THE INTERIOR POINT METHOD AND ITS APPLICATION TO THE FRACTIONAL (g,f)-FACTOR PROBLEM

By

Ping Shi

B. Eng., Beijing University of Posts and Telecommunications, 1985
M. Eng., Beijing University of Posts and Telecommunications, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF MATHEMATICS
INSTITUTE OF APPLIED MATHEMATICS

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 1996

© Ping Shi, 1996
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Mathematics
Institute of Applied Mathematics
The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1Z1

Date: Aug. 26, 1996
Abstract

This thesis contains six chapters. From Chapter 1 to Chapter 5, we give an exposition of interior-point methods (IPM). In Chapter 1 a brief survey of the recent development in interior-point methods is given. Chapter 2 gives a detailed description on the history of the primal-dual path-following methods for linear programming. Chapter 3 introduces several predictor-corrector methods. This is an efficient and powerful class of interior-point methods and a lot of attention has been paid during recent years. Chapter 4 discusses the various efficient linear algebraic methods which are used for solving the Newton equations system. Practically, most of the computational effort in implementations of interior-point methods is taken up in solving the Newton equations system. Therefore an efficient method for solving the Newton equations system can make the whole algorithm very efficient. Chapter 5 talks about the presolving procedure which is used to reduce the size of the LP problem and to make the system sparser before passing the problems to optimization. The presolve procedure is a very efficient adjunct for any LP solver. In chapter 6, we apply the interior-point method to the fractional \((g,f)\)-factor problem. We first exploit the special structure of the fractional \((g,f)\)-factor problem and propose two IPM algorithms to take advantage of the special structure of the fractional \((g,f)\)-factor problem. In each of the two algorithms, we use a different linear programming model for the fractional \((g,f)\)-factor problem in order to make the corresponding algorithm more efficient. Then, we discuss three starting point approaches and the strategy for creating our test problems. Last, we use the high order predictor-corrector primal-dual path-following code HOPDM[45] to test the three starting point approaches and from the computational results we get the conclusion that a fractional \((g,f)\)-factor problem-specific starting point can save iterations of the IPM and make the IPM more efficient for solving fractional \((g,f)\)-factor problem.
Table of Contents

Abstract ii
List of Tables vi
List of Figures vii
Acknowledgement viii

1 Introduction to Interior Point Methods 1

2 Path-following Methods for Linear Programming 7
   2.1 Preliminaries .................................................. 7
       2.1.1 Newton-Raphson’s method for solving nonlinear equations 11
       2.1.2 The Karush-Kuhn-Tucker Conditions .................. 12
   2.2 The Affine Scaling Direction ................................ 13
       2.2.1 The Primal Affine Scaling Direction ................. 13
       2.2.2 The Dual Affine Scaling Direction .................. 14
   2.3 The Centering Direction .................................... 15
       2.3.1 The Primal Centering Direction .................... 15
       2.3.2 The Dual Centering Direction ...................... 16
   2.4 The Primal Path-following Method ......................... 16
   2.5 The Dual Path-following Method ........................... 19
   2.6 The Primal-Dual Path-following Method .................. 19
   2.7 The Primal-Dual Infeasible-Interior-Point Method ....... 25
6.1.2 Staircase Structure .................................................. 71
6.1.3 The Fractional \((g,f)\)-factor Problem .............................. 74
6.2 The Preconditioned Conjugate Gradient Algorithm .............. 78
6.3 Computational Implementation ........................................ 85
  6.3.1 Starting Point .................................................. 86
  6.3.2 Test Problems .................................................. 89
  6.3.3 Computational Results ........................................ 90

Bibliography .............................................................. 95
List of Tables

6.1 The preconditioned conjugate gradient algorithm .......................... 82
6.2 Original problem size ...................................................... 91
6.3 Iteration counts for approach 2 .......................................... 93
6.4 Iteration counts for different starting point approaches ................ 94
List of Figures

6.1 The layered graph $G$ ....................................................... 76
6.2 The staircase matrix $\hat{A}$ of the layered graph $G$ .......................... 77
Acknowledgement

I would like to express my appreciation to my supervisor, Dr. R. P. Anstee, for his guidance and criticism throughout the preparation of this work. I would also like to thank Dr. S. T. McCormick for his helpful suggestions, comments and careful reading of the draft.
Chapter 1

Introduction to Interior Point Methods

Interior point methods for mathematical programming were present at least since the sixties. In 1967, Dikin[25] introduced an affine-scaling method for linear and quadratic programming. Then Fiacco and McCormick[31] developed the IPM as a tool for nonlinear programming. Although Fiacco and McCormick noted that their proposed methods could be applied in full measure to linear programming, neither they nor any other researchers at that time seriously proposed that interior point method would provide a viable alternative to the simplex method for actually solving linear programming problems. This is for two reasons. First, due to the storage limitations, the size of the problems solved at that time did not exceed a couple of hundred rows and columns, and for such sizes the simplex method is practically unbeatable. Secondly, there was no numerical linear algebra for solving large, sparse linear equation systems at that time. (This was done in the '70s and '80s by George and Liu [34].) Therefore the orthogonal projections in IPM were very expensive. IPM needs significantly more memory than the simplex method which was an unacceptable requirement at that time.

Although the simplex method is efficient and elegant, it does not possess a property that became more and more important in the last two decades: polynomial complexity. It was proved that the simplex method has an exponential worst-case complexity[61].

In 1978 Khachiyan[60] provided the first polynomial worst-case complexity algorithm for linear programming by showing how to adapt the ellipsoid method for convex optimization provided by Shor[94] and Yudin and Nemirovskii[116] to linear programming which requires $O(n^4L)$ time, where $L$ is the bit size of the problem. This method has significant theoretical implications for mathematical programming but is not competitive with the simplex method in
In 1984, Karmarkar\cite{57} published his projective algorithm, which was proven to require $O(nL)$ iterations and $O(n^{3.5}L)$ time overall, lower than Khachian's. It was also announced that it was more efficient than the simplex method in practice. Karmarkar's algorithm is essentially different from the simplex method in that it moves through the relative interior of the feasible set instead of following a sequence of vertices as does the simplex method. Karmarkar's work opened a new research field, the field of interior point methods, which has yielded many theoretical and practical achievements.

During the last eleven years a substantial number of contributions have been made in this field, both theoretical and practical. Inspired by Karmarkar's publication many other polynomial methods were proposed by researchers, sometimes containing new ideas, sometimes containing only slight modifications or embellishments. Numerical results have shown that some interior point methods can solve large linear programming problems faster than the simplex method.

For a complete survey, the reader can consult the following excellent articles or books by Andersen, Gondzio, Mészáros and Xu\cite{4}, Goldfarb and Todd\cite{40}, Gonzaga\cite{50}, Hertog and Roos\cite{51}, Lustig, Marsten and Shanno\cite{68}, Todd\cite{98, 99}, Wright\cite{108, 109} and Ye\cite{114}.

I also refer the reader to the following web sites:

1. Linear Programming - Frequently Asked Questions List:
   

2. Interior-Point Methods Online:
   

All variations of IPM can be classified into four main categories:

1. Projective potential reduction methods;
2. Affine scaling methods;
3. Path-following methods;

4. Affine potential reduction methods.

Karmarkar’s algorithm is the original projective potential reduction method. It was studied and improved by many researchers: simplification of the analysis, studies of limiting behavior, removing the initial assumptions, etc. From the practical point of view the early results, e.g. [100], were disappointing. Later more efficient algorithms were developed using better formulas of the linear programming problem. Anstreicher [10], [11], Gay [33], De Ghellinck and Vial [36], Gonzaga [48], Yamashita [110], Ye [112], Ye and Kojima [115] studied the projective potential reduction method. Actually, these earlier methods provide useful perspective on later path-following approaches and the projective algorithms soon became off the main stream of research.

The practical merits of IPM became clear when several researchers proposed and implemented the so-called affine scaling method. In each iteration of the projective potential reduction method a projective transformation is carried out to ‘center’ the current iterate. Instead of doing this projective transformation, Vanderbei et al. [106] and Barnes [13] proposed to do a simple affine scaling to center the current iterate. It turned out that this natural simplification of Karmarkar’s method was already proposed by Dikin [25] in 1967. Cavalier and Soyster [20] also studied this method. Several implementations of the primal and dual affine scaling methods (e.g. [2], [14] and [106]) showed promising results.

The affine scaling methods have no known polynomial time complexity and, in fact, Megiddo and Shub [75] have indicated that these methods may require an exponential number of iterations if they are started “close” to the boundary of the feasible region. Also, Vanderbei [103] has pointed out that these methods can make it difficult to recover dual solutions and prove optimality when there is degeneracy. However, these affine method do work well in practice; see Chen [22] for the primal affine method, and Adler et al. [2], [1], Marsten [71], and Monma and Morton [80] for the dual affine method. Monterio et al. [83] have obtained a polynomial time bound for a primal-dual affine method.
Chapter 1. Introduction to Interior Point Methods

Most recent research in the IPM has concentrated on path-following methods. This class of methods explicitly restricts the iterates to a neighborhood of the central path and follows the path to a solution of the linear programming problem. Both the classical logarithmic barrier method and the center method are called path-following methods. We now describe the main results obtained for this class of methods.

Gill, Murray, Saunders, Tomlin and Wright[38] first built the bridge between Karmarkar’s method and the logarithmic barrier method for linear programming. They showed that the search directions used in Karmarkar’s method and in the logarithmic barrier method were closely related. They also implemented the logarithmic barrier method and obtained promising results. However, they did not prove polynomiality as Karmarkar did. The logarithmic barrier function approach is usually attributed to Frisch[32] and is formally studied in Fiacco and McCormick[31] in the context of nonlinear optimization.

The presentation of a continuous trajectory was first proposed by Karmarkar and extensively studied by Bayer and Lagarias[16], Megiddo[74], and Megiddo and Shub[75]. Megiddo[73] relates these central paths to the classical barrier path in a primal-dual setting. Kojima, Mizuno and Yoshise[62] used this framework to present a primal-dual path-following method which requires $O(nL)$ iterations and overall $O(n^4L)$ time. Monteiro and Adler[81] improve upon this result by using ideas of Gonzaga[46] and Karmarkar[57] to obtain a primal-dual short-step path-following method which require $O(n^{0.5}L)$ iterations and overall $O(n^3L)$ time.

Roos and Vial[90] and Gonzaga[49] independently developed a natural and more practical version of the logarithmic barrier method for linear programming, the long step path-following method. In this method the barrier parameter may be reduced by an arbitrary factor between 0 and 1. This means that the iterates need not lie close to the central path and hence long steps can be taken. They also proved that this method has an $O(nL)$ iteration bound. Hertog[53] gave a simple analysis for the logarithmic barrier method and showed that the overall complexity can be reduced by a factor $\sqrt{n}$, using safeguarded line searches.
Better results are obtained by predictor-corrector methods. The predictor-corrector methods (adaptive primal-dual algorithms) were developed by Mizuno, Todd and Ye[79]. The more practical predictor-corrector algorithms were proposed by Mehrotra[76], Carpenter, Lustig, Mulvey and Shanno[19] and Zhang and Zhang[117]. Nowadays, almost all the existing interior-point codes for general-purpose linear programming problems are based on Mehrotra’s predictor-corrector algorithm. These algorithms represent current state-of-the-art IPM.

The center method was proposed by Huard[55] in 1967. Compared to the logarithmic barrier method, the center method has received less attention.

Renegar[87] proved in 1987 that a very special version of the center method for linear programming is polynomial. In fact, this was the first polynomial path-following method with an $O(\sqrt{nL})$ iteration bound. In this method very short steps are taken. Although this method for linear programming has an $O(\sqrt{nL})$ iteration bound, which is a factor $\sqrt{n}$ better than the projective potential reduction method, it is very inefficient in practice. Later on Vaidya[102] reduced the total complexity bound to $O(n^3L)$, using approximate solutions and rank-one updates. Hertog[52] developed a natural and more practical version of the center method for linear programming, in which the relaxation factor may be any number between 0 and 1, and line searches are allowed.

