu,\, z,\, s,\, \kappa) > 0$, recall the previous system (4.18), the predictor direction can be obtained by solving the following system:

$$\begin{pmatrix} 0 & 0 & A & -b & -r_{p1} & 0 & 0 & 0 \\ 0 & 0 & -F & u & -r_{p2} & -I & 0 & 0 \\ -A^T & F^T & 0 & c & -r_d & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & -r_g & 0 & 0 & -I \\ r_{p1}^T & r_{p2}^T & r_d^T & r_g^T & 0 & 0 & 0 & 0 \\ 0 & Z & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & S & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y^a \\ \Delta w^a \\ \Delta x^a \\ \Delta\tau^a \\ \Delta\nu^a \\ \Delta z^a \\ \Delta s^a \\ \Delta\kappa^a \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -ZWe \\ -XSe \\ -\tau\kappa \end{pmatrix}. \tag{4.40}$$

Then we find the step length by

$$\alpha_p = \arg\max_{\alpha} \{\alpha_i | x + \alpha\Delta x^a \geq 0,\, z + \alpha\Delta z^a \geq 0,\, \tau + \alpha\Delta\tau^a \geq 0\},$$

$$\alpha_d = \arg\max_{\alpha} \{\alpha_i | s + \alpha\Delta s^a \geq 0,\, w + \alpha\Delta w^a \geq 0,\, \kappa + \alpha\Delta\kappa^a \geq 0\}.$$

In addition, we calculate

$$\sigma = \left(\frac{\mu_a}{\mu}\right)^3,$$

where

$$\mu_a = \frac{(x + \alpha_p\Delta x^a)^T(s + \alpha_d\Delta s^a) + (z + \alpha_p\Delta z^a)^T(w + \alpha_d\Delta w^a)}{n + m_f + 1}$$
$$+ \frac{(\tau + \alpha_p\Delta\tau^a)^T(\kappa + \alpha_d\Delta\kappa^a)}{n + m_f + 1}.$$

Then, we obtain the *combined centering-corrector item* by solving the linear system:

$$\begin{pmatrix} 0 & 0 & A & -b & -r_{p1} & 0 & 0 & 0 \\ 0 & 0 & -F & u & -r_{p2} & -I & 0 & 0 \\ -A^T & F^T & 0 & c & -r_d & 0 & -I & 0 \\ b^T & -u^T & -c^T & 0 & -r_g & 0 & 0 & -I \\ r_{p1}^T & r_{p2}^T & r_d^T & r_g^T & 0 & 0 & 0 & 0 \\ 0 & Z & 0 & 0 & 0 & W & 0 & 0 \\ 0 & 0 & S & 0 & 0 & 0 & X & 0 \\ 0 & 0 & 0 & \kappa & 0 & 0 & 0 & \tau \end{pmatrix} \begin{pmatrix} \Delta y^{cc} \\ \Delta w^{cc} \\ \Delta x^{cc} \\ \Delta \tau^{cc} \\ \Delta \nu^{cc} \\ \Delta z^{cc} \\ \Delta s^{cc} \\ \Delta \kappa^{cc} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ r^1 \\ r^2 \\ r^3 \end{pmatrix}, \tag{4.41}$$

where

$$\begin{aligned} r^1 &= \sigma\mu e - ZWe - \Delta z^a \Delta w^a e, \\ r^2 &= \sigma\mu e - XSe - \Delta x^a \Delta s^a e, \\ r^3 &= \sigma\mu - \tau\kappa - \Delta\tau^a \Delta s \kappa^a. \end{aligned} \tag{4.42}$$

Finally, we get the search direction as

$$\begin{aligned} &(\Delta y,\, \Delta w,\, \Delta x,\, \Delta\tau,\, \Delta\nu,\, \Delta z,\, \Delta s,\, \Delta\kappa) \\ &= (\Delta y^{cc},\, \Delta w^{cc},\, \Delta x^{cc},\, \Delta\tau^{cc},\, \Delta\nu^{cc},\, \Delta z^{cc},\, \Delta s^{cc},\, \Delta\kappa^{cc}). \end{aligned}$$

Remark: We use Self-Regular proximities in our implementation, thus with $\mu^+ = \sigma\mu$ we write (4.42) as follows:

$$\begin{aligned} r^1 &= -\mu^+ v_{zw} \nabla\Psi(v_{zw}) - \Delta z^a \Delta w^a e, \\ r^2 &= -\mu^+ v_{xs} \nabla\Psi(v_{xs}) - \Delta x^a \Delta s^a e, \\ r^3 &= -\mu^+ v_{\tau\kappa} \nabla\psi(v_{\tau\kappa}) - \Delta\tau^a \Delta\kappa^a, \end{aligned}$$

where Ψ and ψ are defined in Section 3.2, v_{zw}, v_{xs} and $v_{\tau\kappa}$ are defined in Section 4.2.

One can observe that in predictor-corrector algorithms we need to solve two linear systems, either (4.37) and (4.38), or (4.40) and (4.41) at each iteration. However, (4.38) has the same coefficient matrix as (4.37), (4.41) has the same coefficient matrix as (4.40), thus we need only one Cholesky factorization at each step for both of the two cases. Moreover, if we combine the predictor-corrector method with the simplifying strategy which is described in Section 4.1 and Section 4.2, we can easily deduce that the total cost of solving the Newton system includes one Cholesky factorization and eight back substitutions at each step. From this point of view, we slightly increase the computational cost at each iteration, but the number of iterations is reduced as computational expense shows. Therefore, predictor-corrector algorithms are more efficient.

At end of this section, we present our Self-Regular based predictor-corrector IPM for LO as follows:

Self-Regular Based Predictor-Corrector IPM for LO

Input:

An accuracy parameter $\epsilon > 0$;

choose a Self-Regular function Ψ as proximity;

an initial point $(y^0, w^0, x^0, \tau^0, \nu^0, z^0, s^0, \kappa^0) > 0$.

begin

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

$\mu := (x^T s + z^T w + \tau\kappa)/(n + m_f + 1)$;

while $(n + m_f + 1)\mu \geq \epsilon$ **do**

begin

Predictor

Solve system (4.40) for the predictor direction Δy^a, Δw^a, Δx^a, $\Delta \tau^a$, $\Delta \nu^a$, Δz^a, Δs^a and $\Delta \kappa^a$;

Corrector

Determine the gap barrier parameter update factor σ;

Solve system (4.41) for the corrector direction Δy^{cc}, Δw^{cc}, Δx^{cc}, $\Delta \tau^{cc}$, $\Delta \nu^{cc}$, Δz^{cc}, Δs^{cc} and $\Delta \kappa^{cc}$;

Determine step size α;

Update

$y := y + \alpha \Delta y^{cc}$;

$w := w + \alpha \Delta w^{cc}$;

$x := x + \alpha \Delta z^{cc}$;

$\tau := \tau + \alpha \Delta \tau^{cc}$;

$\nu := \nu + \alpha \Delta \nu^{cc}$;

$z := z + \alpha \Delta z^{cc}$;

$s := s + \alpha \Delta s^{cc}$;

$\kappa := \kappa + \alpha \Delta \kappa^{cc}$;

