INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700  800/521-0600
RICE UNIVERSITY

Implementation of Multiple-Corrections and Presolve Enhancement in the Interior-Point Linear Programming Code LIPSOL

by

Yan Liu

A THESISSubmitted
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

Master of Arts

APPROVED, THESIS COMMITTEE:

Yin Zhang
Associate Professor of Computational and Applied Mathematics, Chairman

Richard A. Tapia
Noah Harding Professor of Computational and Applied Mathematics

Matthias Heinkenschloss
Associate Professor of Computational and Applied Mathematics

Houston, Texas
August, 1998
Implementation of Multiple-Corrections and Presolve Enhancement in the Interior-Point Linear Programming Code LIPSOL

Yan Liu

Abstract
Multiple correction was introduced by Gondzio [5] to accelerate the convergence of iterates generated by primal-dual interior-point algorithms for linear programming. In this work, we implement the multiple correction technique in the interior-point linear programming software LIPSOL with our modifications. In our implementation, we have used modified criteria to start and stop the multiple correction process in order to achieve high efficiency under the LIPSOL environment. In addition, we have implemented a more thorough presolve analysis to enhance the solver’s efficiency. The main presolve technique addressed in this work is to eliminate implied free variables. We have performed extensive computational experiments on the Netlib set of linear programs as well as on some larger linear programs from real-world applications. On the average, our implementation of the enhanced presolve and the multiple correction technique has resulted in a 10% to 20% saving in CPU time.
Acknowledgments

My first thanks go to my thesis advisor, Dr. Yin Zhang, for his endless encouragement, support, and advice. His enthusiasm and talent for doing research has always left me with a sense of inspiration.

I would like to express my heartfelt thanks to my committee members, Dr. Richard Tapia and Dr. Matthias Heinkenschloss, for reading my thesis and giving me precious advice.

I owe thanks to Wei Wang, Qiuyi Xia and Huiping Zhang for their warm-hearted help with my thesis.

Finally, I wish to give my special thanks to my parents for their affection and stimulation.
Contents

Abstract .................................................. ii
Acknowledgments ......................................... iii
List of Illustrations ........................................ vi
List of Tables ............................................... vii

1 Introduction ............................................ 1

2 Infeasible-Interior-Point Method ....................... 4
  2.1 Linear Programming ................................ 4
  2.2 Infeasible-Interior-Point Method .................... 6
  2.3 Features of LIPSOL ................................. 10

3 Implementation of Multiple Correction in LIPSOL ...... 12
  3.1 Multiple Centrality Correction ...................... 12
  3.2 Implementation Details ............................ 15
    3.2.1 Modified Beginning and Stopping Correction Criterion 16
    3.2.2 Scaling the right hand side vector ............... 17
    3.2.3 The Maximum Order of Correction ............... 18
    3.2.4 The Multiple Correction Algorithm Implemented in LIPSOL 19

4 Presolve Analysis and Implementation .................. 21
  4.1 Preprocessing in LIPSOL .......................... 21
  4.2 Free and Implied Free Variables ................... 22
    4.2.1 Implied Variable Bounds ...................... 22
    4.2.2 Reduced Linear Programming Problem ........... 25

5 Computational Results .................................. 27
  5.1 Size Reduction after Presolve Analysis ............. 27
  5.2 Test Results on the Netlib Problems ............... 28
  5.3 Test Results on other collections .................. 30
Illustrations

5.1 Total CPU time percentages for 95 Netlib Problems Solved by LIPSOL v0.41 ................................................................. 29
5.2 Total CPU time percentages for 95 Netlib Problems Solved by our techniques compared to LIPSOL v0.41 ............................ 30
Tables

5.1 Size reductions after presolve on Netlib problems (I) .................................. 31
5.2 Size reductions after presolve on some Netlib problems (II) ......................... 32
5.3 Efficiency of multiple correction and presolve techniques on some large Netlib problems .................................................................................. 33
5.4 Size reductions after presolve on large problems other than Netlib ............ 34
5.5 Efficiency of multiple correction and presolve techniques on large problems other than Netlib ................................................................. 34
Chapter 1

Introduction

Linear Programming has been the dominant paradigm in optimization since Dantzig’s development of the simplex method in the 1940s. In 1984, a wave of research into a new field of methods known as interior-point methods was triggered by the seminal work of Karmarkar [9]. After over a decade since then, not only do we have a solid theoretical foundation for interior-point methods for linear programming, but also a rather comprehensive understanding on their practical efficiency. Among many general algorithmic approaches, it is now widely accepted that an infeasible primal-dual interior point method is a powerful tool to solve very large linear programs.

Among all its implementation, Merhotra’s predictor-corrector [15] technique is widely used. In practice, his technique is incorporated into many modern codes and proved to be very efficient. It proceeds as follows: at each iteration two steps are computed, predictor step and corrector step, both using the same matrix factorization of the Karush-Kuhn-Tucker (KKT) equations. The paper [15] gives a rigorous presentation of this approach and rich computational results as well. The factorization involves almost always significantly more computational effort than the following solves for the predictor and corrector steps.

The method of Sonnevend, Stoer and Zhao [19] uses subspaces spanned by directions generated by higher order derivatives of the central path, or earlier computed points of it as a predictor step. This was later followed by one (or more) centering steps to drive the next iterate sufficiently close to the central path. This approach required feasibility and was tested only on small and dense problems.

Another approach, due to Domich, Boggs, Rogers and Witzgall [3] uses three
independent directions and solves an auxiliary linear program in a three dimensional subspace to find a search direction.


The computational results of Mehrotra in 1992 [16] confirmed the advantages of using a second order method over the use of a pure first order primal-dual algorithm but at the same time showed that introducing higher (than two) order terms had hardly predictable consequences for the overall efficiency measured with CPU time. The discouraging results on the use of the higher order predictor-corrector method were later confirmed in the paper of Karpenter [2].

However, an approach due to Gondzio [5] uses multiple correction to improve the centrality of the current iterate. An important issue of the approach is that the target for corrections is not an analytic center but it is some point in a large neighborhood of the central path that can be easier to reach. The maximum number of corrections depends on the ratio of the efforts to solve and to factorize the KKT systems, and is determined prior to optimization process. This approach has attracted the interest of many scientists in linear programming.

At present, there are several high-quality public-domain interior-point LP solvers, including BPMD, COPLLP, HOPDM, LIPSOL, LOQO, PCx, LPABO. Links to all codes are on http://plato.la.asu.edu/guide.html. There is also a comparison of their performance on a sample of frequently used LP test problems under this address. Several codes (HOPDM, PCx) has incorporate the multiple corrector technique. Some codes (BPMPD, HOPDM, PCx) have a more or less extensive presolver, while LOQO was deliberately written without one.

In this thesis, we describe the implementation of the Multiple Centrality Correction technique in the software package LIPSOL v0.41 [24]. LIPSOL is a Linear-programming Interior-Point SOLvers under the MATLAB environment. Our im-
plementation is based on the method proposed by Gondzio [5], but with different choice for beginning and stopping of corrections. The advantages of MATLAB's sparse-matrix functions is highly exploited. The harder the factorization, the more advantageous higher order corrections might prove.

We also put some effort on the presolve analysis for large scale linear programs. By deleting rows and columns and looking for implied free columns, the problem dimension is reduced. The uses of the presolve analysis and multiple centrality correction result in 10% to 20% savings in CPU time on the Netlib problems.

The thesis is organized as follows. In Chapter 2 we give a brief description of the primal-dual infeasible interior-point method and Merhotra's predictor-corrector technique. In Chapter 3 we introduce Gondzio's multiple centrality correction, and discuss a number of major issues in implementation. Chapter 4 is devoted to discussing presolve analysis. Numerical results are presented in Chapter 5. Finally in Chapter 6 we give our conclusions.
Chapter 2

Infeasible-Interior-Point Method

In this chapter, we start from a description of linear programming. Then we present the framework of primal-dual infeasible-interior-point algorithms. The algorithm generates iterates which are positive (i.e., are interior with respect to the inequality constraints) but do not necessarily satisfy the equality constraints.