The affine potential reduction algorithm was developed by Gonzaga[47]. His algorithm has iteration complexity $O(nL)$. The primal-dual potential function and algorithm were analyzed by Anstreicher and Bosch[12] and Ye[113]. These algorithms possess $O(\sqrt{nL})$ iteration complexity. Ye proved that by doing projected gradient steps (after affine scaling) a constant reduction for the potential function value can be obtained. His method was not a symmetric primal-dual method. Kojima et al.[63] developed a symmetric primal-dual version of this method for linear complementarity problems.

The potential reduction methods take steps of the same form as do path-following methods, but they do not explicitly follow the path and can be motivated independently of the path.
They use a logarithmic potential function to measure the worth of each point in the primal-dual relative interior of the feasible set and aim to achieve a certain fixed reduction in this function at each iteration.
Chapter 2

Path-following Methods for Linear Programming

2.1 Preliminaries

In this section we will give notation and some relevant theory for linear programming problems and IPM. We also present a conceptual path-following algorithm.

We consider the linear programming problem in the standard form

\[
\begin{align*}
\text{minimize} & \quad c^T x, \\
\text{subject to} & \quad Ax = b, \\
& \quad x \geq 0,
\end{align*}
\]

(2.1)

where \( c, x \in \mathbb{R}^n \), \( b \in \mathbb{R}^m \), \( A \in \mathbb{R}^{m \times n} \), \( n > m \).

The corresponding dual problem is

\[
\begin{align*}
\text{maximize} & \quad b^T y, \\
\text{subject to} & \quad A^T y \leq c.
\end{align*}
\]

(2.2)

It can be changed to the following form

\[
\begin{align*}
\text{maximize} & \quad b^T y, \\
\text{subject to} & \quad A^T y + z = c, \\
& \quad z \geq 0,
\end{align*}
\]

(2.3)

where \( y \in \mathbb{R}^m \), \( z \in \mathbb{R}^n \).

With respect to other forms of linear programs, an analogous analysis can be carried out. The linear problem with upper bounded variables will be discussed later.

We assume that the primal feasible region

\[
\mathcal{F}_p = \{ x \in \mathbb{R}_+^n : Ax = b \}, \quad \text{where } \mathbb{R}_+ = \{ x \in \mathbb{R} : x \geq 0 \},
\]

(2.4)
is bounded and has a nonempty relative interior given by

$$\mathcal{F}_P = \{ x \in \mathcal{R}_+^n : Ax = b \}, \quad \text{where} \quad \mathcal{R}_+^n = \{ x \in \mathcal{R} : x > 0 \}. \quad (2.5)$$

The corresponding dual feasible region is

$$\mathcal{F}_D = \{ (y, z) \in \mathcal{R}^m \times \mathcal{R}_+^n : A^T y + z = c \}, \quad (2.6)$$

and the dual relative interior is

$$\mathcal{F}_D = \{ (y, z) \in \mathcal{R}^m \times \mathcal{R}_+^n : A^T y + z = c \}. \quad (2.7)$$

We also use $\mathcal{F} = \mathcal{F}_P \times \mathcal{F}_D$ and $\mathcal{F} = \mathcal{F}_P \times \mathcal{F}_D$ to represent primal-dual feasible region and its relative interior, respectively.

The duality gap is $c^T x - b^T y$ and the complementarity gap is $x^T z$. If $(x, y, z)$ is primal and dual feasible then the duality and complementarity gap are identical.

Our concern starts from the fact that IPM performs well by avoiding the boundary of the feasible set. It does this with the help of a classical resource first used in optimization by Frisch[32] in 1955: the logarithmic barrier function

$$x \in \mathcal{R}^n, x > 0 \implies P(x) = -\sum_{j=1}^n \log x_j = -\log \prod_{j=1}^n x_j. \quad (2.8)$$

This function grows indefinitely near the boundary of the feasible set $\mathcal{F}$, and can be used as a penalty attached to those points. Combining $P(x)$ with the objective makes points near the boundary expensive, and forces any minimization algorithm to avoid them. The unique minimizer of $P(x)$ in $\mathcal{F}$ is the analytic center of $\mathcal{F}$, and it coincides with the point that maximizes the product of the variables in $\mathcal{F}$. Newton-Raphson's method can be adapted to determine a center with a prescribed precision with excellent theoretical and practical performance.

The main idea behind all interior point method is that costs should try to be decreased and simultaneously move away from the boundary. Thus we examine combinations of cost and barrier function, in a traditional construction known as the internal penalized function:

$$\mu \in \mathcal{R}, x \in \mathcal{R}_+^n \implies f(x, \mu) = c^T x - \mu \sum_{j=1}^n \log x_j = c^T x + \mu P(x). \quad (2.9)$$
where $\mu$ is a barrier parameter. For a given value of $\mu$, (2.1) becomes

\[
\begin{align*}
\text{minimize} & \quad f(x, \mu), \\
\text{subject to} & \quad Ax = b.
\end{align*}
\]

Similarly, the internal penalized function for (2.3) is

\[
\mu \in \mathcal{R}, z \in \mathcal{R}_{++}^n \quad \Rightarrow \quad g(y, z, \mu) = b^T y + \mu \sum_{j=1}^{n} \log z_j,
\]

thus transforming (2.3) to

\[
\begin{align*}
\text{maximize} & \quad g(y, z, \mu), \\
\text{subject to} & \quad A^T y + z = c.
\end{align*}
\]

The internal penalized function was extensively studied by Fiacco and McCormick in their book[31], and described since then in all nonlinear programming textbooks. Now associate to each value of the parameter $\mu$ a primal central point $x(\mu)$ uniquely defined by

\[
x(\mu) = \arg\min_{x \in \mathcal{F}_P} f(x, \mu).
\]

The curve $(x(\mu) : \mu \in \mathcal{R})$ is the primal central path for the linear programming problem (2.1).

In the same way we can define a dual central point $(y(\mu), z(\mu))$ as

\[
(y(\mu), z(\mu)) = \arg\min_{y \in \mathcal{F}_D, z \in \mathcal{F}_D} g(y, z, \mu).
\]

The curve $\{(y(\mu), z(\mu)) : \mu \in \mathcal{R}\}$ is the dual central path for the linear programming problem (2.3).

Assume that $\mathcal{F} \neq \emptyset$, i.e. both $\mathcal{F}_P \neq \emptyset$ and $\mathcal{F}_D \neq \emptyset$. We define the primal-dual central path as follows

\[
\Gamma = \{(x, y, z) \in \mathcal{F} : Xz = \mu e \quad \text{where} \quad \mu = \frac{x^T z}{n}\},
\]

where $X = \text{diag}(x_1, x_2, \ldots, x_n)$, and $e$ denotes the vector of all ones that is $e = (1, 1, \ldots, 1)^T$.

Ye[114] gives the following theorem:
Theorem 2.1.1 Let \((x(\mu), y(\mu), z(\mu)) \in \Gamma(\mu)\) then

1. The central pair \((x(\mu), z(\mu))\) is bounded where \(0 < \mu \leq \mu^0\) for any given \(\mu^0 > 0\).
2. For \(0 < \mu' \leq \mu\), \(c^T x(\mu') \leq c^T x(\mu)\) and \(b^T y(\mu') \geq b^T y(\mu)\).
3. \((x(\mu), y(\mu), z(\mu))\) converges to an optimal solution pair for (2.1) and (2.3) as \(\mu \to 0\).

The theorem shows the primal-dual central path is well defined and its limit point is an optimal solution to (2.1) and (2.3).

Path-following algorithms follow the central path. Let \((\Gamma(\mu) : \mu \in \mathcal{R})\) be a parameterization of the central path. All path-following algorithms follow the model below.

Algorithm 2.1. Conceptual path-following: given an initial point \((x^0, y^0, z^0) \in \tilde{\mathcal{F}}, \mu^0 \in \mathcal{R}\), with \((x^0, y^0, z^0) = \Gamma(\mu^0)\).

\[ k := 0. \]

**REPEAT**

Choose \(\mu_{k+1} < \mu_k\).

Call an internal minimization algorithm to find \((x^{k+1}, y^{k+1}, z^{k+1}) = \Gamma(\mu_{k+1})\).

\[ k := k + 1. \]

UNTIL convergence.

The model simply generates a sequence of independent central points \(\Gamma(\mu_k), (k = 1, \ldots)\). The parameterization and the updating rule characterize different algorithms. In the internal minimization algorithm, Newton-Raphson’s algorithm can be used to find the central point for corresponding \(\mu_k\). We will discuss the choices of \(\mu\) later. Exact central points cannot be exactly computed in finite time. The practical algorithms generate points in a neighborhood of the central path converging towards the optimal solution.

Before we present the detailed description about the path-following algorithm, let us review some basic concepts which we will use in the following sections.
2.1.1 Newton-Raphson’s method for solving nonlinear equations

For

\[ F(x) = 0, \quad (2.16) \]

where \( x = (x_1, x_2, \ldots, x_m)^T \), \( F(x) = (f_1(x), f_2(x), \ldots, f_m(x))^T \) and \( f_i(x), (i = 1, 2, \ldots, m) \) are differentiable at \( D \subset \mathbb{R}^m \).

The Taylor expansion of \( f_i(x), (i = 1, 2, \ldots, m) \) at \( x^k \) is

\[ f_i(x) \approx f_i(x^k) + \sum_{j=1}^m \frac{\partial f_i(x^k)}{\partial x_j} \Delta x_j, \quad (2.17) \]

where \( \Delta x_j^k = x_j - x_j^k, \quad (j = 1, 2, \ldots, m) \).

Combine (2.16) and (2.17) we get

\[ \sum_{j=1}^m \frac{\partial f_i(x^k)}{\partial x_j} \Delta x_j^k = -f_i(x^k), \quad (i = 1, 2, \ldots, m), \quad (2.18) \]

or in vector form

\[ F'(x^k) \Delta x^k = -F(x^k), \quad (2.19) \]

where

\[ F'(x^k) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_m} \end{pmatrix}, \]

and

\[ \Delta x^k = (\Delta x_1^k, \ldots, \Delta x_m^k)^T, \]

and \( F'(x) \) is Jacobi matrix of \( F(x) \). Then

\[ x^{k+1} = x^k + \Delta x^k. \quad (2.20) \]

Newton’s method can be applied to unconstrained minimization problems. To minimize \( L(x) \), we take \( F(x) = L'(x) \) and use Newton’s method to search for a zero of \( F(x) \). If \( L(x) \) is
a real valued function of \( m \) variables, a local minimum of \( L \) will satisfy the following system of \( m \) equations in \( m \) variables:

\[
\frac{\partial L(x)}{\partial x_1} = 0, \\
\vdots \\
\frac{\partial L(x)}{\partial x_m} = 0.
\]

So the Newton iteration (2.19) becomes

\[
\nabla^2 L(x^k) \Delta x^k = -\nabla L(x^k),
\]

where \( \nabla L \) is the gradient of \( L \) and \( \nabla^2 L \) is the Hessian of \( L \). If \( L \) is convex, then any local minimizer is also a global minimizer. If \( x^* \) is a local minimizer of \( L(x) \), i.e. \( \nabla L(x^*) = 0 \), and if \( \nabla^2 L(x^*) \) has full rank, then Newton’s method will converge to \( x^* \).