$\mu := (x^T s + z^T w + \tau\kappa)/(n + m_f + 1)$;

end

end

4.7 Outline of the Implementation

So far we have discussed a class of Self-Regular based IPMs (Chapter 3) and a Self-Regular based predictor-corrector algorithm (Section 4.6). Our implementation is based on these two algorithms, respectively. The structure of the implementation is outlined by the following flowchart:

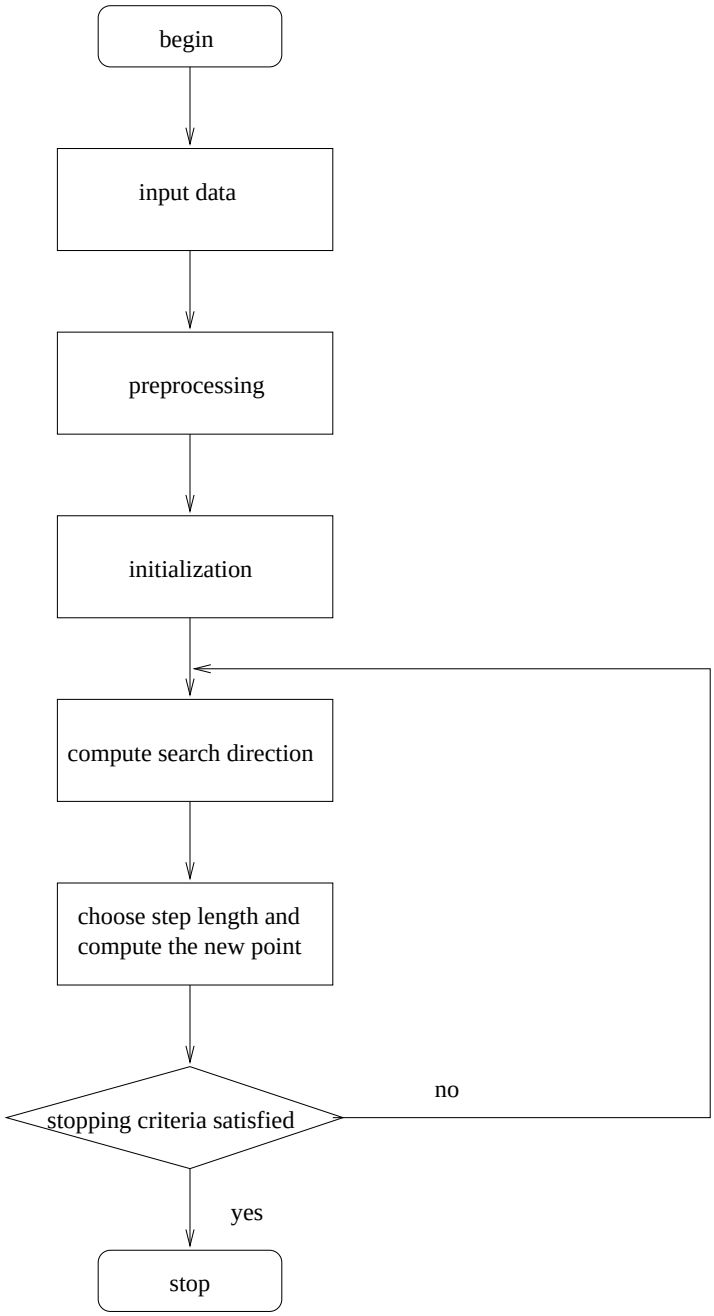

Figure 4.1 The flowchart of the implementation.

To make our implementation more efficient, we make use of some existing software packages, the packages, used in our implementation, are the following:

- We use OSL to perform preprocessing and postprocessing. In addition, we do some more preprocessing and postprocessing (see Section 4.3) by ourselves, since OSL's preprocessing is not sufficient for our purpose.

- We use ESSL to perform computations with matrices, vectors, and scalars.

- We use WSMP to perform Symbolic factorization, Cholesky factorization and back substitutions to solve the Newton systems.

Now we describe how our implementation works. All our explanations and descriptions are based on the flowchart presented in Figure 4.1.

- In the *input data* part, we read input data from an **mps** file which has standard fixed format to describe objective function, coefficient matrix, right-hand side, etc.;

- In the *preprocessing* part, we first use OSL to perform some work, then we use our additional procedures discussed in Section 4.3 to bring the problem in the standard upper bounded form;

- In the *initialization* part, we use the results in Section 2.4. We can choose our starting point as ($y^0 = 0$, $w^0 = e$, $x^0 = e$, $\tau^0 = 1, \nu^0 = 1$, $z^0 = e$, $s^0 = e$, $\kappa^0 = 1$);

- In the *compute the search direction* part, we use the strategy introduced in the previous sections to calculate the search direction at each step[8];

- In the *choose step length* part, we use the method presented in Section 2.4 to compute the step size at each iteration[9];

- In the *stopping criteria* part, we use the procedure proposed in Section 2.5 to decide if the iteration continues or the algorithm stops[10].

So far we have discussed the outline and fundamental structure of the implementation from the design point of view. At end of this section, let us move to the program side. There are twenty major program modules: *main*( ), *preprocessing*( ), *passdata*( ), *main_sr*( ), *main_sr01*( ), *selreg*( ), *selreg01*( ), *sparmv*( ), *sparmtv*( ), *sparmm*( ), *sparmmt*( ), *call_nonzeros*( ), *call_iaja*( ), *call_avals*( ), *symfac*( ), *chofac*( ), *bacsub*( ), *stepsize*( ), *stepsize01*( ) and *postprocessing*( ).

[8]See Section 4.1, 4.2 and 4.6 for more details.

[9]In our implementation, we need to consider the problem with upper bounds. The reader can easily get the analogous results for this case.

[10]We can easily get the analogous results for the problem with upper bounds.