The computationally most attractive infeasible-interior-point method is Mehrotra's predictor-corrector algorithm. Indeed it has been implemented in many software packages. Hence, we will concentrate on the features of this algorithm.

The practical implementations of the infeasible-interior-point methods still differ significantly. They have been continuously improved and encouraged further research in this field.

2.1 Linear Programming

The discussion in this section follows closely from section 2 in [23]. Consider the following primal linear program:

$$\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad Ax = b \\
& \quad 0 \leq x_i \leq u_i \quad i \in \mathcal{I} \\
& \quad 0 \leq x_i \quad i \in \mathcal{J}
\end{align*}$$

(2.1)

where $A \in \mathbb{R}^{m \times n}$, which determines the sizes of other vectors involved, and $\mathcal{I}$ and $\mathcal{J}$ are disjoint index sets such that $\mathcal{I} \cup \mathcal{J} = \{1, 2, \cdots, n\}$ and $\mathcal{I} \cap \mathcal{J} = \emptyset$.

Without loss of generality, let us assume that for some positive integer $n_u \leq n$

$$\mathcal{I} = \{1, 2, \cdots, n_u\}, \quad \text{and} \quad \mathcal{J} = \{n_u + 1, n_u + 2, \cdots, n\}.$$  (2.2)
Given a vector $x$ in $\mathbb{R}^n$, we use the notation $x_u$ for the vector in $\mathbb{R}^n_u$ whose elements are the first $n_u$ elements of $x$, i.e.,

$$[x_u]_i = x_i, \quad i = 1, 2, \ldots, n_u.$$  

Moreover, we define the appending operator "app" from $\mathbb{R}^n_u$ to $\mathbb{R}^n$ that appends $n - n_u$ zeros to vectors in $\mathbb{R}^n_u$, i.e., for $w \in \mathbb{R}^n_u$

$$[\text{app}(w)]_i = \begin{cases} w_i, & 1 \leq i \leq n_u \\ 0, & n_u < i \leq n. \end{cases}$$  

By adding the slack variables $s \in \mathbb{R}^n_u$, we can rewrite the above linear program into the standard form:

$$\begin{align*}
\min & \quad c^T x \\
\text{s.t.} & \quad Ax = b \\
& \quad x_u + s = u \\
& \quad x \geq 0, \ s \geq 0
\end{align*}$$  

(2.4)

The dual of the above standard primal linear program is:

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

(2.5)

where $y$, $z$ and $w$ are the dual variables and slacks.

It is well-known that the solutions of the primal and the dual linear programs, if they exist, satisfy the following KKT conditions which is a system of linear-quadratic equations with nonnegativity constraints on some variables:

$$F(x, z, s, w, y) = \begin{pmatrix} Ax - b \\ x_u + s - u \\ A^T y + z - \text{app}(w) - c \\ xz \\ sw \end{pmatrix} = 0, \ (x, z, s, w) \geq 0,$$  

(2.6)
where \( xz \) and \( sw \) denote component-wise multiplications and the equations \( xz = 0 \) and \( sw = 0 \) are called the complementarity conditions for the linear program.

For nonnegative variables \( x, s, z, w \), we will call the quantity \( x^Tz + s^Tw \) the duality gap (regardless of feasibility or infeasibility of these variables). The duality gap measures the residual of the complementarity portion of \( F \) in \( \ell_1 \)-norm when \( (x, z, s, w) \geq 0 \).

A straightforward calculation shows that the Jacobian matrix of \( F(x, z, s, w, y) \) is

\[
F'(x, z, s, w, y) = \begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
E^T & 0 & I_u & 0 & 0 \\
0 & I & -E & A^T & \\
Z & X & 0 & 0 & 0 \\
0 & 0 & W & S & 0
\end{bmatrix},
\]

(2.7)

where \( E^T = \begin{bmatrix} I_u & 0 \end{bmatrix} \), \( I_u \) is the identity matrix of dimension \( n_u \), \( X = \text{diag}(x) \), \( Z = \text{diag}(z) \), \( S = \text{diag}(s) \) and \( W = \text{diag}(w) \).

Unlike a primal (dual) method which concentrates on solving the primal (dual) program, a primal-dual interior-point method for linear programming solves the KKT system (2.6), which includes all the primal and dual variables and slacks.

### 2.2 Infeasible-Interior-Point Method

Rewrite the KKT system (2.6) into a constrained algebraic system of \( \ell \) equations and \( \ell \) variables with nonnegativity constraints on \( \ell_+ \) variables:

\[
F(v) = 0, \quad v_i \geq 0, \quad 1 \leq i \leq \ell_+,
\]

(2.8)

where \( v = (x, z, s, w, y) \), \( \ell = 2n + 2n_u + m \) and \( \ell_+ = 2n + 2n_u \).

For linear programming, \( F(v) \) contains only linear and quadratic equations and the quadratic ones are complementarity pairs.

In the presence of the nonnegativity constraints we cannot expect to obtain a nonnegative solution by just simply applying the Newton's method. However, by
applying damped Newton on unconstrained system of nonlinear equations:

$$v^{k+1} = v^k - \alpha^k F'(v^k)^{-1} F(v^k),$$  \hfill (2.9)

in which step length $\alpha^k$, is usually chosen from the interval $(0, 1]$, to enhance global convergence, the nonnegativity constraints are satisfied strictly at every iteration if $v_i^0, 1 \leq i \leq \ell_+$ are positive at starting point.

Keeping nonnegative variables strictly positive is not enough. Difficulty in recovering from a situation where a variable is adversely set to too small a value would be expected. To decrease the chances of such mistakes at early stages, it would be desirable that all the complementarity pairs converge to zero at the same pace. say $s_i^k w_i^k = \mu^k \to 0$ as $k \to \infty$ for every index $i$. Towards this goal (and for other theoretic considerations as well), one can perturb each complementarity condition $s_i w_i = 0$ into $s_i w_i - \mu = 0$ and let $\mu$ go to zero in a controlled fashion.

Define the vector $\hat{\epsilon} \in \mathbb{R}^n$ as

$$\hat{\epsilon}_i = \begin{cases} 0, & \text{if } F_i(v) \text{ is linear, } \\ 1, & \text{if } F_i(v) \text{ is quadratic.} \end{cases}$$

We present the primal-dual framework:

**Algorithm FIP:**

Choose an initial point $v^0$ such that $v_i^0 > 0$, $1 \leq i \leq \ell_+$. Set $k = 0$.

**Do** until $\|F(v^k)\|$ is "small"

1. Choose $\mu^k > 0$ and compute $\Delta v^k = -F'(v^k)^{-1} (F(v^k) - \mu^k \hat{\epsilon})$.

2. Choose $\alpha^k > 0$ and set $v^{k+1} = v^k + \alpha^k \Delta v^k$

such that $v_i^{k+1} > 0$, $1 \leq i \leq \ell_+$. Increment $k$ by 1.

**End**

The definition of $\hat{\epsilon}$ indicates that perturbations are only applied to the complementarity conditions, which are all quadratic, but not to the feasibility conditions, which are all linear.
The above primal-dual interior-point approach was first proposed by Kojima, Mizuno and Yorshisi [10], in which a feasible initial point is required and the linear equations in $F(v) = 0$ (i.e., feasibility equations) are always satisfied. According to the classification introduced in [21], this type of methods is called feasible-interior-point methods.

Since a strictly feasible starting point is very difficult to find, in 1989 Lustig, Marsten and Shanno [13] implemented a modified version of the Kojima-Mizuno-Yorshisi interior-point approach. A fundamental modification is the removal of the requirement for a feasible initial point. This approach called infeasible-interior-point method requires nothing more of the starting point than positivity of $v_i^0,$ $1 \leq i \leq \ell_+.$ And it do not require the feasibility of the linear equations during the optimization process.