### 2.1.2 The Karush-Kuhn-Tucker Conditions

We consider the NLP of the following form:

\[
\text{NLP: minimize } \{ f(x) : g_i(x) \geq 0, \text{ for } i = 1, \ldots, m \}. \tag{2.22}
\]

The Karush-Kuhn-Tucker Conditions for NLP are

1. \( \nabla f(x) + \sum_{i=1}^{m} y_i \nabla g_i(x) = 0. \)
2. \( g_i(x) \geq 0 \) for \( i = 1, \ldots, m; \) and \( y \leq 0. \)
3. \( y_i g_i(x) = 0 \) for \( i = 1, \ldots, m. \)

If all the constraints in NLP are equations, the NLP becomes the problem of Lagrange multipliers (PLM). PLM can be stated as follows:

\[
\text{PLM: minimize } \{ f(x) : g_i(x) = 0, \text{ for } i = 1, \ldots, m \}. \tag{2.24}
\]

The Lagrange function is

\[
L(x, y) = f(x) + \sum_{i=1}^{m} y_i g_i(x). \tag{2.25}
\]
Chapter 2. Path-following Methods for Linear Programming

The Optimality Conditions for PLM

A vector \( x \in \mathbb{R}^n \) is optimal for PLM only if there exists a vector \( y \in \mathbb{R}^m \) such that

1. \[ \nabla_x L(x, y) = \nabla f(x) + \sum_{i=1}^{m} y_i \nabla g_i(x) = 0. \tag{2.26} \]

2. \( g_i(x) = 0 \) for \( i = 1, \ldots, m \); and \( y \) is sign free.

The Lagrange function (2.25) transforms a constrained optimization problem with equality constraints into an unconstrained optimization problem.

To solve (2.24) from the Lagrange function (2.25), we minimize the unconstrained function \( L(x, y) \) by solving the system of \( (n + m) \) equations in \( (n + m) \) variables:

\[
\frac{\partial L}{\partial x_j} = \frac{\partial f(x)}{\partial x_j} - \sum_{i=1}^{m} y_i \frac{\partial g_i(x)}{\partial x_j} = 0, \quad \text{for } j = 1, \ldots, n, \\
\frac{\partial L}{\partial y_i} = -g_i(x) = 0, \quad \text{for } i = 1, \ldots, m.
\]

These equations can be solved by Newton's method.

2.2 The Affine Scaling Direction

2.2.1 The Primal Affine Scaling Direction

For \( x^k \in \mathcal{F}_P \), the affine-scaling algorithm generates an improved feasible solution by scaling \( x^k \) to \( e \), the vector of ones, and making a step in the projected steepest-descent direction for the transformed objective function in the transformed space.

A scaling transformation is a change of variables \( x = D\tilde{x} \), where \( D \) is a positive diagonal matrix. Give a point \( x^k \in \mathcal{F}_P \), scaling about \( x^k \) is the scaling transformation \( x = X_k\tilde{x} \), where \( X_k = \text{diag}(x^k_1, \ldots, x^k_n) \).

The scaled version of (2.1) is then

\[
\begin{align*}
\text{minimize} & \quad \tilde{c}^T \tilde{x} \\
\text{subject to} & \quad \tilde{A}\tilde{x} = b, \\
& \quad \tilde{x} \geq 0,
\end{align*}
\tag{2.27}
\]
where \( \tilde{A} = AX_k \) and \( \tilde{c} = X_k c \) are obtained by substitution of \( x = X_k \hat{x} \) in (2.1). The feasible solution \( x = x^k \) of (2.1) corresponds to the feasible solution \( \hat{x} = e \) of (2.27).

Dikin's[25] primal affine-scaling algorithm takes a step from \( \hat{x} = e \) in the direction of the negative projected gradient. This direction is \(-c\) projected onto the null space of \( \tilde{A} \)

\[
\hat{p}_{a f f} = -\tilde{P}\hat{c} = -P_{AX_k}X_k c, \quad (2.28)
\]

where \( P_{AX_k} = I - X_k A^T (AX_k^2 A^T)^{-1} AX_k \).

The affine-scaling direction in the original space corresponds to moving from \( x^k \) in the direction

\[
p_{a f f} = -X_k \tilde{P}\hat{c} = -X_k P_{AX_k}X_k c. \quad (2.29)
\]

This primal affine-scaling direction is used in the primal affine scaling methods[25, 2, 13, 106].

### 2.2.2 The Dual Affine Scaling Direction

Similar to the primal affine scaling direction, the scaled version of (2.3) is

\[
\begin{align*}
\text{maximize} & \quad b^T y, \\
\text{subject to} & \quad Z_k^{-1} A^T y + \hat{z} = Z_k^{-1} c, \\
& \quad \hat{z} \geq 0.
\end{align*}
\]

(2.30)

Gonzaga[48] gave the following Lemma which transforms the dual problem to the primal problem.

**Lemma 2.2.1** Formulation (2.30) is equivalent to

\[
\begin{align*}
\text{minimize} & \quad \bar{b}^T \hat{z} - \bar{b}_0, \\
\text{subject to} & \quad \bar{A}^T \hat{z} = \bar{c}, \\
& \quad \hat{z} \geq 0.
\end{align*}
\]

(2.31)

where

\[
\bar{A}^T = P_{AZ^{-1} \cdot}
\]
Moreover, given \( h \in \mathbb{R}^n \), consider the direction \( d_z = P_A h \), then \( h \) is equivalent to the direction \( d \) for (2.3) given by

\[
d = -(AZ^{-2}A^T)^{-1}AZ^{-1}h.
\]

(2.32)

The dual affine scaling direction is the projected steepest-descent direction for problem (2.31). Use lemma 2.2.1 we get that

\[
d_{aff} = (AZ^{-2}A^T)^{-1}AZ^{-1}b = (AZ^{-2}A^T)^{-1}b.
\]

(2.33)

This direction is used in dual affine scaling methods.

2.3 The Centering Direction

2.3.1 The Primal Centering Direction

IPMs try to avoid the boundary of feasible region \( \mathcal{F} \). A good way of avoiding the boundary is obtained by defining a center for the polytope \( \mathcal{F} \), and by simultaneous consideration of two objectives: reducing costs and centering. We now ignore the cost and turn to the problem of finding a "center" for the polytope \( \mathcal{F} \). Most IPMs use the analytic center which was defined by Sonnevend[95]. The analytic center of \( \mathcal{F}_P \) is the unique point given by

\[
\chi = \arg\min_{x \in \mathcal{F}_P} P(x) = \arg\min_{x \in \mathcal{F}_P} \left(-\sum_{j=1}^{n} \log x_j \right).
\]

(2.34)

The analytic center is approached by solving the following problem

\[
\begin{align*}
\text{minimize} & \quad -\sum_{j=1}^{n} \log x_j, \\
\text{subject to} & \quad Ax = b.
\end{align*}
\]

(2.35)
The Lagrangian is
\[ L(x, y) = -\sum_{j=1}^{n} \log x_j - y^T (Ax - b). \] (2.36)
The first order conditions are
\[ \nabla_x L = -x^{-1}e - A^T y = 0, \]
\[ \nabla_y L = -Ax + b = 0. \] (2.37)
Solving these equations by Newton’s method, we get
\[ p_{cent} = \Delta x^k = X_k P_A X_k e. \] (2.38)
This is the primal centering direction.

2.3.2 The Dual Centering Direction
Similar to the primal centering direction, the dual centering direction is obtained by solving the following problem
\[
\text{minimize } -\sum_{j=1}^{n} \log z_j,
\]
subject to \( A^T y + z = c. \)

The Newton direction for this problem is
\[ d_{cent} = -(AZ^{-2}A^T)^{-1}AZ^{-1}e. \] (2.40)
This is the dual centering direction.

2.4 The Primal Path-following Method
The primal path-following method was first studied by Gill et al. [38]. They consider the linear programming problem in the standard form (2.1).
Replacing non-negativity of constraints in (2.1) with the primal logarithmic barrier penalty terms in the objective function, we get

\[
\begin{align*}
\text{minimize} & \quad c^T x - \mu \sum_{j=1}^{n} \ln x_j = f(x, \mu), \\
\text{subject to} & \quad Ax = b,
\end{align*}
\]

(2.41)

where \( \mu > 0 \) is the barrier penalty parameter. This technique is well known in the context of general constrained optimization problems. One solves the problem penalized by the logarithmic barrier function term for several values of the parameter \( \mu \), with \( \mu \) decreasing to zero, and the result is a sequence of feasible points converging to a solution of the original problem.

The objective function \( f(x, \mu) \) of problem (2.41) is a strictly convex function. This implies that this problem has at most one global minimum, and that this global minimum, if it exists, is completely characterized by the Karush-Kuhn-Tucker stationary conditions (first order conditions). The function \( f(x, \mu) \) achieves the minimal value in its domain (for fixed \( \mu \)) at a unique point, which is denoted by \( x(\mu) \), and called the \( \mu \)-center.

The Lagrangian for (2.41) is

\[
L(x, y, \mu) = c^T x - \mu \sum_{j=1}^{n} \ln x_j - y^T (Ax - b),
\]

(2.42)

and the first order conditions (KKT conditions) for (2.41) are

\[
\begin{align*}
\nabla_x L &= c - \mu X^{-1} e - A^T y = 0, \\
\nabla_y L &= -Ax + b = 0,
\end{align*}
\]

(2.43)

where \( y \) is the vector of Lagrangian multipliers associated with the equality constraints of problem (2.41), and \( X \) denotes the diagonal matrix whose diagonal elements are the variables \( x_j, 1 \leq j \leq n \), which is denoted by \( X_k \) when evaluated at the iterate \( x^k \) (i.e. \( X = diag(x_1, \ldots, x_n) \)).

Now, we apply one iteration of Newton's method to find an approximate solution to (2.43) for a fixed value of \( \mu \). We get the search direction

\[
\Delta x^k = -(1/\mu_k)X_k P_{AX_k} X_k e + X_k P_{AX_k} e = (1/\mu_k) p_{aff} + p_{cent},
\]

(2.44)
where
\[ P_{AX_k} = (I - X_kA^T(AX_k^2A^T)^{-1}AX_k). \] (2.45)

A new estimate optimal solution \( x^{k+1} \) is defined by
\[ x^{k+1} = x^k + \alpha_k \Delta x^k, \] (2.46)

where \( \alpha_k \) is respective step length in the primal spaces chosen to assure the positivity of the variable \( x \). Rather than making several Newton steps to converge to the central trajectory for a fixed value of \( \mu \), the algorithm reduces \( \mu \) by \( \mu_{k+1} = \rho \mu_k, 0 < \rho < 1 \), at each step, and the algorithm continues.

Gill et al. [38] and Gay [33] noted that Karmarkar's method is just a special case of primal logarithmic barrier methods.

We find that (2.44) consist of a centering term to keep away from the boundary, and an affine term that leads toward an optimal solution. When \( \mu_k \to 0 \) optimality dominates, whereas for large \( \mu_k \), the method proceeds toward the analytic center of the feasible region. Den Hertog and Roos [51] showed that most interior point methods have search directions that are linear combinations of these two vectors.

The choice of \( \alpha_k \) and \( \rho \) is important for the convergence of the algorithm. Gonzaga [46] shows a certain choice which updates the barrier penalty parameter \( \mu \) by short steps, forces the algorithm to trace the central path, so that all points generated are near the central path. This algorithm converges in \( O(\sqrt{n}L) \) iterations assuming the initial point \( (x^0, y^0) \) is close to the central path. The overall complexity is \( O(n^3 L) \). Roos and Vial [91] and Gonzaga [49] propose a long-step path-following method based on the same direction (2.44). In their algorithms, the barrier parameter \( \mu \) is reduced by a large factor if the iterate lies close to the central path. Their algorithms also converges in \( O(\sqrt{n}L) \) iterations for certain values of \( \rho \).
Chapter 2. Path-following Methods for Linear Programming

2.5 The Dual Path-following Method

Renegar proposed a special case of dual logarithmic barrier method in [87]. We consider the dual linear programming problem (2.2). We do not use the logarithmic barrier function (2.11). We eliminate the inequality in (2.2) by incorporating it into a logarithmic barrier term appended to the objective to obtain the following transformed problem

\[ \min_y b^T y + \mu \sum_{j=1}^{n} \ln(c_j - a_j^T y), \]  

(2.47)

where \( a_j \) is the \( j \)th column of the matrix \( A \). The first order conditions (KKT conditions) are

\[ b - \mu AZ^{-1} e = 0, \]  

(2.48)

where \( Z \) is the \( n \times n \) diagonal matrix with elements \( z_j = c_j - a_j^T y \). One step of Newton's method yields

\[ \Delta y = \frac{1}{\mu} (AZ^{-2} A^T)^{-1} b - (AZ^{-2} A^T)^{-1} AZ^{-1} e = \frac{1}{\mu} d_{aff} + d_{cent}, \]  

(2.49)

where the first term in (2.49) represents a step toward optimality and the second term is a centering step in the dual space.