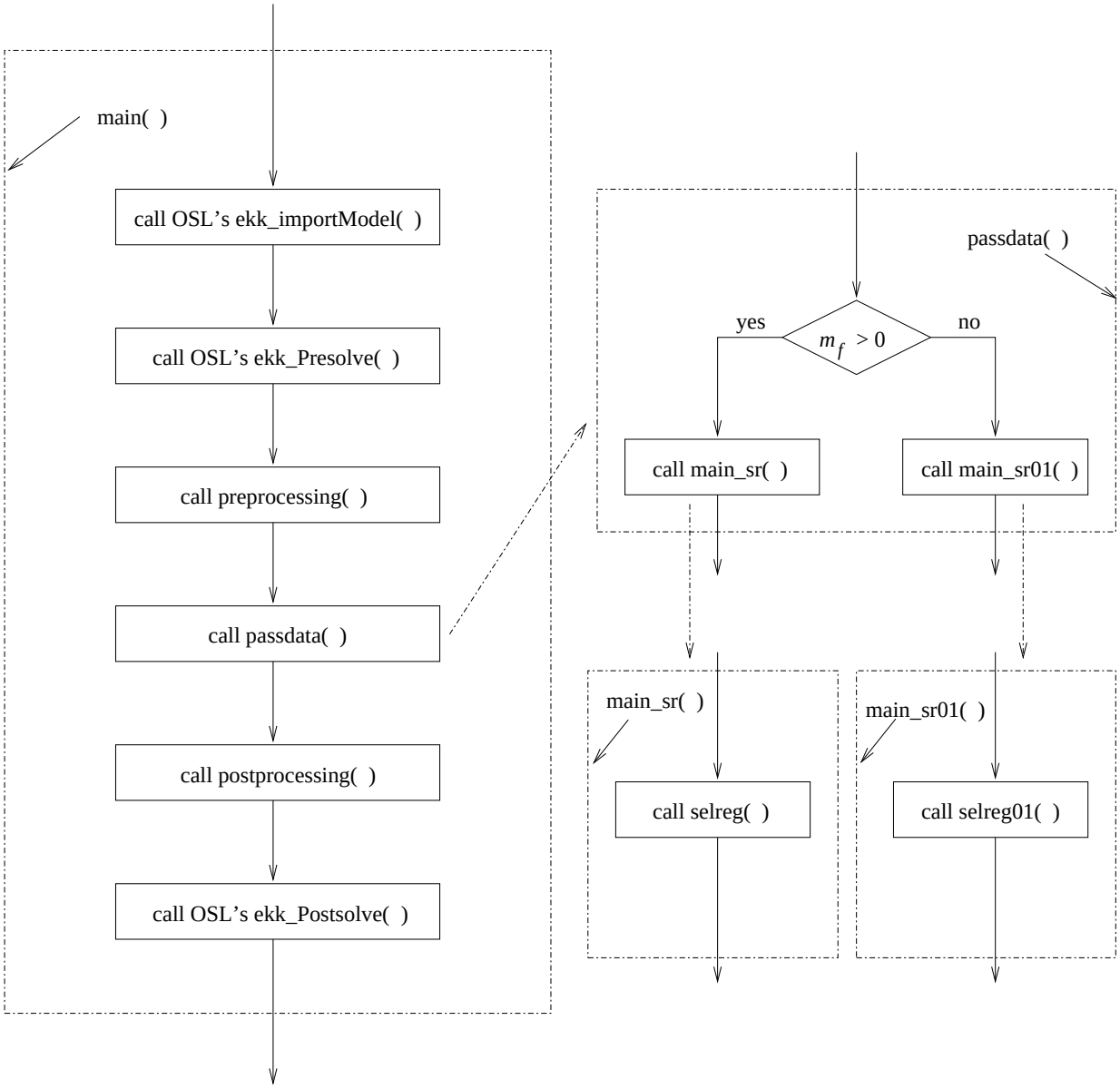

Figure 4.2 The flow chart of the subroutines.

The functionality of each subroutines is described below:

(1) *main*( ) subroutine. It manages the following things:

- Sets up the environment of the whole program;
- Calls OSL's ekk_importModel( ) to read the input data from an MPS file;
- Calls OSL's ekk_Presolve( ) to do preprocessing;
- Calls our own *preprocessing*( ) subroutine to do some extra preprocessing;
- Calls *passdata*( ) subroutine;
- Calls our own *postprocessing*( ) subroutine to do some extra postprocessing;
- Calls OSL's ekk_Postsolve( ) to do postprocessing;
- Stops the program.

(2) *preprocessing*( ) subroutine. Our own preprocessing subroutine after OSL's preprocessing. It performs the following tasks:

- Eliminates the fixed variables;
- Transforms the free variables to be upper bounded variables;
- Transforms the inequality constraints to be equality constraints;
- Transforms the lower bounds of the variables to be zero by shifting.

(3) *passdata*( ) subroutine. It receives data from the subroutine *main*( ), then performs the following tasks:

- Refinery of the data passed from *main*( ) subroutine;
- Computes the number of the upper bounded variables , and generates the corresponding matrix F;
- If $m_f > 0$ then calls the subroutine *main_sr*( ), otherwise calls the *main_sr01*( ) subroutine.

(4) *main_sr*( ) subroutine. An initialization subroutine in the system. It generates an initial point (y^0, w^0, x^0, τ^0, ν^0, z^0, s^0, κ^0) for the problem which has variable upper bounds, and passes the initial values to the subroutine *selreg*( ).

(5) *main_sr01*( ) subroutine. The other initialization subroutine to generate an initial point (y^0, x^0, τ^0, ν^0, s^0, κ^0) for the problem without variable upper bounds. It passes the initial values to the subroutine *selreg01*( ).

(6) *selreg*( ) subroutine. This subroutine solves the Newton system for the problem with variable upper bounds. The structure of this subroutine is illustrated in Figure 4.3, and the functionality of this subroutine is described as follows:

- Receives data from the subroutine *main_sr*( ).
- Calls *symfac*( ) subroutine to perform *ordering* and *symbolic factorization*. It calls the subroutine *symfac*( ) only once.
- Computes D^2 and AD^2A^T at each iteration.
- Calls *chofac*( ) subroutine to perform *Cholesky factorization* at each iteration.
- Calls *bacsub*( ) subroutine to get predictor direction and corrector direction respectively. It calls subroutine *bacsub*( ) twice at each iteration.
- Calls *stepsize*( ) subroutine to calculate step size for the predictor and the corrector direction, respectively. It calls subroutine *stepsize*( ) twice at each iteration.
- Calculates the new iterative point based on the search direction and the step size.

(7) *selreg01*( ) subroutine. This subroutine solves the Newton system for the problem without variable upper bounds. The structure of this subroutine is illustrated in Figure 4.3 as well. It performs the following main tasks:

- Receives data from the subroutine *main_sr01*( ).
- Calls *symfac*( ) subroutine to perform *ordering* and *symbolic factorization*. It calls the subroutine *symfac*( ) only once.
- Computes D^2 and AD^2A^T at each iteration.
- Calls *chofac*( ) subroutine to perform *Cholesky factorization* at each iteration.
- Calls *bacsub*( ) subroutine to get predictor direction and corrector direction respectively. It calls subroutine *bacsub*( ) twice at each iteration.
- Calls *stepsize01*( ) subroutine to calculate step size for the predictor and the corrector direction, respectively. It calls subroutine *stepsize01*( ) twice at each iteration.
- Calculates the new iterative point based on the search direction and the step size.

(8) *sparmv*( ) subroutine. It performs the multiplication of a sparse matrix by a vector.

(9) *sparmtv*( ) subroutine. This subroutine performs the multiplication of the transpose of a sparse matrix by a vector.

(10) *sparmm*( ) subroutine. It computes the multiplication of a sparse matrix by a sparse matrix.

(11) *sparmmt*( ) subroutine. It computes the multiplication of a sparse matrix by a sparse matrix transpose.

(12) *call_nonzeros*( ) subroutine. This subroutine computes the number of nonzero elements in matrix AA^T.