Mehrotra [16] introduced an enhanced version of the infeasible-interior-point algorithm that he called a predictor-corrector method. It is very close to the following algorithmic framework:

**Algorithm IIP:**

Choose an initial point $v^0$ such that $v_i^0 > 0,$ $1 \leq i \leq \ell_+.$ Set $k = 0.$

**Do** until $\|F(v^k)\|$ is "small"

1. Compute $\Delta F v^k = -F'(v^k)^{-1}F(v^k)$ and determine $\mu^k > 0.$
2. Compute $\Delta F v^k = -F'(v^k)^{-1}(F(v^k + \Delta F v^k) - \mu^k \hat{c}).$
3. Choose $\alpha^k > 0$ and set $v^{k+1} = v^k + \alpha^k(\Delta F v^k + \Delta c v^k)$ such that $v_i^{k+1} > 0,$ $1 \leq i \leq \ell_+.$ Increment $k$ by 1.

**End**

Notice that a key feature of this algorithm is addition of a corrector step to the search direction. At each iteration, we need to solve the system with the coefficient matrix $F'(v)$ (see (2.7)) and two different right-hand sides, one for the predictor direction and the other for the corrector one. Both right-hand sides can be written in terms of vectors $r_b, r_u, r_c, r_{xz}$ and $r_{sw}$ as defined below. More specifically, the right-
hand sides for the predictor and the corrector directions are, respectively, the negative of the following two vectors:

\[
\begin{pmatrix}
r_b \\
r_u \\
r_c \\
r_{xz} \\
r_{sw}
\end{pmatrix} = \begin{pmatrix}
Ax - b \\
x_u + s - u \\
A^T y + z - \text{app}(w) - c \\
xz \\
sw
\end{pmatrix}
\text{ and }
\begin{pmatrix}
r_b \\
r_u \\
r_c \\
r_{xz} \\
r_{sw}
\end{pmatrix} = \begin{pmatrix}
0 \\
0 \\
(\Delta P x^k)(\Delta P z^k) - \mu^k c_n \\
(\Delta P s^k)(\Delta P w^k) - \mu^k e_n
\end{pmatrix}
\]

where \( e_n \) and \( e_{n_u} \) are vector of all ones of dimension \( n \) and \( n_u \), respectively. It is worth noting that the first vector is just \( F(v^k) \), and the second \( F(v^k + \Delta v^k) \) subtract the perturbation vector involving \( \mu^k \).

Since \( F'(v) \) is a highly structured sparse matrix, one can solve it by block Gaussian elimination which leads to the following procedure for both the predictor and the corrector steps corresponding to different right-hand sides:

**Procedure** Steps:

1. Form the matrix \( D = [X^{-1} Z + \text{diag}(\text{app}(S^{-1} w))]^{-1} \).
2. Overwrite: \( r_c \leftarrow r_c - X^{-1} r_{xz} + \text{app}(S^{-1}(r_{sw} - Wr_u)) \).
3. Solve for \( \Delta y \) by Cholesky factorization from the linear system: \( (ADA^T)\Delta y = -(r_b + ADr_c) \).
4. Compute \( \Delta x = DA^T(\Delta y + r_c) \),
   \( \Delta z = -X^{-1}(Z \Delta x + r_{xz}) \),
   \( \Delta s = -(\Delta x_u + r_u) \),
   \( \Delta w = -S^{-1}(W \Delta s + r_{sw}) \).

Compare to other infeasible-interior-point methods which compute only one step of direction, Mohrotra's infeasible-interior-point framework requires less iterations. Furthermore, since only one matrix factorization is required per iteration, it is more efficient than algorithms based on the Mizuno-Todd-Ye [17] predictor-corrector approach which requires two matrix factorizations per iteration.
In the variant of Mehrotra's predictor-corrector algorithm, the centering parameter \( \mu^k \) is chosen adaptively. After computing the predictor step \( \Delta P v^k \), the maximum step sizes in the primal \( \hat{\alpha}_p \) and in the dual, \( \hat{\alpha}_d \) spaces preserving nonnegativity of \((x,s)\) and \((z,w)\), respectively are determined and the predicted complementarity gap
\[
\hat{\gamma} = (x + \hat{\alpha}_p \Delta P x)^T (z + \hat{\alpha}_d \Delta P z) + (s + \hat{\alpha}_p \Delta P s)^T (w + \hat{\alpha}_d \Delta P w)
\]
is computed. It is then used to determine the centering parameter
\[
\mu = \left( \frac{\hat{\gamma}}{g} \right)^3 \frac{g}{n + n_u}
\]
(2.10)
where \( g = x^T z + s^T w \) denotes current complementarity gap. Let us observe that the term \( \hat{\gamma}/g \) measures the achievable progress in the predictor direction. If only a small step in the predictor direction can be made, then \( \hat{\gamma}/g \) is close to one and \( \mu = g/(n + n_u) \), which means that we only want to improve centrality. If the predictor direction offers considerable progress in the reduction of the complementarity gap, then more optimistic target (closer to the optimum) is chosen.

Mehrotra's predictor-corrector has been implemented in many software packages. LIPSOL (Linear-programming Interior-Point SOLvers) is the only one designed for the MATLAB environment. In the following section, we will give a brief introduction of its features.

### 2.3 Features of LIPSOL

LIPSOL was developed by Dr. Yin Zhang [24]. It solves large-scale linear programming problems under MATLAB version 4.0 and above. MATLAB, standing for matrix laboratory, is a high-level technical computing environment for numeric computation. It integrates numerical analysis, matrix computation and graphics in an easy-to-use environment where matrix-computational formulas can be expressed in MATLAB scripts, called M-files. MATLAB provides external interface facilities to enable interaction with programs written in Fortran or C languages in the form of the MEX-files, where MEX stands for MATLAB Executable.
LIPSOL utilizes MATLAB’s sparse-matrix data-structure and MEX external interface facility, and at the same time takes advantages of existing, efficient Fortran codes for solving large, sparse, symmetric positive definite linear systems. Specifically, LIPSOL constructs MEX-files from two Fortran packages: a sparse Cholesky factorization package (version 0.3) developed by Esmond Ng and Barry Peyton at the Oak Ridge National Laboratory (ORNL), and a multiple minimum-degree ordering package by Joseph Liu at University of Waterloo [11].

The algorithm in LIPSOL is based on Mehrotra’s predictor-corrector method. (see Algorithm IIIP above) with some modification in the way of computing the steps and choosing the parameters $\mu^k$ and $\alpha^k$. In addition, Dr. Zhang incorporated a number of features in LIPSOL, including the exploitation of sparsity and the treatment of numerical instability as well as dense-columns.

The extensive computational results demonstrate that LIPSOL attains an impressive performance comparable with that of efficient Fortran or C codes in solving large-scale problems. Moreover, LIPSOL inherits a high degree of simplicity and versatility in comparison to its counterparts in Fortran or C languages.
Chapter 3

Implementation of Multiple Correction in LIPSOL

Since the matrix factorization almost always involves significantly more computational effort than the following solves for the predictor and corrector direction terms, it is thus natural to look for the possibility of exploiting even more the information available from such expensive factorizations and to try to reduce their number to a minimum. This gave rise to introducing higher order terms when computing directions.

A successful higher order technique, proposed by Gondzio [5] called multiple centrality corrections can further reduce the number of iterations required for convergence by adaptively adding one or more corrector steps to the framework of Algorithm IIP.

In this chapter, we will introduce Gondzio’s multiple centrality correction method, and then describe our implementation in the software LIPSOL in some detail.

3.1 Multiple Centrality Correction

Multiple centrality correction approach is a new variant of a primal-dual method. Below, we present this approach in more detail. In the following description of the method, predictor direction is produced by second order predictor-corrector technique. The following corrector terms will be concerned with improving the centrality of the subsequent iterate and with increasing the step sizes in primal and dual spaces.

Assume \((x, s)\) and \((y, z, w)\) are primal and dual solutions at a given iteration of the primal-dual algorithm \((x, s, z, w)\) are strictly positive). Next, assume that a predictor direction \(\Delta P\) at this point is determined and the maximum step sizes in primal, \(\alpha_p\), and dual, \(\alpha_d\), spaces are computed that preserve nonnegativity of primal
and dual variables, respectively.