2.6 The Primal-Dual Path-following Method

Most of the new interior-point methods can be seen as variants of the primal-dual logarithmic barrier method due to Megiddo [74] who used logarithmic barrier methods to solve the primal and dual problems simultaneously. Meggiddo's method was developed into a convergent primal-dual logarithmic barrier algorithm by Kojima et al. [62]. This method was shown by Monteiro and Adler [82] to require \( O(n^2 L) \) overall time and no interior point algorithm has been shown to have a better worst-case complexity bound. In practice the primal-dual algorithm outperforms the primal and dual barrier algorithms.

Because in Chapter 6 we will use the primal-dual method to solve the fractional \((g,f)\)-factor problem which has upper bounded variables, now we consider the primal problem in the
following form

\[
\begin{align*}
\text{minimize} & \quad c^T x, \\
\text{subject to} & \quad Ax = b, \\
& \quad 0 \leq x \leq u,
\end{align*}
\] (2.50)

where \( c, x, u \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}, n > m \). Some or all of the upper bounds \( u \) may be infinite. Adding slack variables to \( x \leq u \), we get

\[
\begin{align*}
\text{minimize} & \quad c^T x, \\
\text{subject to} & \quad Ax = b, \\
& \quad x + s = u, \\
& \quad x \geq 0, \\
& \quad s \geq 0,
\end{align*}
\] (2.51)

where \( c, x, s, u \in \mathbb{R}^n \), and \( b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}, n > m \).

The corresponding dual problem is

\[
\begin{align*}
\text{maximize} & \quad b^T y - u^T w, \\
\text{subject to} & \quad A^T y - w + z = c, \\
& \quad z \geq 0, \\
& \quad w \geq 0,
\end{align*}
\] (2.52)

where \( y \in \mathbb{R}^m \), and \( z, w \in \mathbb{R}^n \).

To derive the primal-dual algorithm, we eliminate the inequality constraints in (2.51) by incorporating them in a logarithmic barrier function as

\[
\begin{align*}
\text{minimize} & \quad c^T x - \mu \sum_{j=1}^n \ln x_j - \mu \sum_{j=1}^n \ln s_j, \\
\text{subject to} & \quad Ax = b, \\
& \quad x + s = u.
\end{align*}
\] (2.53)

The Lagrangian for (2.53) is then

\[
L(x, s, y, w, \mu) = c^T x - \mu \sum_{j=1}^n \ln x_j - \mu \sum_{j=1}^n \ln s_j - y^T (Ax - b) + w^T (x + s - u). 
\] (2.54)
Chapter 2. Path-following Methods for Linear Programming

The first order optimality conditions (KKT conditions) for (2.54) are

\[
\begin{align*}
Ax &= b, \\
x + s &= u, \\
AT y + \mu X^{-1}e - w &= c, \\
\mu S^{-1}e - w &= 0. \\
\end{align*}
\]

Introducing the dual slack variable vector \( z = \mu X^{-1}e \), the first order optimality conditions (2.55) give

\[
\begin{align*}
Ax &= b, \\
x + s &= u, \\
AT y + z - w &= c, \\
XZe &= \mu e, \\
SW e &= \mu e, \\
\end{align*}
\]

where \( X, S, Z, \) and \( W \) are diagonal matrices with the elements \( x_j, s_j, z_j \) and \( w_j \), respectively, \( e \) is the \( n \)-vector of all ones and \( \mu \) is a barrier parameter.

The set of solutions of (2.56) \((x(\mu), s(\mu))\) and \((y(\mu), z(\mu), w(\mu))\) defines the central path of the primal and dual problem, respectively. The central path is the set of points at which the product of each primal-dual variable pair is identical. For any primal and dual feasible solution \((x, s)\) and \((y, z, w)\) the quality of centrality is measured by

\[
\delta^2(x, s, y, z, w, \mu) = \sum_{j=1}^{n} \left( \frac{x_j z_j}{\mu} - \frac{\mu}{x_j z_j} \right)^2 + \sum_{j=1}^{n} \left( \frac{s_j w_j}{\mu} - \frac{\mu}{s_j w_j} \right)^2.
\]

If \( \delta(x, s, y, z, w, \mu) = 0 \) then \((x, s)\) and \((y, z, w)\) are on the central path. The smaller \( \delta \) is, the better the points are centered.

From (2.56) we find the complementarity gap is

\[
\begin{align*}
c^T x - b^T y + u^T w &= x^T z - x^T w + u^T w \\
&= x^T z + s^T w \\
&= x^T \mu X^{-1} e + s^T \mu S^{-1} e \\
&= n\mu + n\mu = 2n\mu.
\end{align*}
\]
Therefore when $\mu \to 0^+$ the complementarity gap go to 0. It proves $(x(\mu), s(\mu))$ and $(y(\mu), z(\mu), w(\mu))$ converge to the optimal solution for $\mu \to 0^+$. Let us observe that the first three equations of (2.56) are linear and force primal and dual feasibility of the solution. The last two equations are nonlinear and become the complementarity conditions for $\mu = 0$, which together with the feasibility constraints provides optimality of the solutions.

A single iteration of the basic primal-dual algorithm makes one step of Newton's method applied to the first order optimality conditions (2.56) with a given $\mu$ and then $\mu$ is decreased. The algorithm terminates when the complementarity gap is reduced below a predetermined tolerance.

Assuming that the current estimate of the primal and dual variables satisfies both primal and dual feasibility, one step of Newton's method is applied to (2.56) by solving the following system of linear equations

$$
\begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
I & 0 & I & 0 & 0 \\
0 & AT & 0 & I & -I \\
Z & 0 & 0 & X & 0 \\
0 & 0 & W & 0 & S
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta s \\
\Delta z \\
\Delta w
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
0 \\
0 \\
\mu e - XZe \\
\mu e - SWe
\end{bmatrix}
$$

Solving (2.59) we get the search directions

$$
\Delta y = (A\Theta A^T)^{-1} A\Theta \rho(\mu),
\Delta x = \Theta (A^T \Delta y - \rho(\mu)),
\Delta z = \mu X^{-1} e - Ze - X^{-1} Z \Delta x,
\Delta w = S^{-1} (\mu e - SW e + W \Delta x),
\Delta s = -\Delta x,
$$

(2.60)

where $\Theta = (S^{-1} W + X^{-1} Z)^{-1}$ and $\rho(\mu) = \mu (S^{-1} - X^{-1}) e - (W - Z) e$. A new approximation
The barrier parameter $\mu$ serves as a centering parameter, pulling the primal variables and dual slacks away from zero.

Let $\text{gap}(0)$ denote the current duality gap. From (2.58) we know the duality gap is

$$\text{gap}(0) = c^T x - b^T y + u^T w = z^T x + w^T s.$$  

(2.63)

Let $\alpha \leq \min\{\alpha_P, \alpha_D\}$. From (2.59) we know, after one iteration, the new duality gap is

$$\text{gap}(\alpha) = (z + \alpha \Delta z)^T (x + \alpha \Delta x) + (w + \alpha \Delta w)^T (s + \alpha \Delta s)$$

$$= z^T x + w^T s + \alpha (Z \Delta x + X \Delta z + W \Delta s + S \Delta w)^T e$$

$$= \text{gap}(0) + \alpha (2n \mu - \text{gap}(0)).$$

Thus the duality gap is reduced at each step as long as $2n \mu - \text{gap}(0) < 0$, i.e.

$$\mu < \frac{\text{gap}(0)}{2n} = \frac{c^T x - b^T y + u^T w}{2n}. \quad (2.64)$$
In the primal-dual algorithm, we choose \( \mu \) by the algorithm of McShane et al.[72] and Lustig[66], namely, for feasible \( x \) and \( y \),
\[
\mu = \frac{c^T x - b^T y + u^T w}{\phi(n)},
\]
and
\[
\phi(n) = \begin{cases} 
  n^2, & \text{if } n \leq 5000, \\
  n\sqrt{n}, & \text{if } n > 5000.
\end{cases}
\]

The algorithm is repeated until the relative duality gap satisfies
\[
\frac{c^T x - b^T y + u^T w}{1 + |b^T y - u^T w|} < \epsilon,
\]
for a user predetermined \( \epsilon \).

In comparing primal, dual, and primal-dual methods, we find that all search directions consist of a matrix of the form \( ADA^T \), where \( D \) is a diagonal matrix. The content of \( D \) varies, but the computational work does not.

There are several advantages for the primal-dual method. First, primal feasibility may be obtained well before optimality. For the dual affine method, a primal feasible solution cannot in general be recovered until optimality. Thus if a run must be stopped short of optimality, no primal feasible solution can be guaranteed to be available for purely dual methods. Second, under primal and dual degeneracy, when large steps are taken, primal methods cannot be guaranteed to yield dual optimal solutions at optimality, nor can dual methods yield primal optimal solution in general[115]. The primal-dual method yields both under any degeneracy conditions. Third, for primal feasible \((x, s)\) and dual feasible \((y, z, w)\), the exact current duality gap (2.67) is always known. Thus an excellent measure of how close the given solution is to the optimal is always available. Finally, the primal-dual method allows for separate step lengths in the primal and dual spaces as shown in (2.61). McShane, Monma, and Shanno[72] have proven that these separate step lengths significantly reduce the number of iterations to convergence.
2.7 The Primal-Dual Infeasible-Interior-Point Method

In the discussions so far, we have been assuming that a feasible solution \((x^0, s^0) \in \mathcal{F}_P\) of the primal problem and a feasible solution \((y^0, z^0, w^0) \in \mathcal{F}_D\) of the dual problem are available so that we can immediately start the algorithm. In general, it is necessary to provide a means for determining an \((x^0, s^0, y^0, z^0, w^0) \in \mathcal{F}_P \times \mathcal{F}_D\) as an initial point for the algorithm.

Lustig\cite{65} has used the following pair of primal and dual linear programs to find the solution of a problem for which an initial interior feasible point is not known.

The augmented primal system is

\[
\begin{align*}
\text{minimize} & \quad c^T x + c_a x_a, \\
\text{subject to} & \quad A x + d_P x_a = b, \\
& \quad x + s = u, \\
& \quad d_P^T x + x_b = b_a, \\
& \quad x, s, x_a, x_b \geq 0.
\end{align*}
\] (2.68)

The dual of this system is

\[
\begin{align*}
\text{maximize} & \quad b^T y - u^T w + b_a y_a, \\
\text{subject to} & \quad A^T y + d_D y_a + z - w = c, \\
& \quad d_P^T y + z_a = c_a, \\
& \quad y_a + z_b = 0, \\
& \quad z, w, z_a, z_b \geq 0,
\end{align*}
\] (2.69)

where \(d_P = b - A x^0\) and \(d_D = A^T y^0 + z^0 - w^0 - c\), \(x^0 > 0, z^0 > 0, w^0 > 0\) and \(y^0\) are arbitrary initial points.

We need to take \(c_a\) and \(b_a\) satisfying

\[
c_a > d_P^T y^0 = (b - A x^0)^T y^0,
\] (2.70)

and

\[
b_a > d_D^T x^0 = (A^T y^0 + z^0 - w^0 - c)^T x^0.
\] (2.71)
so that \((x^0, x^0_a, s^0, x^0_b)\) and \((y^0, y^0_a, z^0, x^0_a, z^0_b, w^0)\) are feasible solution of (2.68) and (2.69), respectively, where

\[
\begin{align*}
  x^0 &= x^0, \\
  x^0_a &= 1, \\
  x^0_b &= b - d^T x^0, \\
  s^0 &= u - x^0, \\
  y^0 &= y^0, \\
  y^0_a &= -1, \\
  z^0 &= z^0, \\
  w^0 &= w^0, \\
  z^0_a &= c_a - d^T y^0, \\
  z^0_b &= 1.
\end{align*}
\]

Therefore, we can apply the algorithm to the artificial problems (2.68) and (2.69) with these initial feasible solutions.