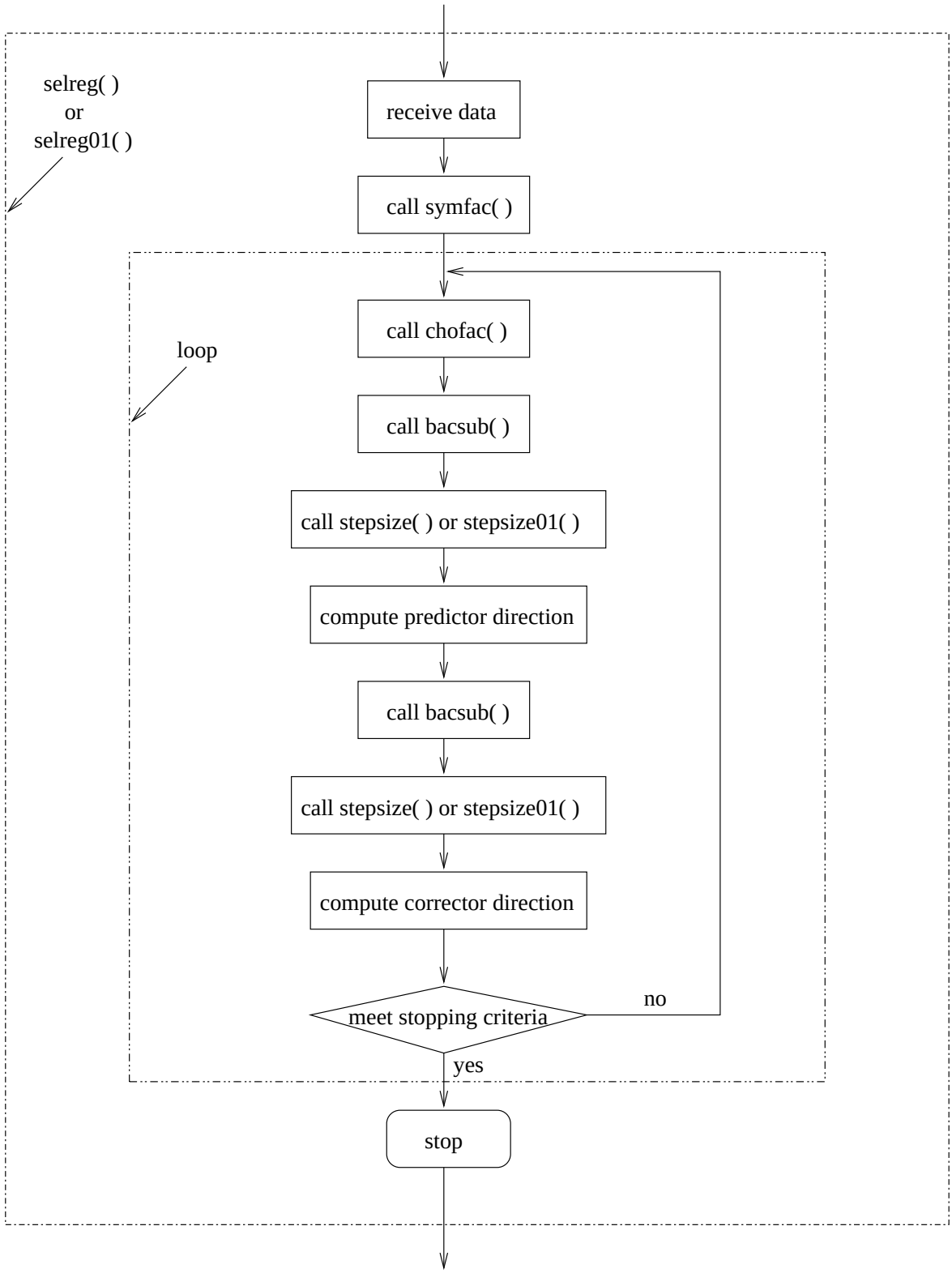

Figure 4.3 The structure of the subroutines *selreg*( ) and *selreg01*( )

(13) *call_iaja*( ) subroutine. This subroutine computes the index vectors ia and ja for matrix AA^T.

(14) *call_avals*( ) subroutine. This subroutine computes the nonzero element vector $avals$ for the matrix AD^2A^T.

(15) *symfac*( ) subroutine. It calls WSMP to perform *ordering* and *symbolic factorization* for the matrix AD^2A^T defined in (4.14) or (4.29).

(16) *chofac*( ) subroutine. The Cholesky factorization subroutine. At each iteration, it decomposes the matrix AD^2A^T and gets the Cholesky factors by calling WSMP.

(17) *bacsub*( ) subroutine. It calls WSMP to perform *back substitution* and *iterative refinement* for the system (4.14) or (4.29), and obtains the search direction of the Newton system by parameter passing at each iteration.

(18) *stepsize*( ) subroutine. It computes the *step length* according to the current points and search direction for the problem with variable upper bounds. The *step length* α can be calculated as follows:

$$\alpha_{max} := \arg\max_{\alpha \geq 0}\{(x,\ w,\ \tau,\ z,\ s,\ \kappa) + \alpha(\Delta x,\ \Delta w,\ \Delta\tau,\ \Delta z,\ \Delta s,\ \Delta\kappa) \geq 0\},$$

and

$$\alpha = 0.99995\alpha_{max}.$$

(19) *stepsize01*( ) subroutine. The other subroutine to calculate the *step length* for the problem without variable upper bounds. The *step length* α can be calculated as follows:

$$\alpha_{max} := \arg\max_{\alpha \geq 0}\{(x,\ \tau,\ s,\ \kappa) + \alpha(\Delta x,\ \Delta\tau,\ \Delta s,\ \Delta\kappa) \geq 0\},$$

and

$$\alpha = 0.99995\alpha_{max}.$$

(20) *postprocessing*( ) subroutine. Reverses the transformations done by *preprocessing*( ) subroutine.

The more detailed description of these subroutines is described in a separate document called "*Description of the subroutines*".

Chapter 5

Computational Results and Conclusions

In this chapter, we present our computational results using the Self-Regular based predictor-corrector IPM combining with the homogeneous self-dual embedding strategy which was discussed in the previous chapters of this thesis. Our comparisons include two parts: 1) we compare our testing results with the results obtained from the OSL package; 2) we compare the computational performance by evaluating the effect of the barrier parameter q in the Self-Regular proximity function. Finally, we get our conclusions based on these testing results and comparisons.

Most of the testing problems are from **netlib**, and some of the testing problems are from the OSL package. All our codes are written in IBM C and VisualAge C++ which are compatible with the standard C and C++. All our testing was run on the IBM RS/6000 work station. The operating system is IBM AIX 4.3 and the compiler[1] is IBM **xlC**.

5.1 Comparison with OSL

In this section, we compare the iteration number and optimal value obtained by our implementation to those obtained by the OSL solver. First, we test the problems which are from the OSL package. All these problems are very small (the size of these problems are all smaller than 30×30).

Table 5.1 gives the testing results of these problems. The problem names are listed in the first column. The iterations required by our implementation are given in the second column. The iterations required by OSL are given in the fourth column. Column 3 gives the optimal value obtained by our implementation. Column 5 gives the optimal value

[1]We use some existing packages (IBM OSL, ESSL and WSMP) in our implementation, thus we use compiler option:

-libosl.so -lessl -lpthread -lm_r -lxlf90_r -lwsmp .

obtained by the OSL solver.

From the results in Table 5.1, we find: 1) the iteration number of our implementation is less than or equal to the iteration number of the OSL for each of these small problems; 2) the total iteration number of our implementation is 54 and the average iteration number of our implementation is 5.4. On the other hand, the total iteration number of OSL is 106 and the average iteration number of OSL is 10.6; 3) the objective value computed by our implementation is very close to the objective value computed by the OSL solver for each of these small problems.