Then look for a corrector direction $\Delta^n$ such that larger step sizes in primal and dual spaces are allowed for a composite direction

$$\Delta = \Delta^p + \Delta^n$$

For this purpose, Gondzio suggested to enlarge these step sizes in the primal and dual spaces from $\alpha_p$ and $\alpha_d$ to $\tilde{\alpha}_p = \min(\alpha_p + \delta, 1)$ and $\tilde{\alpha}_d = \min(\alpha_d + \delta, 1)$, respectively. $\delta$ is some prescribed increase of step sizes. To make it possible, a corrector term $\Delta_m$ has to compensate for the negative components in the primal and dual variables

$$\begin{align*}
(\tilde{x}, \tilde{s}) &= (x, s) + \tilde{\alpha}_p(\Delta^p x, \Delta^p s), \\
(\tilde{y}, \tilde{z}, \tilde{w}) &= (y, z, w) + \tilde{\alpha}_d(\Delta^p y, \Delta^p z, \Delta^p w).
\end{align*}$$

As we expect advantages from improving the centrality of the next iterate, which is $(\tilde{x}, \tilde{s}, \tilde{y}, \tilde{z}, \tilde{w})$, an additional requirement is imposed on $\Delta^n$ term, namely, to drive the trial point (3.2) (3.3) back to the vicinity of the central path.

Theoretically, there exists much freedom in the choice of a target close to the central path. The most natural guess would be to drive the trial point (3.2) (3.3) to the analytic center, i.e., to define the following target in the space of the complementarity products

$$v = (\mu e, \mu e) \in \mathbb{R}^{2n},$$

where $\mu$ is the centering parameter chosen by the heuristic of Mehrotra (2.10). However, extensive computational tests showed this point to be much too optimistic, hence unreachable target. Instead, Gondzio [5] provides a new way to compute the target. He computes the complementarity products for the trial point,

$$\tilde{v} = (\tilde{X} \tilde{z}, \tilde{S} \tilde{w}) \in \mathbb{R}^{2n},$$
and projects them component-wise on a hypercube \( H = [\beta_{\min \mu}, \beta_{\max \mu}]^{2n} \) to define the target

\[
v_t = \pi(\tilde{v}|H) \in R^{2n}.
\]  

(3.6)

where \( \pi \) is a projector. Gondzio suggests that it is much better to concentrate the effort on correcting only out-lier complementarity products, which is the ones that do not belong to the interval \((\beta_{\min \mu}, \beta_{\max \mu})\).

A corrector term of the direction solves the linear system with the coefficient matrix \( F'(v) \) (see (2.7)) and the right hand side

\[
\begin{pmatrix}
r_b \\
r_u \\
r_c \\
r_xz \\
r_sw
\end{pmatrix} =
\begin{pmatrix}
0 \\
0 \\
0 \\
v_t - \tilde{v}
\end{pmatrix}
\]  

(3.7)

Note that the right hand side in the linear system has nonzero elements only in a subset of positions of \( v_t - \tilde{v} \) that refer to the complementarity products which do not belong to \((\beta_{\min \mu}, \beta_{\max \mu})\).

The modified centering direction, \( \Delta^m \) is now used to correct the search direction

\[
\Delta = \Delta^p + \Delta^m
\]  

(3.8)

The new step sizes in primal and dual spaces are determined and the primal-dual algorithm can move to the next iterate or continue the multiple correction process.

The correcting process can easily be repeated a desirable number of times. Direction \( \Delta \) of (3.8) becomes in such a case a new predictor \( \Delta^p \) for which a new trial point is computed from (3.2) (3.3). The point (3.5) in the complementarity products space is then used to define the new target (3.6). Next, a new modified centering direction \( \Delta^m \) that solves the linear system with the right hand side (3.7) is computed and added to the predictor term as in (3.8).
It is important to measure the improvement resulting from the use of the modified centering direction, especially if multiple correction are allowed. As Gondzio suggested[5], correcting terminates when the step sizes in the primal and dual space $\hat{\alpha}_p$ and $\hat{\alpha}_d$ determined for a composite direction (3.8) do not increase sufficiently compared with the step size $\alpha_p$ and $\alpha_d$ found earlier for a predictor direction. Namely, stop correcting if

$$\hat{\alpha}_p < \alpha_p + \gamma \delta_\alpha \quad \text{or} \quad \hat{\alpha}_d < \alpha_d + \gamma \delta_\alpha$$

(3.9)

where $\gamma$ is some prescribed tolerance.

Quite often, the effort to factorize the matrix is tens or even hundreds of times larger than the one required in the following solves. Whenever this is the case, the correcting process is repeated a desirable number of times. An advantage of the approach is that computing every single corrector term needs exactly the same effort.

The question arises about the choice of “optimal” number of corrections for a given problem. We incorporate Gondzio’s idea in the computation of maximum order of correction, which will be presented in section 3.2.2.

In the following section, we will talk about our implementation of multiple centrality correction method in the software LIPSOL v0.41[24].

### 3.2 Implementation Details

LIPSOL is an Linear-programming Interior-Point SOLvers under the MATLAB environment (see section 2.3). Our implementation is based on Gondzio’s multiple centrality correction method with some modifications.

In our implementation, the centering parameter $\mu$ is chosen by the heuristic that is very similar to Mehrotra’s one. It is based on the possible progress in the predictor direction $\Delta^p$ (see 2.10). Then we compute Mehrotra’s corrector direction $\Delta^c$. The sum of predictor and Mehrotra’s corrector direction

$$\Delta = \Delta^p + \Delta^c$$

(3.10)
becomes the first predictor direction for the multiple centrality correctors (see (3.1)). After that, we start correcting with the multiple centrality correctors. All these modified centering directions use the same $\mu$ as the one determined earlier.

In the following, we discuss our modifications of Gondzio's method and present the algorithm of multiple centrality corrections implemented in LIPSOL.

### 3.2.1 Modified Beginning and Stopping Correction Criterion

In Gondzio's method, the multiple centrality corrector step is always performed after Mehrotra's predictor and corrector step.

However, we observe that sometimes, the maximum step sizes in primal and dual spaces $\alpha_p$ and $\alpha_d$ for the composite direction (3.10) is rather small. This composite direction is then treated as the predictor step. In this case, the following multiple corrector step will have difficulty to make any improvement on the step sizes $\alpha_p$ and $\alpha_d$, which will result in the stopping of corrections (see (3.9)). If this is a general case for some problems, the effort we put on the multiple centrality effort will not make any progress.

Therefore, we modify the stopping correction criterion (3.9) to make it more adaptive. The multiple centrality corrector step is not always performed after Mehrotra's predictor and corrector steps.

Suppose that $(x, s)$ and $(y, z, w)$ ($(x, s, z, w) > 0$) are primal and dual solutions at the current iteration. Then, a predictor direction $\Delta_p$ at this point is determined, and a ratio test is performed to determine the maximum step sizes in primal, $\alpha_p$ and dual, $\alpha_d$ spaces that preserve nonnegativity of primal and dual variables. Let

$$\alpha^0 = \min(\alpha_p, \alpha_d) \quad (3.11)$$

Next, we compute Mehrotra's correct direction $\Delta^c$ (see the framework IIP). A ratio test is performed again for the composite direction $\Delta = \Delta_p + \Delta^c$, the step size determined are $\hat{\alpha}_p$ and $\hat{\alpha}_d$. Let
\[ \alpha^i = \min(\alpha^i_p, \alpha^i_d) \] (3.12)

We compare \( \alpha^0 \) with \( \alpha^i \), if \( \alpha^i \) is much less than \( \alpha^0 \), the multiple centrality correction is not performed. That is, we stop the multiple centrality correction if

\[ \alpha^i \leq \sigma \alpha^0 \] (3.13)

where \( \sigma < 1 \) is some prescribed tolerance. In our implementation \( \sigma \) is set to 0.9.