In the limit as \(b_a\) and \(c_a\) approach infinity, the search directions of (2.68) and (2.69) satisfy the following system

\[
\begin{align*}
  A \Delta x &= x_a d_P, \\
  \Delta x + \Delta s &= 0, \\
  A^T \Delta y + \Delta z - \Delta w &= y_a d_D, \\
  Z \Delta x + X \Delta z &= -X Ze + \mu e, \\
  W \Delta s + S \Delta w &= -Sw e + \mu e.
\end{align*}
\]  

(2.72)

From (2.68), we get

\[
x_a d_P = b - A x,
\]

(2.73)
and from (2.69), we get
\[ y_a d_D = c - A^T y - z + w. \] (2.74)

Lustig [65] proposes a primal-dual infeasible-interior-point method which relaxes the primal and dual feasibility assumption on the primal-dual logarithmic barrier method, and shows this method is equivalent to (2.72), which is derived using artificial vectors when the artificial cost become infinitely large. He applies Newton's method to the first order conditions (2.56) without requiring primal and dual feasibility. He assumes only that \( x \) and \( s \) satisfy \( x + s = u \) with \( x > 0 \) and \( s > 0 \), which implies that \( \Delta s \) still satisfies \( \Delta s = -\Delta x \). The defining Newton equations for the change in the variables are then
\[
\begin{align*}
A\Delta x &= b - Ax, \\
\Delta x + \Delta s &= 0, \\
A^T \Delta y + \Delta z - \Delta w &= c - A^T y - z + w, \\
Z \Delta x + X \Delta z &= -XZe + \mu e, \\
W \Delta s + S \Delta w &= -SW e + \mu e.
\end{align*}
\] (2.75)

The resulting search directions are
\[
\begin{align*}
\Delta y &= (A\Theta A^T)^{-1} \{A\Theta [\rho(\mu) - r_D] - (Ax - b)\}, \\
\Delta x &= \Theta (A^T \Delta y - \rho(\mu) + r_D), \\
\Delta z &= \mu X^{-1}e - Ze - X^{-1}Z\Delta x, \\
\Delta w &= S^{-1}(\mu e - SW e + W \Delta x), \\
\Delta s &= -\Delta x,
\end{align*}
\] (2.76)

where \( \Theta = (S^{-1}W + X^{-1}Z)^{-1} \), \( \rho(\mu) = \mu(S^{-1} - X^{-1})e - (W - Z)e \), and \( r_D = A^T y + z - w - c \).

In comparing (2.72) and (2.75), we find they are identical. Therefore, we get the conclusion that the primal-dual logarithmic barrier method using the limiting feasible direction (i.e. \( b_a, c_a \to \infty \)) leads to the primal-dual infeasible-interior-point method.

The infeasible-interior-point method requires only the positivity of \((x^0, s^0)\) and \((z^0, w^0)\) and \(x^0 + s^0 = u\). The generated sequence \((x^k, s^k, y^k, z^k, w^k)\) is not restricted to the interior of
the feasible region. An iterate \((x^k, s^k, y^k, z^k, w^k)\) is required to satisfy neither the equality constraints \(Ax = b\) of primal problem nor \(A^T y - w + z = c\) of the dual problem, but only the positivity \(x > 0\) and \(z > 0\) and \(x + s = u\).

From (2.75) we see that the search direction is still a Newton step toward the point on the central path. It tries to get rid of all the infeasibility in the equality constraints in a single step. If a full step is ever taken (i.e. \(\alpha = 1\)), the residuals \(r_P = Ax - b\) and \(r_D = A^T y + z - w - c\) become zero, and all subsequent iterates remain strictly feasible.

The choice of step length parameter \(\alpha_P\) and \(\alpha_D\) are same as in primal-dual method described in last section. The barrier parameter \(\mu\) is chosen as:

for feasible \(x\) and \(y\),
\[
\mu = \frac{c^T x - b^T y + u^T w}{\phi(n)},
\]  
and when feasibility has not been obtained
\[
\mu = \frac{c^T x - b^T y + u^T w + M\delta_1 + M\delta_2}{\phi(n)},
\]
where
\[
\delta_1 = \|b - Ax\|/\|b - Ax^0\|, \quad \text{(2.79)}
\]
\[
\delta_2 = \|c - A^T y - z + w\|/\|c - A^T y^0 - z^0 + w^0\|, \quad \text{(2.80)}
\]
and
\[
\phi(n) = \begin{cases} n^2, & \text{if } n \leq 5000, \\ n\sqrt{n}, & \text{if } n > 5000. \end{cases} \quad \text{(2.81)}
\]

Here \(M\) is chosen as in [72], namely
\[
M = \rho\phi(n) \max \left\{ \max_{1 \leq j \leq n} \{|c_j|\}, \max_{1 \leq i \leq m} \{|b_i|\} \right\}, \quad \text{(2.82)}
\]
and \(\rho\) is a scalar multiplier.

The algorithm is repeated until the relative duality gap satisfies
\[
\frac{c^T x - b^T y + u^T w}{1 + |b^T y - u^T w|} < \varepsilon,
\]
for a user predetermined \(\varepsilon\).
Mizuno, Todd, and Ye[78] developed the adaptive primal-dual algorithm which is also called a predictor-corrector algorithm. In this algorithm the predictor step is a affine scaling (damped Newton) step for problem (2.56), producing a new strictly feasible iterate. This affine scaling step can make a large reduction in complementarity gap. The subsequent corrector step is a centered Newton step to move the point back to the central path. In this corrector step, the choice of the barrier parameter $\mu$ is based on the predictor step. Both the predictor and corrector steps require essentially the same amount of work, the evaluation and factorization of the Jacobian matrix.

Later on, Mehrotra[76] proposed a more practical predictor-corrector algorithm. A common feature in these two predictor-corrector algorithms is that the value of the barrier parameter in the corrector step depends on the predictor step. However, unlike Mizuno, Todd, and Ye’s corrector step, Mehrotra’s corrector step does not evaluate a new Jacobian matrix. Instead, it reuses the Jacobian matrix used by the predictor step.

Mehrotra’s predictor-corrector method has been used by almost all LP interior-point implementations [76]. In some sense these predictor-corrector methods follow the central path in primal-dual form, but certain of algorithms allow a very loose approximation to the central path.

3.1 Mehrotra’s Predictor-Corrector Method

Mehrotra[76] proposed a predictor-corrector method which takes a single “corrector” step to the central path after each “predictor” step to decrease $\mu$. Lustig present an extension of
Mehrotra’s method in [67]. His method can be derived directly from the first order conditions (2.56). Rather than applying Newton’s method to (2.56) to generate correction term to the current estimate, Lustig substitute the new point into (2.56) directly, yielding

\begin{align*}
A(x + Ax) &= b, \\
(z + Ax) + (s + As) &= u, \\
A^T(y + Ay) + (z + Az) - (w + Aw) &= c, \\
(X + AX)(Z + AZ)e &= \mu e, \\
(S + AS)(W + AW)e &= \mu e,
\end{align*}

where $\Delta X, \Delta S, \Delta Z$, and $\Delta W$ are diagonal matrices with the elements $\Delta x_j, \Delta s_j, \Delta z_j$ and $\Delta w_j$, respectively. Simple algebra reduces (3.1) to the equivalent system

\begin{align*}
A\Delta x &= b - Ax, \\
\Delta x + \Delta s &= u - x - s, \\
A^T\Delta y + \Delta z - \Delta w &= c - A^T y - z + w, \\
Z\Delta x + X\Delta z &= \mu e - XZe - \Delta X\Delta Ze, \\
W\Delta s + S\Delta w &= \mu e - SWe - \Delta S\Delta We.
\end{align*}

The left-hand side of (3.2) is identical to (2.75), while the right-hand side has two distinct differences. The first is the upper bounds equation. In (2.75) we always choose $x^0$ and $s^0$ satisfying $x^0 + s^0 = u$ with $x^0 > 0$ and $s^0 > 0$. This means the right-hand side of the second equation is always zero. For problems with small upper bounds this requirement will bring computational instability. Therefore we assume only that $x^0 > 0$ and $s^0 > 0$ but do not require that the upper-bound constraint $x^0 + s^0 = u$ be satisfied initially. We allow the method to iterate to bound feasibility in precisely the same manner it iterates to primal and dual feasibility. The second difference between (3.2) and (2.75) is the presence of the nonlinear terms $\Delta X\Delta Ze$ and $\Delta S\Delta We$ in the right-hand side of the last two equations in (3.2). Therefore Mehrotra’s predictor-corrector technique incorporates higher order information when approximating the
central trajectory and computing the direction step. This method looks for a parametric representation of the trajectory that goes from a given (infeasible) point in primal and dual spaces to a solution of (2.51) and (2.52).

The predictor-corrector technique decomposes direction step \( \Delta = (\Delta x, \Delta s, \Delta y, \Delta z, \Delta w) \) into two parts

\[
\Delta = \Delta_a + \Delta_c, \tag{3.3}
\]

i.e., combine affine-scaling, \( \Delta_a \) and centering, \( \Delta_c \) components. The term \( \Delta_a \) (affine direction) is obtained by solving

\[
\begin{align*}
A \Delta_a x &= b - Ax, \\
\Delta_a x + \Delta_a s &= u - x - s, \\
A^T \Delta_a y + \Delta_a z - \Delta_a w &= c - A^T y - z + w, \\
Z \Delta_a x + X \Delta_a z &= -XZc, \\
W \Delta_a s + S \Delta_a w &= -SWe.
\end{align*}
\tag{3.4}
\]

Actually the affine scaling direction \( \Delta_a \) is the predictor direction. It then used in two ways. One is to approximate the nonlinear terms in the right-hand side of (3.2). The other one is to dynamically estimate \( \mu \).

Mehrotra use the affine scaling direction \( \Delta_a \) to performs the standard ratio test on both the primal and dual variables to determine the maximum stepsizes \( \delta_P \) and \( \delta_D \) which preserve nonnegativity of \((x,s)\) and \((z,w)\).

\[
\begin{align*}
\hat{\delta}_P &= \min \left\{ \min_j \left\{ \frac{x_j}{-\Delta_a x_j}, \Delta_a x_j < 0 \right\}, \min_j \left\{ \frac{s_j}{-\Delta_a s_j}, \Delta_a s_j < 0 \right\} \right\}, \\
\hat{\delta}_D &= \min \left\{ \min_j \left\{ \frac{z_j}{-\Delta_a z_j}, \Delta_a z_j < 0 \right\}, \min_j \left\{ \frac{w_j}{-\Delta_a w_j}, \Delta_a w_j < 0 \right\} \right\}, \\
\delta_P &= 0.99995\hat{\delta}_P, \\
\delta_D &= 0.99995\hat{\delta}_D,
\end{align*}
\tag{3.5}
\]

where the 0.99995 factor keeps the iterates from actually touching the boundary.
Chapter 3. Predictor-Corrector Methods

The predicted complementarity gap that would result from a step in the affine direction is

\[ g_a = (x + \delta_P \Delta_a x)^T(z + \delta_D \Delta_a z) + (s + \delta_P \Delta_a s)^T(w + \delta_D \Delta_a w). \]  

(3.6)

Lustig's [67] estimate for \( \mu \) is then

\[ \mu = \left( \frac{g_a}{x^T z + s^T w} \right)^2 \left( \frac{g_a}{n} \right) = \left( \frac{g_a}{g} \right)^2 \left( \frac{g_a}{n} \right). \]  

(3.7)

where \( g = x^T z + s^T w \) denotes current complementarity gap. The term \( g_a/g \) measures the achievable progress in the affine scaling direction.

Now (3.7) chooses a small \( \mu \) when good progress can be made in the affine direction and a large \( \mu \) when the affine direction produces little improvement. This means the poor progress in the affine direction generally indicates the need for more centering and hence a large value of \( \mu \). If the affine scaling direction offers considerable progress in the reduction of the complementarity gap, then a more optimistic target is chosen.