Table 5.1 Testing results of LO problems in small size

Problem	Iterations	Objective value	Iterations(OSL)	Objective value(OSL)
exlp1	5	3.236851	12	3.236842
exlp2	3	0.000001	14	0.0000007
exlp3	6	70.00000	8	70.00000
exlp4	4	7.000000	10	7.000000
exlp5	10	332.9160	9	332.9160
exlp6	6	28.00000	9	28.00000
exlp7	5	3.236855	13	3.236842
exlp8	5	3.236851	12	3.236842
exlp9	4	-100.0000	7	-100.0000
exlp10	6	1.500000	12	1.500000
Total	54		106	
Average	5.4		10.6	

Now we consider the testing problems from **netlib**. Table 5.2 summarizes the basic characteristics of these problems, such as the number of rows, columns and the the number of nonzeros of these problems.

Table 5.3 presents the computational results on these problems both from our implementation ($q = 1$) and the OSL solver. The problem names are listed in the first column. The iterations required by our implementation are given in the second column. The iterations required by OSL are given in the fourth column. Column 3 gives the optimal value obtained by our implementation. Column 5 gives the optimal value obtained by the OSL solver.

From the results in Table 5.3, we find: 1) the iteration numbers of our implementation are comparable with those of OSL; 2) our implementation is robust, it can solve all the testing problems, but there is one problem can not be solved by OSL.

Table 5.2 Problem Statistics

Problem name	Rows	Columns	Nonzeros	Problem name	Rows	Columns	Nonzeros
25fv47	822	1571	11127	forplan	162	421	4916
80bau3b	2263	9799	29063	ganges	1310	1681	7021
adlittle	57	97	465	gfrd-pnc	617	1092	3467
afiro	28	32	88	greenbea	2393	5405	31499
agg	489	163	2541	greenbeb	2393	5405	31499
agg2	517	302	4515	grow15	301	645	5665
agg3	517	302	4531	grow22	441	946	8318
bandm	306	472	2659	grow7	141	301	2633
beaconfd	174	262	3476	israel	175	142	2358
blend	75	83	521	kb2	44	41	291
bnl1	644	1175	6129	lotfi	154	308	1086
bnl2	2325	3489	16124	maros	847	1443	10006
boeing1	351	384	3865	maros-r7	3137	9408	151120
boeing2	167	143	1339	modszk1	688	1620	4158
bore3d	234	315	1525	nesm	663	2923	13988
brandy	221	249	2150	perold	626	1376	6026
capri	272	353	1786	pilot	1442	3652	43220
cycle	1904	2857	21322	pilot.ja	941	1988	14706
czprob	930	3523	14173	pilot.we	723	2789	9218
d2q06c	2172	5167	35674	pilot4	411	1000	5145
d6cube	416	6184	43888	pilot87	2031	4883	73804
degen2	445	534	4449	pilotnov	976	2172	13129
degen3	1504	1818	26230	recipe	92	180	752
dfl001	6072	12230	41873	sc105	106	103	281
e226	224	282	2767	sc205	206	203	552
etamacro	401	688	2489	sc50a	51	48	131
fffff800	525	854	6235	sc50b	51	48	119
finnis	498	614	2714	scagr25	472	500	2029
fit1d	25	1026	14430	scagr7	130	140	553
fit1p	628	1677	10894	scfxm1	331	457	2612
fit2d	26	10500	138018	scfxm2	661	914	5229
fit2p	3001	13525	60784	scfxm3	991	1371	7846

Table 5.2 Problem Statistics (Continue)

Problem name	Rows	Columns	Nonzeros
scorpion	389	358	1708
scrs8	491	1169	4029
scsd1	78	760	3148
scsd6	148	1350	5666
scsd8	398	2750	11334
sctap1	301	480	2052
sctap2	1091	1880	8124
sctap3	1481	2480	10734
seba	516	1028	4847
share1b	118	225	1182
share2b	97	79	730
shell	537	1775	4900
ship04l	403	2118	8450
ship04s	403	1458	5810
ship08l	779	4283	17085
ship08s	779	2387	9501
ship12l	1152	5427	21597
ship12s	1152	2763	10941
sierra	1228	2036	9252
stair	357	467	3857
standata	360	1075	3038
standgub	362	1184	3147
standmps	468	1075	3686
stocfor1	118	111	474
stocfor2	2158	2031	9492
stocfor3	16676	15695	74004
truss	1001	8806	36642
tuff	334	587	4523
vtp.base	199	203	914
wood1p	245	2594	70216
woodw	1099	8405	37478

Table 5.3 Computational comparison with OSL

Problem name	Iterations	Precise digits	Iterations(OSL)	Precise digits(OSL)
25fv47	31	8	26	9
80bau3b	38	5	43	12
adlittle	13	9	12	10
afiro	8	8	8	10
agg	22	5	22	11
agg2	22	9	18	11
agg3	21	8	17	11
bandm	19	6	16	10
beaconfd	10	8	8	11
blend	8	7	11	11
bnl1	34	7	25	7
bnl2	38	8	34	11
boeing1	23	7	23	9
boeing2	15	10	16	10
bore3d	16	10	18	12
brandy	20	10	15	12
capri	19	7	18	11
cycle	36	7	24	10
czprob	33	10	30	11
d2q06c	54	7	31	7
d6cube	18	6	21	8
degen2	12	11	14	9
degen3	14	11	19	11
dfl001	45	5	50	6
e226	17	10	21	6
etamacro	26	7	31	7
fffff800	32	7	30	7
finnis	24	7	25	6
fit1d	20	10	21	10
fit1p	19	10	16	10
fit2d	23	11	26	11
fit2p	21	9	Failed	Failed
forplan	25	6	24	6

Table 5.3 Computational comparison with OSL (continue)

Problem name	Iterations	Precise digits	Iterations(OSL)	Precise digits(OSL)
ganges	19	6	15	6
gfrd-pnc	20	10	16	10
greenbea	51	5	48	3
greenbeb	53	5	59	5
grow15	21	7	17	11
grow22	22	6	20	11
grow7	18	6	15	11
israel	20	7	21	9
kb2	16	11	15	11
lotfi	16	5	14	10
maros	26	8	24	8
maros-r7	14	7	13	11
modszk1	29	5	23	7
nesm	40	6	35	6
perold	39	6	34	6
pilot	50	4	30	4
pilot.ja	35	7	57	6
pilot.we	45	6	53	6
pilot4	35	8	31	7
pilot87	68	6	43	6
pilotnov	27	5	22	7
recipe	9	11	9	11
sc105	9	7	9	10
sc205	11	6	10	10
sc50a	9	6	9	9
sc50b	8	8	7	10
scagr25	15	7	15	11
scagr7	14	7	14	7
scfxm1	26	7	16	2
scfxm2	28	8	26	7
scfxm3	28	7	21	9

Table 5.3 Computational comparison with OSL (continue)

Problem name	Iterations	Precise digits	Iterations(OSL)	Precise digits(OSL)
scorpion	12	9	14	11
scrs8	24	5	21	5
scsd1	9	8	10	8
scsd6	11	8	12	12
scsd8	10	10	10	11
sctap1	19	9	15	10
sctap2	20	10	17	10
sctap3	22	10	17	12
seba	31	11	19	12
share1b	29	6	22	9
share2b	10	6	13	10
shell	26	5	19	10
ship04l	20	10	16	11
ship04s	20	9	15	11
ship08l	23	9	16	10
ship08s	21	9	15	11
ship12l	28	10	18	11
ship12s	24	11	17	11
sierra	20	9	18	11
stair	15	4	18	3
standata	14	8	15	10
standgub	14	8	15	9
standmps	19	9	20	12
stocfor1	14	8	15	10
stocfor2	31	7	21	11
stocfor3	55	6	33	5
truss	19	10	17	10
tuff	19	4	19	12
vtpbase	9	8	10	10
wood1p	15	5	19	4
woodw	27	5	25	10