We observe that in Gondzio’s method, multiple corrector term is tried to improve the centrality of the next iterate and increase step lengths in the primal and dual spaces. Enlarging step lengths in both spaces and making a hypothetical further move along the the predictor direction to so called trial point (see (3.2) (3.3)) is for this purpose. By applying the rule (3.13), the multiple corrector term will be efficiently performed. When the step lengths \( \alpha^i_p, \alpha^i_d \) of the direction \( \Delta \) (3.8) do not increase compared with the predictor step lengths \( \alpha_p, \alpha_d \), this direction is not a good search direction. Therefore, not performing multiple correction will be a better decision in this situation.

### 3.2.2 Scaling the right hand side vector

Note that in Gondzio’s method, special care is taken to restore centrality of the next iteration. The trial points (3.2) (3.3) will cross, in general, the boundary of the feasible region, the multiple corrector is to drive this trial point towards the target (3.6). The analytic center is usually unachievable, while the defined target is some point in a large neighborhood of the central path that can be easier to reach.

In our implementation, we found that when we solve the linear system for the multiple corrector term with the right hand side (3.7), the nonzero part of \( \nu_i - \bar{\nu} \) remain badly scaled if there were very large complementarity products in \( \bar{\nu} \). To prevent the undesirable effect of this bad scaling, we take another scaling. All components of
\( v_i - \tilde{v} \) smaller than \(-\beta_{\text{max}} \mu\) are replaced with \(-\beta_{\text{max}} \mu\). Those larger than \(\beta_{\text{max}} \mu\) are replaced with \(\beta_{\text{max}} \mu\).

### 3.2.3 The Maximum Order of Correction

How many corrections do we need for a given problem? This depends on the ratio of the costs of factorization and solve of the KKT system. This ratio can be measured after preprocessing KKT system and before the optimization starts. In Gondzio's paper [5], a simple heuristic is used to determine the maximum number of centrality corrections \(K\) allowed when solving a given problem.

In our implementation in LIPSOL, we use the same way to determine \(K\). LIPSOL applies sparse Cholesky decomposition [24] to handle normal equations systems (see procedure step in section 2.2). Hence, after neglecting the operations of multiplying the LP constraint matrix \(A\), or its transpose, with a vector and that of building \(ADA^T\), we use the following measures [6] for factorization and solve efforts

\[
E_f = \sum_{i=1}^{m} l_i^2 \quad \text{and} \quad E_s = 2 \times \sum_{i=1}^{m} l_i + 12 \times n \tag{3.14}
\]

where \(l_i\) is the number of off-diagonal nonzero entries in column \(i\) of the Cholesky matrix and \(m, n\) are the LP problem sizes. The term \(12 \times n\) in \(E_s\) is expected to account for the necessary ratio tests and vector initializations.

We compute the ratio \(r_f/s = E_f/E_s\) and allow one centrality corrector if \(r_f/s\) exceeds 10. In such a case an additional solve adds less than 10% to a single primal-dual iteration but we expect at least 15% reduction of the iteration count, hence a saving in the overall CPU time. If \(r_f/s \leq 10\), then no centrality corrector is added, so the method reduces to the classical Mehrotra’s predictor-corrector.

Due to smaller expected savings resulting from the use of the second centrality corrector, second centrality corrector is allowed only if \(r_f/s\) exceeds 30. Third correction is allowed if \(r_f/s\) exceeds 50. More than three correctors are allowed only for problems with very expensive factorization. Namely, if \(r_f/s > 50 \times p\), then \(K = p + 2\)
correctors can be added. We do not allow $K$ to exceed 10 but we also observed that problems where centrality correctors of order higher than 4 are used appear very rarely.

### 3.2.4 The Multiple Correction Algorithm Implemented in LIPSOL

In this section, we present the algorithm we use to implement multiple centrality correction method in LIPSOL.

**Algorithm** MCC:

**Input:**

$(x, s)$ and $(y, z, w)$: current pair of primal and dual solutions;

**Parameters:**

$\beta_{\min}, \beta_{\max}$: relative threshold value for out-lier complementarity products;

$\delta_\alpha$: the required increase of step sizes;

$\gamma$: the minimum acceptable increase of step sizes;

$K$: the maximum number of multiple centrality correctors allowed;

$\sigma$: the relative threshold value for starting the multiple correction steps;

**Initialize:**

$\Delta^p$: predictor direction

(Mehrotra’s predictor direction, see Algorithm IIP in section 2.2);

$\alpha_p, \alpha_d$: step sizes along $\Delta^p$ in the primal and dual spaces, respectively;

$k = 0$: Mehrotra’s correction (see Algorithm IIP in section 2.2);

$\alpha^0 = \min(\alpha_p, \alpha_d)$;

**Begin the procedure:**

while $k < K + 1$

if $(k == 0)$

compute Mehrotra’s corrector $\Delta^c$;

perform the ratio test for a composite direction (3.10);

compute $\alpha^1$ from (3.12);
Test for starting the multiple correction:
if \((\alpha^1 \leq \sigma \alpha^0)\)
terminate correction and end the procedure:
else
\(\Delta^p = \Delta;\)
end
else
compute trial point (3.2) (3.3);
define the target (3.6);
compute the corrector \(\Delta^n\) with the right hand side (3.7);
perform the ratio test for a composite direction (3.8);
Test for improvement:
if \((\hat{\alpha}_p \geq \alpha_p + \gamma \delta \alpha \text{ and } \hat{\alpha}_d \geq \alpha_d + \gamma \delta \alpha)\)
\(k = k + 1;\)
\(\Delta^p = \Delta;\)
else
\(\Delta = \Delta^p;\)
terminate correction and end the procedure;
end
end
End the procedure
Output:
\(\Delta: \text{ direction after multiple centrality correction.}\)

In our implementation \(\beta_{\text{min}} = 0.1, \beta_{\text{max}} = 10, \delta_\alpha = 0.1, \gamma = 0.1 \text{ and } \sigma = 0.9.\)
The choice of the maximum number of correctors allowed, \(K\) is specified in the last section. This algorithm is implemented in the M-file MulCorrect.m in LIPSOL.
Chapter 4

Presolve Analysis and Implementation

Many linear programs are formulated in a way that is not necessarily the most suitable for a direct application of an LP solver. It is known, almost since the first applications of linear programming, that a presolve analysis of the problem before passing it to a solver often reduces solution time considerably.

The computational effort in interior point methods is dominated by the orthogonal projections, as we see in the last chapters. In general, the smaller constraint matrix $A$, the cheaper are the orthogonal projections. Unless the problem is irreducible, time spent in the presolve analysis will almost always pay off in terms of overall CPU time.

In this chapter, we discuss our extension of preprocessing part in LIPSOL. This part of work aims at reducing the problem dimension. Detecting unboundedness or infeasibility naturally follows, in most cases, from this part. Let us mention that the techniques discussed here already appeared in the paper of Brearley, Mitra and Williams ([1]). We start from a description of the presolve techniques in LIPSOL.

4.1 Preprocessing in LIPSOL

In LIPSOL, the preprocessing function preprocess.m performs a number of tasks:

- Delete empty rows and column

  Depending on the type of the constraint and of the sign of a right hand side component, an empty row must either be redundant or infeasible. Depending on the presence of finite bounds and the sign of the objective coefficient, a variable referring to an empty column is either fixed to one of its bounds, or the problem is unbounded (or dual infeasible).
• Delete fixed variables

If a variable’s upper bound and lower bound share the same value, this variable is fixed to its bounds.

• Singleton rows

A singleton row uniquely determines the value of a variable that appears in it. If this value is beyond the variable’s bounds, then the problem is infeasible.

This function also perform other tasks, such as checking obvious infeasibility, shifting nonzero lower bounds to zero, whenever any of these events occurs.

4.2 Free and Implied Free Variables

Free variables sometimes appear in LP problems. Usually, each one is replaced by two nonnegative ones. Implied free variable could be found by comparing the implied bounds with the original bounds \( l_k \) and \( u_k \). If the implied bounds are at least as tight as the original bounds, then the original bounds can be removed and the variables becomes free.