The actual new step \((\Delta x, \Delta y, \Delta z, \Delta s, \Delta w)\) is then chosen as the solution to

\[
\begin{align*}
A\Delta x &= b - Ax, \\
\Delta x + \Delta s &= u - x - s, \\
A^T \Delta y + \Delta z - \Delta w &= c - A^T y - z + w, \\
Z \Delta x + X \Delta z &= \mu e - X Ze - \Delta_a X \Delta_a Ze, \\
W \Delta s + S \Delta w &= \mu e - SW e - \Delta_a S \Delta_a We.
\end{align*}
\]  

(3.8)

Note that the full primal-dual affine scaling terms, \( \Delta_a X \Delta_a Ze \) and \( \Delta_a S \Delta_a We \), are added to the right-hand side of (3.8). The step lengths \( \delta_P \) and \( \delta_D \) defined by (3.5) are used to compute \( \mu \) but not to modify the affine scaling correction terms. Thus the affine correction added is one that results from taking a full step of length one in the affine direction. Whenever a primal and dual full step of length one can be taken, primal feasibility and dual feasibility are achieved exactly within the numerical accuracy of the computations. So the solution to (3.8) can be
written as

\[
\begin{align*}
\Delta y &= \Delta ay + \Delta cz, \\
\Delta x &= \Delta ax + \Delta cx, \\
\Delta z &= \Delta az + \Delta cz, \\
\Delta w &= \Delta aw + \Delta cw, \\
\Delta s &= \Delta as + \Delta cs,
\end{align*}
\tag{3.9}
\]

where \(\Delta ax, \Delta ay, \Delta az, \Delta aw\) and \(\Delta as\) are the solution to (3.4) and the correction terms \(\Delta cx, \Delta cy, \Delta cz, \Delta cw\) and \(\Delta cs\) satisfy

\[
\begin{align*}
A\Delta cx &= 0, \\
\Delta cx + \Delta cs &= 0, \\
A^T\Delta cy + \Delta cz - \Delta cw &= 0, \\
Z\Delta cx + X\Delta cz &= \mu e - \Delta axX\Delta az e, \\
W\Delta cs + S\Delta cw &= \mu e - \Delta asS\Delta aw e.
\end{align*}
\tag{3.10}
\]

According to (3.4)

\[
x^Tz + \Delta ax^Tz + \Delta az^Tz = s^Tw + \Delta aw^Tw + \Delta aw^Ts = 0.
\tag{3.11}
\]

If the full step of one were achieved on the affine scaling step, the new complementarity gap would be

\[
(x + \Delta ax)^T(z + \Delta az) + (s + \Delta as)^T(w + \Delta aw)
= x^Tz + \Delta ax^Tz + \Delta az^Tz + \Delta ax^T\Delta az + s^Tw + \Delta as^Taw + \Delta aw^Tw + \Delta aw^Ts + \Delta as^T\Delta aw
= \Delta ax^T\Delta az + \Delta as^T\Delta aw.
\tag{3.12}
\]

Therefore (3.10) could describe a centered Newton step from the point \(x + \Delta ax, y + \Delta ay, z + \Delta az, s + \Delta as\) and \(w + \Delta aw\) except that the corrections \(\Delta ax, \Delta ay, \Delta az, \Delta as\) and \(\Delta aw\) have not been added to the diagonal matrices on the left-hand side of (3.10). Thus the centering correction terms are a Newton step from the point achieved by a full affine step, but using
the second-derivative matrix at the current point \( x, y, z, w \) and \( s \) as an approximation to the Hessian at the point corresponding to a full affine step.

From (3.4) and (3.10), we see that a single iteration of the predictor-corrector primal-dual method needs two solutions of the same large, sparse linear system for two different right hand sides.

The iterate obtained from the predictor-corrector method can be viewed as a point on a quadratic path which approximates the central trajectory and pushes an iterate towards an optimum along such an approximation. Tapia et al.\[96\] demonstrate that Mehrotra's predictor-corrector methods is the level-1 composite Newton variant of the Newton interior-point method and prove that it has a quadratic local convergence rate.

The computational results of Lustig\[67\] show that the predictor-corrector method almost always reduces the iteration count and usually reduces computation time. Furthermore, as problem size and complexity increase, the improvements in both iteration count and execution time become greater. Thus the predictor-corrector method is generally very computationally efficient.

3.2 Carpenter's Higher-Order Predictor-Corrector Method

As we described in previous section, Mehrotra's predictor-corrector method first solves (3.4) for the predictor direction \( \Delta_a \), then it computes the barrier parameter \( \mu \) as a function of both the current point \( (x, s), (y, z, w) \) and \( \Delta_a \), and finally, it uses \( \Delta_a \) to correct the centered direction that would be obtained by solving (3.10). The systems we solve to obtain the predictor or the corrector direction each involve the same matrix. Each of these directions is obtained based on the evaluation of the same derivative. The idea of the predictor-corrector procedure is to reduce the work required in the primal-dual interior point procedure by reusing the factorization required to solve the Newton system (3.4). The factorization phase is computationally much more expensive than the solve phase.

The method in last section performs only one correction in obtaining the direction. It can
Chapter 3. Predictor-Corrector Methods

easily be extended to perform several corrections in a multiple predictor-corrector procedure. Instead of solving (3.8) once at each step of the primal-dual interior point method, it can be solved repetitively with each direction corrected based on the previous direction. That is, we substitute at each step the $\Delta x, \Delta y, \Delta z, \Delta w$ and $\Delta s$ terms found by solving (3.8) back into the right-hand side $\Delta a$.

Now, we outline the multiple predictor-corrector algorithm [19] which is invoked at each iteration of the IPM.

*Algorithm MPC (Multiple Predictor-Corrector)*

Given $\nu^k = (x^k, s^k, y^k, z^k, w^k)$ with $x^k, s^k, z^k, w^k > 0$.

**Step 1:** Solve (3.4) for the affine direction $\Delta a \nu$.

**Step 2:** Compute $\mu(\nu^k, \Delta a \nu)$ using (3.7).

**Step 3:** For $i = 1, \ldots, m_k$ ($m_k$: the number of corrections in iteration $k$) do

solve the following system for $\Delta \nu^i$:

\begin{align*}
A \Delta x^i &= b - A x, \\
\Delta x^i + \Delta s^i &= u - x - s, \\
A^T \Delta y^i + \Delta z^i - \Delta w^i &= c - A^T y - z + w, \\
Z \Delta x^i + X \Delta z^i &= \mu e - X Ze - \Delta X^{i-1} \Delta Z^{i-1} e, \\
W \Delta s^i + S \Delta w^i &= \mu e - S We - \Delta S^{i-1} \Delta W^{i-1} e.
\end{align*}

**end do.**

Define $\Delta \nu = \Delta \nu^{m_k}$.

**Step 4:** Perform the ratio test to determine primal and dual steplength $\alpha_P$ and $\alpha_D$. 
Step 5: Move to the new point $\nu^{k+1}$ defined by

$$
egin{align*}
x^{k+1} & = x^k + \alpha_P \Delta x, \\
s^{k+1} & = s^k + \alpha_P \Delta s, \\
y^{k+1} & = y^k + \alpha_D \Delta y, \\
z^{k+1} & = z^k + \alpha_D \Delta z, \\
w^{k+1} & = w + \alpha_D \Delta w.
\end{align*}
$$

\text{(3.14)}

The primal and dual step lengths $\alpha_P$ and $\alpha_D$ are chosen to insure the nonnegativity of the variables $x, s, z$ and $w$. Given $\nu$ and a direction $\Delta \nu$, the ratio functions $\hat{\alpha}_P(\nu, \Delta \nu)$ and $\hat{\alpha}_D(\nu, \Delta \nu)$ are defined as

\begin{align*}
\hat{\alpha}_P(\nu, \Delta \nu) & = \min \left\{ \min_j \left\{ \frac{z_j}{-\Delta x_j}, \Delta x_j < 0 \right\}, \min_j \left\{ \frac{s_j}{-\Delta s_j}, \Delta s_j < 0 \right\} \right\}, \\
\hat{\alpha}_D(\nu, \Delta \nu) & = \min \left\{ \min_j \left\{ \frac{z_j}{-\Delta z_j}, \Delta z_j < 0 \right\}, \min_j \left\{ \frac{w_j}{-\Delta w_j}, \Delta w_j < 0 \right\} \right\}.
\end{align*}

\text{(3.15)}

$$
\alpha_P = 0.99995 \hat{\alpha}_P(\nu, \Delta \nu), \\
\alpha_D = 0.99995 \hat{\alpha}_D(\nu, \Delta \nu),
$$

$\alpha_P$ and $\alpha_D$ are the steps that are actually taken in MPC.

Carpenter et al. [19] give a heuristic strategy for determining $m_k$: the number of corrections in iteration $k$. They make different consideration on feasible point and infeasible point.

For feasible points, let $g^i$ be the complementarity that would result from taking a step in the direction obtained after $i$ corrections. If $g^i < g^{i-1}$ and $i$ is less than some maximum number of corrections, then an $(i + 1)^{st}$ correction is performed. If $g^i \geq g^{i-1}$, then we stop correcting and use the direction $\Delta \nu^{i-1}$. Therefore, another correction is considered only if the previous one decreased the complementarity.

When correcting from an infeasible point, the value $m_k$ is chosen so that the algorithm reduces complementarity while reducing infeasibility. Carpenter et al. [19] define

$$
G^i = g^i + (1 + \alpha_D)\|d_D\|_1 + (1 - \alpha_P)(\|d_P\|_1 + \|d_u\|_1).
$$

\text{(3.16)}
where $d_P, d_D$ and $d_u$ are the primal, dual and upper bound infeasibilities at the current point which are defined as

$$d_P = b - Ax,$$
$$d_D = c - A^T y - z + w,$$
$$d_u = u - x - s.$$  \hspace{1cm} (3.17)

If $i$ is less than the allowable maximum and $G^i < G^{i-1}$, then an $(i+1)$st correction is performed.

Carpenter et al. proved that the multiple predictor-corrector algorithm is equivalent to the level-$m$ composite Newton method.

Although the use of higher order terms usually results in the reduction of the number of iterations to reach optimality, it does not guarantee the total time savings due to the increased effort in a single iteration. Computational results indicate that on average, the second order methods seem to be the most efficient. In the case that Cholesky factorization is very expensive compared with a single solve of Newton equations, we prefer to use higher (than two) order method to save the number of factorizations. But Carpenter's higher-order method has difficulty of building an accurate higher order approximation of the central trajectory. The method presented in the next section solves this problem.

### 3.3 Gondzio's Multiple Centrality Corrections Method

Gondzio offered a multiple centrality corrections method in [43]. This method combines the multiple centrality correction with a choice of reasonable well centered targets that are supposed to be easier to reach than perfectly centered analytic centers. The algorithm takes special care on restoring centrality of the next iterate and, at the same time, to increase stepsize in the primal and dual spaces. The effort of multiple corrections does not concentrate on reducing the complementarity gap. He believe the complementarity gap will be sufficiently reduced if a long step along a primal-dual affine scaling direction is made. Driving the primal-dual point as close to the central path as possible is thus an investment that is expected to pay off in the ability to make a larger step in the next iterate.
This method uses the second order predictor-corrector technique which introduced in Section 3.1 to produce the predictor direction. The following corrector terms concern improving the centrality of the subsequent iterate and with increasing the stepsizes in primal and dual spaces.

For a given point \((x, s)\) and \((y, z, w)\), the quality of centrality is measured by

\[
\delta^2(x, s, y, z, w, \mu) = \sum_{j=1}^{n} \left( \sqrt{\frac{x_j z_j}{\mu}} - \sqrt{\frac{\mu}{x_j z_j}} \right)^2 + \sum_{j=1}^{n} \left( \sqrt{\frac{s_j w_j}{\mu}} - \sqrt{\frac{\mu}{s_j w_j}} \right)^2, \tag{3.18}
\]

or

\[
\delta(x, s, y, z, w, \mu) = \|Xz - \mu e\| + \|X^{-1}Z^{-1}e - \frac{1}{\mu}e\| + \|Sw - \mu e\| + \|S^{-1}W^{-1}e - \frac{1}{\mu}e\|. \tag{3.19}
\]

What really hurts a primal-dual algorithm is a large discrepancy between complementarity products \(x_jz_j\) and \(s_jw_j\). Both too small and too large complementarity products, compared with their average \(\mu_a = (x^T z + s^T w)/(2n)\), are undesirable.