Table 5.4 Computational results for the Kennington problems

Problem name	Iterations	Precise digits	Iterations(OSL)	Precise digits(OSL)
cre-a	30	6	33	7
cre-b	33	6	47	8
cre-c	38	9	34	8
cre-d	33	7	50	7
ken-07	16	5	14	8
ken-11	22	5	19	7
ken-13	24	6	24	7
ken-18	35	6	30	7
osa-07	31	6	24	7
osa-14	36	6	25	6
osa-30	33	5	35	5
osa-60	46	8	32	8
pds-02	34	8	21	8
pds-06	44	6	32	7
pds-10	48	4	46	7
pds-20	69	5	56	7

Table 5.4 presents the computational results for the Kennington problems. These problems are a set of 16 very large problems arising from Military Aircraft Applications. From the results in Table 5.4, we find:

(1) the total iteration number of our implementation is 572 and the total iteration number of OSL is 522. Furthermore, the average iteration number of our implementation is 36 and the average iteration number of OSL is 33;

(2) For the "cre-*" problems, the iteration numbers of our implementation are less than those of OSL; For the "ken-*" problems, the iteration numbers of our implementation are almost same as those of OSL; But for the "osa-*" and "pds-*" problems, the iteration numbers of our implementation are worse than those of OSL. The main reason probably is that the problems "cre-*", "ken-*", "osa-*" and "pds-*" have different structure, and then have different computational performance.

5.2 Comparison with Different q

In this section we compare the computational performance by evaluating the effect of the barrier parameter q in the Self-Regular proximity function.

In Table 5.5 we compare the number of iterations required to solve the very small testing problems with different q ($q = 1, 2, 3, 4$ and 5).

The results in Table 5.5 indicate that when q increases, the average number of iterations increases for these small problems.

Table 5.5 Comparison with different q for the very small problems

Problem	Iter ($q = 1$)	Iter ($q = 2$)	Iter ($q = 3$)	Iter ($q = 4$)	Iter ($q = 5$)
exlp1	5	5	5	6	6
exlp2	3	3	3	3	3
exlp3	6	6	6	6	6
exlp4	4	4	4	4	4
exlp5	10	10	10	10	10
exlp6	6	6	6	6	6
exlp7	5	5	6	6	6
exlp8	5	5	5	6	6
exlp9	4	5	5	5	5
exlp10	6	6	6	6	8
Total	54	55	56	58	60
Average	5.4	5.5	5.6	5.8	6

In Table 5.6 we compare the number of iterations required to solve the testing problems from **netlib** with $q = 1$, 2, 3.

From the results in Table 5.6, we find:

- When q increases, the average number of iterations increases;
- The total number of iterations for $q = 1$ is 2155 and the average number of iterations for $q = 1$ is 23;
- The total number of iterations for $q = 2$ is 2305 and the average number of iterations for $q = 2$ is 24.5. It is 6.9% more than the number of iterations for $q = 1$;
- The total number of iterations for $q = 3$ is 2576 and the average number of iterations for $q = 3$ is 27.4. It is 11.7% more than the number of iterations for $q = 2$.

Table 5.6 Computational comparison with different q

Problem name	Iterations ($q = 1$)	Iterations ($q = 2$)	Iterations ($q = 3$)
25fv47	31	33	38
80bau3b	38	42	57
adlittle	13	15	15
afiro	8	10	11
agg	22	24	25
agg2	22	23	25
agg3	21	23	26
bandm	19	19	21
beaconfd	10	11	11
blend	8	9	9
bnl1	34	37	42
bnl2	38	41	51
boeing1	23	26	27
boeing2	15	17	18
bore3d	16	18	20
brandy	20	20	23
capri	19	22	22
cycle	36	36	46
czprob	33	33	39
d2q06c	54	57	67
d6cube	18	20	21
degen2	12	13	13
degen3	14	15	15
dfl001	45	45	54
e226	17	19	23
etamacro	26	28	32
fffff800	32	36	37
finnis	24	24	25
fit1d	20	20	20
fit1p	19	20	22
fit2d	23	23	23
fit2p	21	21	25
forplan	25	25	25

Table 5.6 Computational comparison with different q (continue)

Problem name	Iterations ($q = 1$)	Iterations ($q = 2$)	Iterations ($q = 3$)
ganges	19	20	20
gfrd-pnc	20	21	24
greenbeb	53	53	69
grow15	21	21	22
grow22	22	28	28
grow7	18	19	21
israel	20	24	25
kb2	16	18	18
lotfi	16	17	17
maros	26	28	30
maros-r7	14	16	16
modszk1	29	29	29
nesm	40	40	43
perold	39	41	41
pilot	50	50	62
pilot.ja	35	38	38
pilot.we	45	46	51
pilot4	35	37	46
pilot87	68	68	80
pilotnov	27	30	32
recipe	9	10	10
sc105	9	11	11
sc205	11	12	13
sc50a	9	10	10
sc50b	8	9	9
scagr25	15	16	16
scagr7	14	14	14
scfxm1	26	27	27
scfxm2	28	30	30
scfxm3	28	28	31

Table 5.6 Computational comparison with different q (continue)

Problem name	Iterations ($q = 1$)	Iterations ($q = 2$)	Iterations ($q = 3$)
scorpion	12	13	13
scrs8	24	26	29
scsd1	9	11	12
scsd6	11	12	22
scsd8	10	11	12
sctap1	19	20	24
sctap2	20	20	22
sctap3	22	22	26
seba	31	31	33
share1b	29	30	33
share2b	10	11	11
shell	26	26	34
ship04l	20	21	22
ship04s	20	22	22
ship08l	23	24	33
ship08s	21	21	24
ship12l	28	33	35
ship12s	24	25	25
sierra	20	25	29
stair	15	16	16
standata	14	16	20
standgub	14	16	20
standmps	19	19	27
stocfor1	14	16	16
stocfor2	31	31	34
stocfor3	55	55	63
truss	19	20	22
tuff	19	21	21
vtpbase	9	11	12
wood1p	15	16	18
woodw	27	28	35
Total	2155	2305	2576
Average	23	24.5	27.4

Table 5.7 Comparison with different q for the kennington problems

Problem name	Iterations ($q = 1$)	Iterations ($q = 2$)	Iterations ($q = 3$)
cre-a	30	31	38
cre-b	33	36	47
cre-c	38	39	45
cre-d	33	33	37
ken-07	16	16	21
ken-11	22	22	24
ken-13	24	24	26
ken-18	35	35	36
osa-07	31	31	31
osa-14	36	36	38
osa-30	33	37	37
osa-60	46	46	49
pds-02	34	34	34
pds-06	44	44	48
pds-10	48	54	57
pds-20	69	69	69
Total	572	587	637
Average	35.5	36.5	39.5

In Table 5.7 we compare the number of iterations required to solve the kennington problems from **netlib** with $q = 1, 2, 3$.