Free variables are highly undesirable for an interior point solver, especially the one using normal equations approach to compute orthogonal projections.

However, if a free variable refers to a singleton column, it can be removed from the problem formulation since its elimination does not affect any other constraint.

In our implementation, we check all singleton columns for free or implied free variables. If such a variable is found, then the constraint is removed, and the objective function is adjusted. In the following section we introduce the way to find an implied free variable.

4.2.1 Implied Variable Bounds

Assume all LP (2.1) variables have explicit (possibly infinite) bounds.

\[
0 \leq x \leq u
\] (4.1)
These bounds can be used to determine the following lower and upper limits for a constraint $i$

$$b_i = \sum_{k \in N_{i^*}} a_{ik} u_k,$$

and

$$\bar{b}_i = \sum_{k \in P_{i^*}} a_{ik} u_k,$$

where $P_{i^*} = \{k : a_{ik} > 0\}$ and $N_{i^*} = \{k : a_{ik} < 0\}$.

Suppose that column $j$ is a singleton column of matrix $A$. The row number of the nonzero entry is $i$. Could the variable $x_j$ be free?

First, we compute the implied bounds of $x_j$ by $b_i$ and $\bar{b}_i$. If $j \in P_{i^*}$, we have

$$b_i + a_{ij} x_j \leq \sum_k a_{ik} x_k \leq \bar{b}_i,$$  \hspace{1cm} (4.2)

and

$$\bar{b}_i + a_{ij} (x_j - u_j) \geq \sum_k a_{ik} x_k \geq b_i.$$  \hspace{1cm} (4.3)

If $j \in N_{i^*}$, we have

$$b_i + a_{ij} (x_j - u_j) \leq \sum_k a_{ik} x_k \leq \bar{b}_i,$$  \hspace{1cm} (4.4)

and

$$\bar{b}_i + a_{ij} x_j \geq \sum_k a_{ik} x_k \geq b_i.$$  \hspace{1cm} (4.5)

These give new implied bounds on $x_j$

$$l'_j \leq x_j \leq u'_j$$

where

$$u'_j = (b_i - \bar{b}_i)/a_{ij} \quad j \in P_{i^*},$$  \hspace{1cm} (4.6)
\[ u'_j = \frac{(b_i - \overline{b_i})}{a_{ij}} \quad j \in N_{i^*} \] 

(4.7)

and

\[ l'_j = u_j + \frac{(b_i - \overline{b_i})}{a_{ij}} \quad j \in P_{i^*} \] 

(4.8)

\[ l'_j = u_j + \frac{(b_i - b_i)}{a_{ij}} \quad j \in N_{i^*} \] 

(4.9)

Notice that if \( u_j = +\infty \), the computation of lower bounds (4.8) and (4.9) will cause numerical instabilities. In such a case, (4.3) and (4.5) are replaced by the following inequalities

\[ a_{ij}x_j + \sum_{k \in P_{i^*} - \{j\}} a_{ik}u_k \geq \sum_k a_{ik}x_k \geq b_i, \quad \text{if} \ j \in P_{i^*} \]

or

\[ a_{ij}x_j + \sum_{k \in N_{i^*} - \{j\}} a_{ik}u_k \leq \sum_k a_{ik}x_k \leq b_i, \quad \text{if} \ j \in N_{i^*} \]

giving

\[ x_j \geq l'_j = \left( b_i - \sum_{k \in P_{i^*} - \{j\}} a_{ik}u_k \right) / a_{ij}, \quad \text{if} \ j \in P_{i^*} \]

and

\[ x_j \geq l'_j = \left( b_i - \sum_{k \in N_{i^*} - \{j\}} a_{ik}u_k \right) / a_{ij}, \quad \text{if} \ j \in N_{i^*} \]

respectively.

If the implied bounds \( l'_j \) and \( u'_j \) on the variable \( j \) are at least as tight as the original bounds, which means that

\[ 0 \leq l'_j \leq x_j \leq u'_j \leq u_j \]

then the original bounds can be moved and the variable \( j \) becomes free. If this is the case, since the variable \( j \) refers to a singleton column, this variable can be
removed from the problem formulation since its elimination does not affect any other constraint. Correspondingly, the constraint $i$ that implies bounds on the variable $j$ can also be eliminated. However, this constraint cannot be forgotten as it conveys information about the linear dependency of the eliminated variables $j$ with respect to the variables that remain in the reduced problem. This information will be used to recover the optimal value of an implied free variable from the solution of the reduced problem. The last issue is how to get the reduced problem?

4.2.2 Reduced Linear Programming Problem

First, we have to make sure that the objective function of the reduced problem matches the original objective function. The analysis of a dual constraint (see (2.5)) indicates that whenever we got a singleton implied free variable $x_j$, it determines a unique value for the associated dual variable

$$y_i = c_j / a_{ij}$$

$x_j$ could be expressed as the linear combination of the rest variables

$$x_j = (b_i - \sum_{k \neq j} a_{ik} x_k) / a_{ij}. \quad (4.10)$$

Taking the contribution of $x_j$ into account, the objective function could be modified as follows:

$$c^T x = c'^T x' + y_i b_i \quad (4.11)$$

where

$$c' = (c_1 - y_i a_{i1}, \ldots, c_{j-1} - y_i a_{i,j-1}, c_{j+1} - y_i a_{i,j+1}, \ldots, c_n - y_i a_{in})^T,$$

and

$$x' = (x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n)^T.$$
Then the original LP problem is reduced to

\[
\begin{align*}
\min & \quad c^T x' \\
\text{s.t.} & \quad A' x' = b' \\
& \quad 0 \leq x' \leq u'
\end{align*}
\]  

(4.12)

where \( A', b' \) and \( u' \) are the matrix or vector after elimination of row \( i \) from \( A, b \) and \( u \) respectively. (For the convenience of presolve analysis, we have assumed that all variables have explicit, possibly infinite bounds.)

After solving the reduced LP problem, the eliminated rows are used in a postsolve analysis to compute the value of a free variable and to restore the original objective function from (4.10) and (4.11).

The above procedures are implemented in preprocess.m and postprocess.m in LIP-SOL. Our numerical experiments show that this approach results in a good improvement of the efficiency.
Chapter 5

Computational Results

In this chapter, we describe our numerical experiments and present computational results. The software LIPSOL under which we implement the higher order correction method is LIPSOL v0.41. The MATLAB version is v5.1.

Our numerical experiments were performed on Sparc station 10 Model 41 with 128 Megabytes of memory and 1 Megabytes of cache. The operating system is SunOS 5.5.1. We used the flags -O -mips2 to compile the MEX-files in LIPSOL.

The test problems used in our experiments were the Netlib set of linear programs. Netlib set is an electronically distributed collection of linear programs originated and maintained by David Gay [4] at the Bell Laboratory. The collection includes a number of fairly large problems from real-world applications. The size of the problems ranges from a few dozen variables to over fifteen thousands. Over the years, the Netlib set has become the standard set of test problems for testing and comparing algorithms and software for linear programming. In the following, we compare the results of our implementation with LIPSOL v0.41. As we noted before, LIPSOL is based on the framework of Algorithm IIP.

5.1 Size Reduction after Presolve Analysis

We test our implementation of the enhanced presolve and multiple centrality corrections methods on the 95 linear programming problems in the Netlib collection.

Table (5.1) and (5.2) demonstrate the size reduction for the Netlib problems after applying our presolve technique. The results clearly show that this technique usually reduce the size of the problems, sometimes in a considerable way. The size reduction
of the problems contributes a lot to the improvement of the efficiency.

5.2 Test Results on the Netlib Problems

Our approach of multiple correction was also incorporated into our implementation. The heuristic of Section 3.2.3 proved very successful in the choice of the optimal order of corrections appropriate for a given problem.