The step \(\Delta\) of (2.75) aims at drawing all complementarity products to the same value \(\mu\).

Usually, there is little hope to reach such a goal. Gondzio's[43] approach combines the choice of targets[56] that are supposed to be easier to reach with the use of multiple correctors.

Assume \((x, s)\) and \((y, z, w)\) are primal and dual solutions at a given iteration of the primal-dual algorithm, and \(x, s, z\) and \(w\) are strictly positive. We also assume that the predictor direction \(\Delta_P\) at this point, and the maximum stepsizes in primal \(\alpha_P\) and dual \(\alpha_D\) are determined.

The algorithm looks for a corrector direction \(\Delta_m\) such that larger stepsizes in primal and dual spaces are allowed for a composite direction

\[
\Delta = \Delta_P + \Delta_m. \tag{3.20}
\]

To enlarge these stepsizes from \(\alpha_P\) and \(\alpha_D\) to \(\tilde{\alpha}_P = \min(\alpha_P + \delta_\alpha, 1)\) and \(\tilde{\alpha}_D = \min(\alpha_D + \delta_\alpha, 1)\), respectively, a corrector term \(\Delta_m\) has to compensate for the negative components in the primal and dual variables

\[
\begin{align*}
(\tilde{x}, \tilde{s}) &= (x, s) + \tilde{\alpha}_P(\Delta_P x, \Delta_P s), \\
(\tilde{y}, \tilde{z}, \tilde{w}) &= (y, z, w) + \tilde{\alpha}_D(\Delta_P y, \Delta_P z, \Delta_P w).
\end{align*} \tag{3.21}
\]
Chapter 3. Predictor-Corrector Methods

The algorithm tries to reach the goal by adding the corrector term $\Delta_m$ that drives from this exterior trial point to the next iterate $(\hat{x}, \hat{s}, \hat{y}, \hat{z}, \hat{w})$ lying in the vicinity of the central path. However, there is little chance to attain the analytic center in one step, i.e., to reach the point

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

in the space of complementarity products. Hence the algorithm computes the complementarity products $\tilde{\nu} = (\tilde{Xz}, \tilde{Sw} \in \mathbb{R}^{2n})$ and concentrates its effort only on correcting their outliers. Thus, these complementarity products are projected component-wise on a hypercube $H = [\beta_{min}, \beta_{max}]^{2n}$. This gives the following target

$$\nu_t = \pi(\tilde{\nu} | H) \in \mathbb{R}^{2n}.$$  \hfill (3.23)

The corrector term of the direction is defined by the following system of linear equations

$$\begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
I & 0 & I & 0 & 0 \\
0 & A^T & 0 & I & -I \\
Z & 0 & 0 & X & 0 \\
0 & 0 & W & 0 & S
\end{bmatrix}
\begin{bmatrix}
\Delta_m x \\
\Delta_m y \\
\Delta_m s \\
\Delta_m z \\
\Delta_m w
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
0 \\
\nu_t - \tilde{\nu}
\end{bmatrix}.  \hfill (3.24)
$$

The right hand side in the above system of equations has nonzero elements only in a subset of positions of $\nu_t - \tilde{\nu}$ that refer to the complementarity products which do not belong to the interval $(\beta_{min}, \beta_{max})$.

Once corrector term $\Delta_m$ is computed, the new stepsizes $\alpha_p$ and $\alpha_D$ are determined for the composite direction

$$\Delta = \Delta_p + \Delta_m.$$ \hfill (3.25)

Then the primal-dual algorithm can move to the next iteration.

The correcting process can easily be repeated a desirable number of times. Direction $\Delta$ of (3.25) becomes a new predictor $\Delta_p$ for which a new trial point is computed from (3.21). The point in the complementarity products space is then used to define the new target (3.23). Then,
Chapter 3. Predictor-Corrector Methods

a new modified centering direction \( \Delta_m \) from (3.24) is computed and added to the predictor term as in (3.25).

In this algorithm, correcting terminates when the stepsizes in the primal and dual spaces \( \hat{\alpha}_P \) and \( \hat{\alpha}_D \) determined for a composite direction (3.25) do not increase sufficiently compared with the stepsizes \( \alpha_P \) and \( \alpha_D \) found earlier for a predictor direction. That is, we stop correcting if

\[
\hat{\alpha}_P < \alpha_P + \gamma \delta_\alpha \quad \text{or} \quad \hat{\alpha}_D < \alpha_D + \gamma \delta_\alpha,
\]

(3.26)

where \( \gamma \) is some prescribed tolerance.

The computational experience of [43] shows that this method saves significant CPU time over the Mehrotra's predictor-corrector method.

The Higher Order Primal-Dual Method (HOPDM) [45] which we will use in Chapter 6 is an implementation of this algorithm.

3.4 Starting Point

All interior point methods (including the infeasible IPMs) are very sensitive to the choice of an starting point. The choice of a good starting point for an IPM is still an open question. Good guesses of \((x^0, s^0, y^0, z^0, w^0)\) can reduce the computational effort considerably. One would like the starting point to be well centered and to be as close to primal and dual feasibility as possible. Surprisingly, points that are relatively close to the optimal solution (but are not well centered) often lead to bad performance and/or numerical difficulties.

The starting point in most implementations of primal-dual infeasible IPMs [45, 76, 66] are some variation of the approximate solution of the following auxiliary QP problem which is given in [4]:

\[
\begin{aligned}
\text{minimize} \quad & c^T x + \frac{\rho}{2} (x^T x + s^T s), \\
\text{subject to} \quad & Ax = b, \\
& x + s = u,
\end{aligned}
\]

(3.27)

where \( \rho \) is a predetermined weight parameter. A solution of (3.27) can be given by an explicit
formula and can be computed at a cost comparable to a single interior point iteration. It is supposed to minimize the norm of the primal solution \((x, s)\) and promote points that are better in the sense of the LP objective. As the solution of (3.27) may have negative components in \(x\) and \(s\), those negative components are pushed towards positive values sufficiently bounded away from zero (all elements smaller than \(\delta\) are replaced by \(\delta\), say, \(\delta = 1\)). Independently, an initial dual solution \((y, z, t)\) is chosen similarly to satisfy \(y = 0\) and the dual constraint. Again, all elements of \(z\) and \(t\) smaller than \(\delta\) are replaced by \(\delta\).
Chapter 4

Solving the Newton Equations

In section 2.5, we obtained Newton equations (2.75). If we do not require that the upper bound constraint $x^0 + s^0 = u$ is satisfied initially, i.e. we allow the method to iterate to bound feasibility in precisely the same manner it iterates to primal and dual feasibility, then we get the Newton equations system

\[
\begin{bmatrix}
A & 0 & 0 & 0 & 0 \\
I & 0 & I & 0 & 0 \\
0 & A^T & 0 & I & -I \\
Z & 0 & 0 & X & 0 \\
0 & 0 & W & 0 & S \\
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta s \\
\Delta z \\
\Delta w \\
\end{bmatrix}
= 
\begin{bmatrix}
\xi_b \\
\xi_u \\
\xi_c \\
\mu e - XZe \\
\mu e - SWe \\
\end{bmatrix},
\]

(4.1)

where

\[
\xi_b = b - Ax, \\
\xi_u = u - x - s, \\
\xi_c = c - A^Ty - z + w,
\]

denote the violation of the primal, the upper bound and the dual constraints, respectively.

After elimination of

\[
\Delta z = X^{-1}(\mu e - XZe - Z\Delta X), \\
\Delta S = \xi_u - \Delta x, \\
\Delta w = S^{-1}(\mu e - SWe - W\Delta s) = S^{-1}(\mu e - SWe - W\xi_u + W\Delta x),
\]

42
Chapter 4. Solving the Newton Equations

it reduces to

\[
\begin{bmatrix}
-D^{-2} & A^T \\
A & 0
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y
\end{bmatrix} =
\begin{bmatrix}
r \\
h
\end{bmatrix},
\]

(4.2)

where

\[
D^2 = (X^{-1}Z + S^{-1}W)^{-1},
\]

\[
r = \xi_c - X^{-1}(\mu e - XZe) + S^{-1}(\mu e - SWe) - S^{-1}W\xi_u,
\]

\[
h = \xi_b.
\]

This form of the Newton equations system usually is known as the augmented system. The coefficient matrix of this system is symmetric indefinite and is easier and cheaper to factorize than the original form.

We can further reduce (4.2) to the normal equations

\[
(AD^2A^T)\Delta y = g = AD^2r + h
\]

(4.3)

by eliminating \(\Delta x\) from (4.2). The matrix for normal equations is positive definite and symmetric, has smaller size \((m \times m)\), and may be more dense.

The solution of the Newton equations system is the computationally most involved task in any interior point method. All IPMs solve an identical system of linear equations. The only difference is in the value of the diagonal matrix \(D^2\) and the right-hand side. The coefficient matrix in the Newton equations system is usually large and sparse, since the constraint matrix \(A\) is itself large and sparse in most applications.

Both linear systems (4.2) and (4.3) can be solved using either direct or iterative methods. These two approaches are not completely disjoint; the process of constructing a preconditioner for iterative methods is in many ways similar to direct factorization [41]. The choice of the method (direct or iterative) depends on a large number of factors. Karmarkar in [58] gave a detailed discussion of these factors and their influence on the choice of the method.

Direct methods involve the factorization of the system coefficient matrix, usually obtained through Gaussian elimination. Implementations of methods in this class require the use of
specific data structures and special pivoting strategies, in an attempt to reduce the amount of
fill-in during Gaussian elimination. Notable examples of software using this type of technique
are SPARSPAK[34], YSMP[30], and MA27[29, 28].

Iterative methods generate a sequence of approximate solutions to the system of linear
equations usually involving only matrix-vector multiplications in the computation of each iterate. Methods like Jacobi, Gauss-Seidel, Chebychev, Lanczos (see [41] for a description of these
methods) and the conjugate gradient are attractive by virtue of their low storage requirements,
but these methods have slow convergence unless an effective preconditioner is applied.

For IPMs, iterative methods, e.g., conjugate gradient algorithms, are not competitive in the
general case due to the difficulties in choosing a good and computationally cheap preconditioner.
Some success with iterative methods for special LP problems has been obtained [84, 89, 85].
Resende indicated in [88] that the direct method is not appropriate for solving Newton equations
system that occurs in network flow problems. This is because the $AD^2A^T$ and its Cholesky
factor are much denser than the matrix $A$, even if the incidence matrix $A$ has no dense columns.

Portugal and Resende[85] used an iterative method to solve the minimum cost network flow
problem. They used a heuristic to select the preconditioner. The initial selection is the diagonal
preconditioner, since it tends to outperform the other preconditioners during the initial interior
point iterations. If the number of conjugate gradient iterations exceeds $\sqrt{m}$, the diagonal
preconditioner is changed to the spanning tree preconditioner.

For general purpose IPMs, a direct method is better than an iterative method to solve
the Newton equations. Hence, almost all the available implementations of the general purpose
IPMs use a direct approach to solve the Newton equations and they all use some variant of
the symmetric triangular $LAL^T$ decomposition, where $L$ is a lower triangular matrix and $\Lambda$ is
a block diagonal matrix with blocks of size 1 or 2. An alternative direct approach is the QR
decomposition of $A$. Although this approach uses orthogonal transformations and guarantees
high accuracy of solutions, it cannot be used in practice since it is prohibitively expensive.

Another more efficient approach for solving Newton equations system is called the adaptive
approach. This approach combines the advantages of both the direct and iterative methods. It uses an adaptive automated procedure to determine whether to use a direct or iterative method. Computational results show this approach is more efficient than a pure direct method.