From the results in Table 5.7, we find:

- Our implementation can solve the kennington problems stably and robustly with different q;
- When q increases, the average number of iterations increases;
- The total number of iterations for $q = 1$ is 572 and the average number of iterations for $q = 1$ is 35.5;
- The total number of iterations for $q = 2$ is 587 and the average number of iterations for $q = 2$ is 36.5. It is 2.6% more than the number of iterations for $q = 1$;
- The total number of iterations for $q = 3$ is 637 and the average number of iterations for $q = 3$ is 39.5. It is 8.5% more than the number of iterations for $q = 2$.

5.3 Dynamic Update of q

Computational results about the testing problems with different q are presented in the previous sections. In this section, we experiment with a so-called *dynamic method* that adapting increase q when the step length is short. The strategy of the dynamic method is:

- Set $q = 1$;
- Set parameter $steplength = 0.01$;
- Compute search direction;
- Calculate step length α;
- If $\alpha < steplength$, then
  - let $q = q + 2$;
  - let $\mu = \sqrt{\frac{x^T s + z^T w + \tau\kappa}{x^{-T} s^{-1} + z^{-T} w^{-1} + \tau^{-1}\kappa^{-1}}}$;
  - calculate search direction;
- Calculate step length α;
- Compute the new iterate;
- Let $q = 1$.

In table 5.8 we compare the number of iterations required to solve the testing problems with different q and dynamic update of q.

Table 5.8 Comparison with different q and dynamic q

Problem	Iter ($q = 1$)	Iter ($q = 2$)	Iter ($q = 3$)	Iter (dynamic)
cycle	36	36	46	37
czprob	33	33	39	34
d2q06c	54	57	67	57
pilot	50	50	62	48
scfxm1	26	27	27	27
shell	26	26	34	25
ship12l	28	33	35	28
ship12s	24	25	25	23
stocfor3	55	55	63	52
woodw	27	28	35	26

The results in Table 5.8 indicate that dynamic method can only improve the performance of our implementation very little. But dynamic method is probably a good idea for our implementation, there is still room to go further for this topic, we need to do a lot of tuning on this method.

5.4 Re-scaling of the Input Data

In this section, we discuss the effect of the re-scaling of the input data, and support our findings by the computational results.

In our implementation, we use OSL to do preprocessing. There are a couple of problems that were scaled poorly after OSL preprocessing. For example, the problem "FORPLAN" has been scaled poorly because some elements in vectors 'b' and 'u' are very big while some other elements are very small. For this problem our IPM implementation is not giving the solution in less than 100 iterations. If we re-scale this problem, then our algorithm can work well. Moreover, the problem "DFL001" has been poorly scaled on objective coefficient vector 'c'. For this problem our IPM implementation is not giving the solution in less than 100 iterations. If we re-scale this problem, then our algorithm can work correctly. Table 5.9 gives the computational results of these two problems with and without re-scaling.

Table 5.9 Computational results for re-scaling problems

Problem Name	dfl001	forplan
Iter ($q = 1$ before re-scaling)	failed	failed
Iter ($q = 1$ after re-scaling)	45	25
Iter ($q = 2$ before re-scaling)	failed	failed
Iter ($q = 2$ after re-scaling)	45	25
Iter ($q = 3$ before re-scaling)	failed	failed
Iter ($q = 3$ after re-scaling)	54	25

5.5 Results for the Infeasible Problems

In this section we discuss the infeasible LO problems. Table 5.10 gives the testing results of these problems. The problem names are listed in the first column. The rows, columns and nonzeros of the problems are listed in the second, third and fourth column, respectively. The iteration numbers of our implementation are listed in the fifth column, and The iteration numbers of OSL are listed in the last column.

From the results in Table 5.10, we find:

- Our implementation is robust, it can detect the infeasibility for all the problems; On the other hand, OSL has failed (no conclusion in 100 iterations) for 6 problems for detecting the infeasibility. From this point of view, our implementation is much stable than OSL for detecting the infeasibility of the testing problems.
- Our implementation can detect the infeasibility of the testing problems with low iteration number. On the other hand, OSL uses much more iterations to detect the infeasibility of the testing problems. From this point of view, our implementation is much better than OSL for detecting the infeasibility of the testing problems.

Table 5.10 Computational results for the infeasible problems

Problem name	Rows	Columns	Nonzeros	Iterations	Iterations(OSL)
bgdbg1	349	407	1485	*	*
bgetam	401	688	2489	*	*
bgprtr	3577	824	17604	8	23
box1	232	261	912	5	8
ceria3d	3577	824	17604	*	*
chemcom	289	720	2190	*	*
cplex1	3006	3221	10664	12	5
cplex2	225	221	1059	9	23
ex72a	198	215	682	6	14
ex73a	194	211	668	6	15
forest6	67	95	270	7	30
galenet	9	8	16	*	*
gosh	3793	10733	97257	*	*
gran	2569	2520	20151	*	*
greenbeainf	2505	5405	35159	*	*
itest2	10	4	17	*	*
itest6	12	8	23	*	*
klein1	55	54	696	14	failed
klein2	478	54	4585	16	failed
klein3	995	88	12107	18	failed
mondou2	313	604	1623	9	33
pang	362	460	2666	21	32
pilot4i	411	1000	5145	*	*
qual	324	464	1714	34	failed
reactor	319	637	2995	*	*
refinery	324	464	1714	12	failed
vol1	324	464	1714	30	failed
woodinfe	36	89	209	*	*

* *means the infeasibility of the problem is detected at preprocessing stage.*

5.6 Conclusions

Our implementation of the new IPMs is based on the homogenous embedding model and the Predictor-Corrector strategy. We gave the computational results both for the feasible and the infeasible problems in the previous sections.

From the computational results and comparisons presented in the previous sections, we can get the following conclusions:

(1) Our computational results are encouraging, and our implementation has comparable performance with the commercial solver OSL for solving LO problems.

(2) When q increases from 1 to 3, the average iteration number increases slowly.

(3) Our implementation is robust, it can solve small, medium and large LO problems with low iteration numbers. On the other hand, there is one problem can not be solved by OSL.

(4) Our implementation is better than OSL for detecting the infeasibility of the testing problems. For example, our implementation can detect the infeasibility for all the testing problems, but OSL has failed for 6 problems; The average iteration number of our implementation is much less than that of OSL for detecting the infeasibility of the testing problems.

Chapter 6

Further Work

We presented our computational results for the implementation of the new IPMs in Chapter 5. As a useful research tool, our implementation still has room to improve. In the last chapter of this thesis, we summarize our suggestions for the future work as follows:

(1) Develop an efficient algorithm to avoid the small step length in the iterations. Since the small step length is not only due to the computation of present iteration, but also due to the search direction of the previous iteration. For example, suppose we get a small step length at iteration n for an LO problem, we can avoid this trouble by adjusting the search direction and the step length at iteration $n-1$.

(2) Handle dense columns of the matrix A. Though dense columns do not occur often and not very important, we still need to handle it to make our implementation more complete.

(3) Handle numerical singularity of the matrix AD^2A^T. Though WSMP can handle this case now, we still need to consider it if we develop our own sparse matrix package in the future.