According to the estimate of the maximum order of correctors, multiple centrality correction was allowed in the case of 75 problems. This gave at least a 20% reduction of the total iteration number (compared with LIPSOL v0.41). Only in a case of 9 problems (known to be rather unstable, like GREENBEA) did our procedure produce a increase in the iteration count.

Since we did not take into account all intermediate steps in the computations of ratio \(r_{jj}\), (see (3.14)), sometimes the decrease of the iteration number does not make savings in the CPU time, especially for small size problems. We observe that the CPU time saving vary for different problems but they reach, in the average, 10% and show a tendency to increase for more difficult problems.

To demonstrate that our presolve and multiple correction techniques result in savings in CPU time on difficult and large scale problems, we divide the 95 Netlib problems into two sets. Set \(P\) includes 53 problems. Our techniques save the CPU time on these problems (compared with LIPSOL v0.41). Set \(Q\) are the other 42 problems which CPU time did not decrease when solved by our techniques.

The following two pie charts, Fig. 5.1 and Fig. 5.2 clearly show the CPU time change. Fig. 5.1 displays the percentage that the CPU time summation of each problem set contributes to the total CPU time of all test problems solved by LIPSOL v0.41. Denote \(t_i^0\) as the CPU time of the problem \(i\) solved by LIPSOL v0.41, the chart indicates,

\[
\frac{\sum_{i \in P} t_i^0}{\sum_{i \in P \cup Q} t_i^0} \times 100\% = 87\% \quad \text{and} \quad \frac{\sum_{i \in Q} t_i^0}{\sum_{i \in P \cup Q} t_i^0} \times 100\% = 13\%.
\]
which shows that the CPU time of the problem set $\mathcal{P}$ makes far great contribution towards the total CPU time.

Fig. 5.2 shows the percentage change when the problems are solved by our implementation of presolve and multiple correction method. Denote $t_i^1$ as the CPU time of the problem $i$ solved by our techniques, the percentage of the two pieces are:

$$\frac{\sum_{i \in \mathcal{P}} t_i^1}{\sum_{i \in \mathcal{P} \cup \mathcal{Q}} t_i^0} \times 100\% = 73\% \quad \text{and} \quad \frac{\sum_{i \in \mathcal{Q}} t_i^1}{\sum_{i \in \mathcal{P} \cup \mathcal{Q}} t_i^0} \times 100\% = 14\%.$$

The missing part of the pie is the total CPU time saving, about 13%. Since the problems in set $\mathcal{Q}$ are generally very small, their increased CPU time does not affect the savings on the total problems. The analysis clearly indicates the advantages for the use of multiple correction and presolve techniques for large scale problems.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{chart.png}
\caption{Total CPU time percentages for 95 Netlib Problems Solved by LIPSOL v0.41}
\end{figure}
Figure 5.2  Total CPU time percentages for 95 Netlib Problems Solved by our techniques compared to LIPSOL v0.41

Table (5.3) shows the time saving for some larger Netlib problems.

5.3 Test Results on other collections

In the last experiment, we use problems from other collections, which are not included in the Netlib. These problems are large scale industry models, which are more difficult to solve. Some of them arise from real world applications, see ([20]), ([12]) and ([18]). All the problems could be found at the web site: http://plato.la.asu.edu/guide.html.

Table (5.4) gives their statistics in original form and after presolve reduction. Table (5.5) reports the CPU time of solving them. We compared our results with LIPSOL v0.41 in these two tables. Again, the analysis of Table (5.4) (5.5) results shows that the CPU time saving over predictor-corrector primal-dual methods are evident on large scale problems.
<table>
<thead>
<tr>
<th>Problem</th>
<th>Original size</th>
<th>After LIPSOL presolve</th>
<th>After our presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m</td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td>25fv47</td>
<td>821</td>
<td>1876</td>
<td>798</td>
</tr>
<tr>
<td>80ban3b</td>
<td>2262</td>
<td>12061</td>
<td>2235</td>
</tr>
<tr>
<td>agg</td>
<td>488</td>
<td>615</td>
<td>488</td>
</tr>
<tr>
<td>beaconfd</td>
<td>173</td>
<td>295</td>
<td>148</td>
</tr>
<tr>
<td>blend</td>
<td>74</td>
<td>114</td>
<td>74</td>
</tr>
<tr>
<td>bnl1</td>
<td>643</td>
<td>1586</td>
<td>632</td>
</tr>
<tr>
<td>bnl12</td>
<td>2324</td>
<td>4486</td>
<td>2268</td>
</tr>
<tr>
<td>boeing1</td>
<td>351</td>
<td>726</td>
<td>347</td>
</tr>
<tr>
<td>boeing2</td>
<td>166</td>
<td>305</td>
<td>140</td>
</tr>
<tr>
<td>bora3d</td>
<td>233</td>
<td>334</td>
<td>199</td>
</tr>
<tr>
<td>brandy</td>
<td>220</td>
<td>303</td>
<td>149</td>
</tr>
<tr>
<td>csprob</td>
<td>929</td>
<td>3562</td>
<td>737</td>
</tr>
<tr>
<td>capri</td>
<td>271</td>
<td>496</td>
<td>267</td>
</tr>
<tr>
<td>cycle</td>
<td>1903</td>
<td>3378</td>
<td>1801</td>
</tr>
<tr>
<td>d2q06c</td>
<td>2171</td>
<td>5831</td>
<td>2171</td>
</tr>
<tr>
<td>df1001</td>
<td>6071</td>
<td>12230</td>
<td>6071</td>
</tr>
<tr>
<td>e226</td>
<td>223</td>
<td>472</td>
<td>220</td>
</tr>
<tr>
<td>etamacro</td>
<td>400</td>
<td>816</td>
<td>357</td>
</tr>
<tr>
<td>fffff600</td>
<td>524</td>
<td>1028</td>
<td>501</td>
</tr>
<tr>
<td>finnis</td>
<td>497</td>
<td>1064</td>
<td>492</td>
</tr>
<tr>
<td>forplan</td>
<td>161</td>
<td>492</td>
<td>135</td>
</tr>
<tr>
<td>gangsas</td>
<td>1309</td>
<td>1706</td>
<td>1137</td>
</tr>
<tr>
<td>greenbca</td>
<td>2392</td>
<td>5598</td>
<td>2318</td>
</tr>
<tr>
<td>greenbeb</td>
<td>2392</td>
<td>5602</td>
<td>2317</td>
</tr>
<tr>
<td>lottfi</td>
<td>153</td>
<td>366</td>
<td>151</td>
</tr>
<tr>
<td>maros</td>
<td>846</td>
<td>1966</td>
<td>835</td>
</tr>
<tr>
<td>maros-r7</td>
<td>3136</td>
<td>9408</td>
<td>3136</td>
</tr>
<tr>
<td>modsmt1</td>
<td>687</td>
<td>1622</td>
<td>686</td>
</tr>
</tbody>
</table>
Table 5.2  Size reductions after presolve on some Netlib problems (II)