(4) Investigate some other proximity functions, compare the performance of different Self-Regular proximity functions, and propose an efficient strategy for choosing the proximity functions.

(5) Improve the performance of preprocessing and postprocessing. In preprocessing part, we take some results from the OSL preprocessing and do some more preprocessing by ourselves. The more efficient the preprocessing is, the better the performance is of our implementation. Thus we need to develop our own full-version efficient preprocessing programs. One of my colleagues in our lab is doing this job, and the independent library will be integrated with our existing implementation.

(6) Develop our own linear algebra package and sparse matrix package. We use ESSL to perform linear algebra operations, and use WSMP to solve Newton equations in our implementation. These two packages are too general and occupied too much

memory. They reduced the performance of our implementation to a certain extent. Thus we need to develop our own efficient packages to substitute ESSL and WSMP.

(7) Develop callable libraries. We can develop some other independent libraries such as the library of step length and the library of initialization. This modular method would be more efficient and flexible for our complete optimization solver.

Bibliography

[1] E.D. Andersen and K.D. Andersen. The MOSEK Interior Point Optimizer for linear programming: An Implementation of the Homogeneous algorithm. In H. Frenk, K. Roos, T. Terlaky and S. Zhang, Eds., *High Performance Optimization*, pages 197-232. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.

[2] E.D. Andersen, J. Gondzio, C. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, Ed., *Interior Point Methods of Mathematical Programming*, pages 189-252. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[3] 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, T.U. Delft, The Netherlands, 1998.

[4] K.D. Andersen. A Modified Schur-Complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Transactions on Mathematical Software*, Vol. 22, No. 3, page 348-356, 1996.

[5] R.E. Bixby. Solving Real-World Linear Programs: A Decade and more of Progress. *Operations Research*, Volume 50, No. 1, 3-15, 2002.

[6] CPLEX Optimization Inc.. ILOG AMPL CPLEX System User's Guide (Version 7.0), 2000.

[7] J. Czyzyk, S. Mehrotra, M. Wagner, and S.J. Wright. PCx User's Guide (Version 1.1), November 3, 1997.

[8] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[9] G.B. Dantzig. Linear Programming. *Operations Research*, Volume 50, No. 1, 42-47, 2002.

[10] Dash Associates. XPRESS-MP User Guide, 1999.

[11] J. Farkas. Theorie der einfachen ungleichungen. *J. Reine und Angewandte Mathematik*, 124:1-27, 1902.

[12] S.I. Gass. The First Linear-Programming Shoppe. *Operations Research*, Volume 50, No. 1, 61-68, 2002.

[13] A.J. Goldman and A.W. Tucker. Theory of linear programming. In H.W. Kuhn and A.W. Tucker, Eds., *Linear Inequalities and Related Systems*, Annals of Mathematical Studies, 38:63-97, Princeton University Press, Princeton, NJ, 1956.

[14] J. Gondzio. HOPDM (version 2.12) - A Fast LP Solver Based on a Primal-Dual Interior Point Method, *European Journal of Operational Research*, 85 (1995) 221-225.

[15] O. Güler. Limiting Behavior of the weighted central paths in linear programming. *Mathematical Programming*, 65:347-363, 1994.

[16] O. Güler and Y. Ye. Convergence behaviour of interior-point algorithms. *Mathematical Programming*, 60(2):215-228, 1993

[17] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4, 373-395, 1984.

[18] L.G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244:1093-1096, 1979. Translated into English in *Soviet Mathematics Doklady* 20, 191-194.

[19] M. Kojima, N. Megiddo, and S. Mizuno. A primal-dual infeasible interior-point algorithm for linear programming. *Mathematical Programming*, 61:263-280, 1993.

[20] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise. A Unified approach to interior point algorithms for linear complementarity problems, Volume 538 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, 1991.

[21] M. Kojima, S. Mizuno, and A. Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo, Ed., *Progress in Mathematical Programming: Interior Point and Related Methods*, pages 29-47. Springer Verlag, NY, 1989.

[22] L. McLinden. The analogue of moreau's proximation theorem, with applications to the nonlinear complementarity problems. *Pacific J. of Mathematics*, 88:101-161, 1980.

[23] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo, Ed., *Progress in Mathematical Programming: Interior Point and Related Methods*, pages 131-158. Springer Verlag, NY, 1989. Identical version in: *Proceedings of the 6th Mathematical Programming Symposium of Japan, Nagoya, Japan*, pages 1-35, 1986.

[24] S. Mehrotra. Higher order methods and their performance. Technical Report 90-16R1, Northwestern University, 1991.

[25] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2:575-601, 1992.

[26] S. Mizuno and A. Nagasawa. A primal-dual affine scaling potential reduction algorithm for linear programming. *Mathematical Programming*, 62:119-131, 1993.

[27] R.C. Monterio, I. Adler, and M.G.C. Resende. A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. *Mathematics of Operations research*, 15(2), 191-214.

[28] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*, McGraw-Hill, 1996.

[29] G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd. *Optimization*, North-Holland, 1989.

[30] M. Padberg. *Linear Optimization and Extensions*, Springer, 1999.

[31] J. Peng, C. Roos, and T. Terlaky. Self-Regular Proximities and New Search Directions for Linear and Semidefinite Optimization. Technical Report, AdvOL 2000/1, McMaster University, 2000.

[32] J. Peng. *New Design and Analysis of Interior Point Methods*, Ph.D Thesis, T.U. Delft, The Netherlands, 2001.

[33] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 50:59-93, 1988.

[34] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and Algorithms for Linear Optimization. An Interior Approach*. John Wiley & Sons, Chichester, UK, 1997.

[35] G. Sonnevend. An "analytic center" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In A. Prèkopa, J. Szelezsán and B. Strazicky, Eds., *System Modelling and Optimization: Proceeding of the 12th IFIP Conference held in Budapest, Hungary, September 1985*, volume 84 of *Lecture Notes in Control and Information Science*, pages 866-876. Springer Verlag, Berlin, West-Germany, 1986.

[36] T. Terlaky (Ed.). I*nterior-Point Methods of Mathematical Programming*. Kluwer Academic Publishers, 1996.

[37] A.W. Tucker. Dual systems of homogeneous linear relations. In H.W. Kuhn and A.W. Tucker, Eds., *Linear Inequalities and Related Systems, Annals of Mathematical Studies*, 38:3-18, Princeton University Press, Princeton, NJ, 1956.

[38] R.J. Vanderbei. LOQO User's Manual (Version 3.10). *Optimization Methods and Software*, 11/12:485-514, 1999.

[39] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM Publications, 1997.

[40] X. Xu, P.-F. Hung, and Y. Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62:151-171, 1996.

[41] Y. Ye. On the finite convergence of interior-point algorithms for linear programming. *Mathematical Programming*, 57:325-335, 1992.

[42] Y. Ye. *Interior Point Algorithms, Theory and Analysis*, John Wiley & Sons, Chichester, UK, 1997.

[43] Y. Ye, M. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53-67, 1994.

[44] D.B. Yudin and A.S. Nemirocskii. Informational Complexity and Effective Methods for Solving Convex Extremal Problems. Matekon, 13:24-45, 1977.

[45] Y. Zhang. User's guide to LIPSOL: linear-programming interior point solvers V0.4. *Optimization Methods and Software*, 11/12:385-396, 1999.