<table>
<thead>
<tr>
<th>Problem</th>
<th>Original size</th>
<th>After LIPSOL presolve</th>
<th>After our presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m</td>
<td>a</td>
<td>m</td>
</tr>
<tr>
<td>nemm</td>
<td>662</td>
<td>3105</td>
<td>654</td>
</tr>
<tr>
<td>perold</td>
<td>625</td>
<td>1594</td>
<td>625</td>
</tr>
<tr>
<td>pilot4</td>
<td>410</td>
<td>1211</td>
<td>402</td>
</tr>
<tr>
<td>pilot87</td>
<td>2030</td>
<td>6680</td>
<td>2030</td>
</tr>
<tr>
<td>pilotja</td>
<td>940</td>
<td>2355</td>
<td>924</td>
</tr>
<tr>
<td>pilotwe</td>
<td>722</td>
<td>3008</td>
<td>722</td>
</tr>
<tr>
<td>pilotov</td>
<td>975</td>
<td>2446</td>
<td>961</td>
</tr>
<tr>
<td>pilotx</td>
<td>1441</td>
<td>4860</td>
<td>1441</td>
</tr>
<tr>
<td>recipe</td>
<td>91</td>
<td>204</td>
<td>85</td>
</tr>
<tr>
<td>scfxml</td>
<td>330</td>
<td>600</td>
<td>322</td>
</tr>
<tr>
<td>scfxm2</td>
<td>660</td>
<td>1200</td>
<td>644</td>
</tr>
<tr>
<td>scfxml3</td>
<td>990</td>
<td>1800</td>
<td>966</td>
</tr>
<tr>
<td>scorpion</td>
<td>388</td>
<td>466</td>
<td>375</td>
</tr>
<tr>
<td>scrs8</td>
<td>490</td>
<td>1275</td>
<td>485</td>
</tr>
<tr>
<td>sctapl</td>
<td>300</td>
<td>660</td>
<td>300</td>
</tr>
<tr>
<td>sctap2</td>
<td>1090</td>
<td>2500</td>
<td>1090</td>
</tr>
<tr>
<td>sctap3</td>
<td>1480</td>
<td>3340</td>
<td>1480</td>
</tr>
<tr>
<td>ship04l</td>
<td>402</td>
<td>2166</td>
<td>356</td>
</tr>
<tr>
<td>ship04s</td>
<td>402</td>
<td>1506</td>
<td>268</td>
</tr>
<tr>
<td>ship08l</td>
<td>778</td>
<td>4363</td>
<td>688</td>
</tr>
<tr>
<td>ship08s</td>
<td>778</td>
<td>2467</td>
<td>416</td>
</tr>
<tr>
<td>ship12l</td>
<td>1151</td>
<td>5533</td>
<td>838</td>
</tr>
<tr>
<td>ship12s</td>
<td>1151</td>
<td>2869</td>
<td>466</td>
</tr>
<tr>
<td>sierra</td>
<td>1227</td>
<td>2735</td>
<td>1222</td>
</tr>
<tr>
<td>standata</td>
<td>359</td>
<td>1382</td>
<td>359</td>
</tr>
<tr>
<td>stocfor2</td>
<td>2157</td>
<td>3045</td>
<td>2157</td>
</tr>
<tr>
<td>stocfor3</td>
<td>16675</td>
<td>23541</td>
<td>16675</td>
</tr>
<tr>
<td>tuff</td>
<td>333</td>
<td>630</td>
<td>292</td>
</tr>
<tr>
<td>woodw</td>
<td>1098</td>
<td>8418</td>
<td>1098</td>
</tr>
</tbody>
</table>
Table 5.3  Efficiency of multiple correction and presolve techniques on some large Netlib problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>LIPSOL v0.41</th>
<th>Multiple Corrections</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iter</td>
<td>time</td>
</tr>
<tr>
<td>25fv47</td>
<td>25</td>
<td>35.53</td>
</tr>
<tr>
<td>80bau3b</td>
<td>40</td>
<td>181.73</td>
</tr>
<tr>
<td>bnl2</td>
<td>31</td>
<td>90.23</td>
</tr>
<tr>
<td>cycle</td>
<td>24</td>
<td>104.04</td>
</tr>
<tr>
<td>d2q06c</td>
<td>32</td>
<td>223.40</td>
</tr>
<tr>
<td>degen3</td>
<td>37</td>
<td>214.03</td>
</tr>
<tr>
<td>df001</td>
<td>83</td>
<td>16619.37</td>
</tr>
<tr>
<td>fitlp</td>
<td>16</td>
<td>154.89</td>
</tr>
<tr>
<td>greenbea</td>
<td>43</td>
<td>193.52</td>
</tr>
<tr>
<td>greenbeb</td>
<td>38</td>
<td>206.96</td>
</tr>
<tr>
<td>maros-r7</td>
<td>15</td>
<td>1126.85</td>
</tr>
<tr>
<td>nesm</td>
<td>33</td>
<td>65.01</td>
</tr>
<tr>
<td>pilot87</td>
<td>38</td>
<td>1355.55</td>
</tr>
<tr>
<td>pilotja</td>
<td>31</td>
<td>83.64</td>
</tr>
<tr>
<td>pilots</td>
<td>31</td>
<td>407.21</td>
</tr>
<tr>
<td>pilotwe</td>
<td>37</td>
<td>47.10</td>
</tr>
<tr>
<td>seba</td>
<td>22</td>
<td>56.79</td>
</tr>
<tr>
<td>woodw</td>
<td>28</td>
<td>111.97</td>
</tr>
</tbody>
</table>
Table 5.4  Size reductions after presolve on large problems other than Netlib

<table>
<thead>
<tr>
<th>Problem</th>
<th>Original size</th>
<th>After LIPSOL presolve</th>
<th>After our presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m</td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td>co9</td>
<td>10789</td>
<td>22924</td>
<td>10537</td>
</tr>
<tr>
<td>lp22</td>
<td>2958</td>
<td>10392</td>
<td>2958</td>
</tr>
<tr>
<td>mod2</td>
<td>34774</td>
<td>66409</td>
<td>34320</td>
</tr>
<tr>
<td>world</td>
<td>34506</td>
<td>67147</td>
<td>34081</td>
</tr>
<tr>
<td>cre-b</td>
<td>9648</td>
<td>77137</td>
<td>7240</td>
</tr>
<tr>
<td>cre-d</td>
<td>8926</td>
<td>73948</td>
<td>6476</td>
</tr>
<tr>
<td>ken-13</td>
<td>28632</td>
<td>42659</td>
<td>23393</td>
</tr>
<tr>
<td>osa-30</td>
<td>4350</td>
<td>104374</td>
<td>4350</td>
</tr>
<tr>
<td>pds-10</td>
<td>16558</td>
<td>49932</td>
<td>16239</td>
</tr>
</tbody>
</table>

Table 5.5  Efficiency of multiple correction and presolve techniques on large problems other than Netlib

<table>
<thead>
<tr>
<th>Problem</th>
<th>LIPSOL v0.41</th>
<th>Multiple Corrections</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>iter</td>
<td>time</td>
</tr>
<tr>
<td>co9</td>
<td>48</td>
<td>1360.07</td>
</tr>
<tr>
<td>lp22</td>
<td>37</td>
<td>2514.30</td>
</tr>
<tr>
<td>mod2</td>
<td>83</td>
<td>8749.19</td>
</tr>
<tr>
<td>world</td>
<td>87</td>
<td>9031.47</td>
</tr>
<tr>
<td>cre-b</td>
<td>42</td>
<td>3362.87</td>
</tr>
<tr>
<td>cre-d</td>
<td>38</td>
<td>2612.70</td>
</tr>
<tr>
<td>ken-13</td>
<td>27</td>
<td>2103.76</td>
</tr>
<tr>
<td>osa-30</td>
<td>36</td>
<td>2274.19</td>
</tr>
<tr>
<td>pds-10</td>
<td>52</td>
<td>8256.06</td>
</tr>
</tbody>
</table>
Chapter 6

Conclusions

In this thesis, we presented the multiple correction technique proposed by Gondzio [5], described in detail our implementation and, in particular, our modifications on the starting and stopping criteria for the multiple correction process.

Our implementation was carried out under LIPSOL v0.41 environment (see [24]). The computational experiments were performed on the entire Netlib set of test problems as well as some large-scale problems from real-world applications.

We also discussed the presolve technique of finding and eliminating implied free variables appearing in singleton columns. Numerical results have shown that our implementation of this technique usually reduce the problem size further and very often considerably. The problem-size reduction contributes significantly to the improvements on the efficiency of linear programming solvers.

Numerical results have shown that our implementation of the enhanced presolve and multiple correction techniques has clearly improved the efficiency of the LIPSOL package, leading to CPU-time savings on most nontrivial problems. More importantly, these savings have shown a tendency to increase as the problems get more difficult and larger.
Bibliography


[8] P. F. Hung, Y. Ye. An Asymptotical $O(\sqrt{n}L)$-Iteration Path-Following Linear


[16] S. Mehrotra. Higher Order Methods and their Performance Technical Report 90-16R1, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, USA.