4.1 Direct Methods

From the previous discussion, we get the conclusion that one practical approach to solving the Newton equations in general purpose IPM codes is the direct method using $LAL^T$ decomposition. There are two major alternative approaches corresponding to the solving of the normal equations (4.3) and the augmented system (4.2).

4.1.1 The Normal Equations Approach

Many current implementations [57, 38, 67, 72, 76] solve the normal equations directly by using Gaussian elimination to compute the Cholesky factorization $LAL^T = AD^2A^T$. A major advantage of this approach is that it is based on a symmetric positive definite matrix guaranteed to yield a factorization of the form $LAL^T$, where $L$ is lower triangular with unit diagonal and $A$ is diagonal with strictly positive entries. Given the $LAL^T$ decomposition, $\Delta y$ is easily obtained from (4.3) by solving two triangular systems. Moreover, the sparsity pattern in the decomposition is independent of the value of $D^2$ and hence is constant in all IPM iterations. Therefore, a good sparsity-preserving pivot order can be chosen with much care, even if it involves considerable computational effort, since it will be used extensively throughout the whole solution process. The sparsity-preserving sequence of pivots (ordering) in which the symmetric Gaussian elimination is performed can be defined in advance, i.e., before the numerical operations start. Thus the analysis phase is completely separated from the factorize phase [27] and the analysis phase has to be done only once, before the whole solution process.

The success of the implementation of the Cholesky factorization depends on the quality of its analysis phase, i.e. reordering for sparsity. Its goal is to find a permutation matrix $P$ such that the Cholesky factor of $PAD^2A^TP^T$ is the sparsest possible.
The overall solution of the normal equations is typically divided into four distinct independent phases:

1. Find an appropriate ordering $P$ for $AD^2A^T$.

2. Set up a data structure for $L$, the Cholesky factor of $PAD^2A^TP^T$.

3. Numerically factor $PAD^2A^TP^T$ into $LAL^T$.

4. Solve $(LAL^T)(PAy) = Pg$.

Step 1 and step 2 are the analysis phase. They depend only on the structure of $AD^2A^T$ and are independent of its numerical values. Step 3 and step 4 are the factorize phase.

Finding the permutation that yields the minimum fill-in in the Cholesky factors is an NP-hard problem[111]. As a practical consequence, we cannot expect to find an efficient algorithm that identifies an optimal ordering. Hence, heuristics are devised that efficiently compute a permutation that approximates the effect of the optimal one.

Two such heuristics, namely the minimum degree and the minimum local fill-in orderings [27, 34, 35] are particularly useful in the context of IPM implementations. Now we give a brief description of these two heuristics.

- Minimum Degree Ordering

The minimum degree ordering algorithm follows the Markowitz criterion[69], which is designed for unsymmetric matrices. Markowitz observed that in the $k$th step of an sparse Gaussian elimination the local best pivot candidate $a_{ij}$ is the one that minimizes

$$f_{ij} = (r_i - 1)(c_j - 1), \quad (4.4)$$

where $r_i$ and $c_j$ are the numbers of nonzero entries of row $i$ and column $j$ in the $k$th Schur complement. The value $f_{ij}$ gives the number of floating point operations required by the $k$th step of Gaussian elimination and, at the same time, estimates the potential fill-in caused by this step.
Tinney and Walker[97] applied this strategy to symmetric matrices which results in the minimum degree ordering heuristic[92, 97]. This heuristic can be regarded as an attempt to minimize the number of arithmetic operations in the next stage of the Gaussian elimination algorithm, and as an attempt to select the pivot column that introduces the least fill-in in the next stage. This local minimization strategy does not guarantee a global minimum for either the amount of fill-in or total number of arithmetic operations performed during Gaussian elimination. Nevertheless, the strategy has been proved to be very effective in reducing arithmetic operations and fill-ins.

When symmetric positive definite systems are considered, pivot selection is restricted to diagonal elements, hence the Markowitz merit function simplifies to

\[ f_j = (c_j - 1)^2 \]  \hspace{1cm} (4.5)

which leads to a simple rule that the best candidate for the pivot column is, among the columns in the portion of the matrix still to be factored, the one with the minimum number of nonzero entries. Interpreting this process in terms of the elimination graph[35] one sees that this is equivalent to the choice of the node that has the minimum degree, which gave the name to this heuristic.

The minimum degree ordering algorithm can be implemented efficiently both in terms of speed and storage requirements. George and Liu in[35] described several powerful enhancements of the basic algorithm. The various methods that have been developed to improve the performance of the basic minimum degree algorithm are mass elimination, indistinguishable nodes(supernodes), incomplete degree update, element absorption and multiple elimination. George and Liu[35] also discussed the tie-breaking strategy used to select the next pivot column among all the columns matching the minimum number of nonzero entries.

- Minimum Fill-in Ordering

The minimum fill-in ordering heuristic results from the effort of trying to reduce the
amount of fill-in further than the Markowitz criterion. The function (4.5) considerably overestimates the expected number of fill-ins in a given iteration of the Gaussian elimination because it does not take into account the fact that in many positions of the predicted fill-in, nonzero entries already exist. It is possible that another node, although more expensive in term of (4.5), would produce less fill-in as the elimination step would mainly update already existing nonzero entries of the the Schur complement. The minimum local fill-in ordering heuristic selects, at each stage of the Gaussian elimination procedure, the pivot element that introduces the minimum amount of fill-in. To count predicted fill-in one has to simulate the elimination step, which is quite an expensive operation. Generally, the minimum local fill-in algorithm produces a sparse factorization but at higher initial cost to obtain the ordering.

The motivation for introducing the minimum local fill-in heuristic in IPM is straightforward. IPMs solve a sequence of systems of linear equations sharing an identical nonzero structure. Therefore, the ordering procedure will be executed only once at the beginning of the algorithm, and the resulting permutation remains valid for the remaining iterations. By contrast, the Gaussian elimination procedure is repeated in every iteration of the IPM and any computational savings achieved by a better ordering is multiplied by the total number of iterations of the algorithm. Moreover, the effort of ordering a matrix according to either heuristic does not constitute a significant portion of the algorithm’s total computation effort. Therefore, the extra effort involved in performing the minimum local fill-in heuristic does not impact significantly the algorithm’s running time.

The normal equations approach has been observed to work well for many linear programs. Nevertheless, it is known to encounter difficulties in two common situations.

The first drawback of the normal equations approach is that dense columns in $A$ can create unacceptably dense matrix $AD^2A^T$ and thus unacceptably dense Cholesky factors $L$. There are several methods to solve this problem. Adler[2] removes the dense columns from $A$, and then corrects for their absence by a conjugate gradient method to solve the normal equations using
the incomplete Cholesky factors as preconditioners. Gill et al. [37] and Choi et al. [24] employ the Schur complement algorithm and partition matrix $A$ into sparse and dense columns. Let

$$A = [A_1 \ A_2],$$

(4.6)

where $A_1 \in \mathbb{R}^{m \times (n-k)}$ and $A_2 \in \mathbb{R}^{m \times k}$ are matrices built of sparse and dense columns, respectively. The Schur complement mechanism based on (4.6) and an explicit decomposition of the matrix

$$AD^2A^T = A_1D_1^2A_1^T + A_2D_2^2A_2^T$$

(4.7)

into a presumably sparse part $A_1D_1^2A_1^T$ and a significantly denser symmetric rank $k$ update of it. A Cholesky decomposition is then computed for the "sparse" part and the dense rank-$k$ update is handled via the Sherman-Morrison-Woodbury formula. This method is not guaranteed to work correctly because the sparse part may be rank deficient. Whenever this happens, the Cholesky decomposition of $A_1D_1^2A_1^T$ does not exist and the Sherman-Morrison-Woodbury update is not well defined. Therefore in a practical implementation a small diagonal regularization term is added to $A_1D_1^2A_1^T$ such that the decomposition exists. The method usually works satisfactorily for a small number of dense columns. Andersen [5] proposed a remedy to the rank deficiency arising in the Schur complement mechanism.

Lustig [67] proposed an algorithm which combines Adler’s and Gill’s method efficiently. This algorithm monitors the spread between the largest and the smallest diagonal elements of the factorization $L_1L_1^T = A_1D_1^2A_1^T$. When this spread becomes large, all smaller diagonal elements are set to 1 and the remaining elements of these columns are set to 0. These Cholesky factors are used as preconditioners for a preconditioned conjugate gradient method with the dense columns added to the matrix $A$. This method eliminates rank deficiency and removes much of the instability.

A more numerically sound alternative is based on expanded normal equations formed by "splitting" dense columns [42, 104], but the efficiency of this kind of modification is not fully established. In any case, all of these approaches have the drawback of requiring that dense columns be identified in $A$ in some arbitrary way.
The second difficulty occurs when the linear program contains "free variables" which are not explicitly bounded above or below. To transform the problem to the standard form, any free variable has to be replaced with a difference of two nonnegative variables: \( x_f = x^+ - x^- \). The presence of logarithmic terms in the objective function causes very fast growth of both split brothers. Although their difference may be kept relatively close to the optimal value of \( x_f \), both \( x^+ \) and \( x^- \) tend to infinity. This results in a serious loss of accuracy in (4.6). A remedy used in many IPM implementations is to prevent excessive growth of \( x^+ \) and \( x^- \).

The difficulties of the normal equations motivated several researchers to pay special attention to the augmented system form of the Newton equations which allows more freedom in the pivot choice.

### 4.1.2 The Augmented System Approach

An alternative approach that avoids the difficulties of the normal equations is to apply elimination directly to factor the larger "augmented system" (4.2). This approach is based on performing a Bunch-Parlett[18] factorization to the symmetric indefinite matrix

\[
K = \begin{bmatrix}
-D^{-2} & A^T \\
A & 0
\end{bmatrix}.
\]  

This factorization has the form \( PKP^T = L\Lambda L^T \), where \( P \) is a permutation matrix and \( \Lambda \) is a symmetric indefinite block diagonal matrix with \( 1 \times 1 \) and \( 2 \times 2 \) blocks.

Because the matrix is symmetric indefinite, the equations (4.2) can not be solved quite so easily as the normal equations. There exists a stable elimination procedure for symmetric indefinite systems, due to Bunch and Parlett[18], but it must allow for certain \( 2 \times 2 \) pivots along the diagonal in addition to the usual \( 1 \times 1 \) pivot elements. The factorization \( L\Lambda L^T \) is guaranteed to exist[39], but unlike the positive definite case(normal equations approach), the pivot order cannot be computed symbolically because the permutation choice is based on numerical values and both the sparsity and stability of the triangular factor. Thus, the pivot order cannot be fixed once in the analysis phase, but must be recomputed as \( x \) and \( z \) are updated in each
iteration. On the other hand, due to the greater freedom in the choice of the pivot order, the augmented system factorization may produce significantly sparser factors than that of the normal equations.

Actually the normal equations are a special case of the augmented system (4.8) in which the first \( n \) pivots are chosen from the \( D^2 \) part regardless their stability properties and without any concern about the fill-in they produce. Since \( D \) is positive, no zero pivots are encountered, and after the \( n \)th stage of symmetric elimination the partially reduced matrix has the form

\[
\begin{pmatrix}
-D^{-2} & 0 \\
0 & AD^2A^T
\end{pmatrix}
\]

Each stage of the elimination performs a rank-one update of the lower right \( m \times m \) submatrix, and after \( n \) stages, \( AD^2A^T \) is explicitly formed. The remaining matrix is positive definite and the factorization \( L_mD_mD_m^T \) of \( AD^2A^T \) is guaranteed to exist. Moreover, the last \( m \) pivots can be performed in any order, using whatever efficient pivoting strategy is preferred. For example, the minimum degree or the minimum local fill ordering heuristics might be used. When the symmetric elimination is complete, we have

\[
K = L_kD_kL_k^T
\]

where \( D_k \) is an \((n + m) \times (n + m)\) diagonal matrix of the form

\[
D_k = \begin{bmatrix}
-D^{-2} & 0 \\
0 & D_m
\end{bmatrix}
\]

and

\[
L_k = \begin{bmatrix}
I & 0 \\
-AD^2 & L_m
\end{bmatrix}
\]

Hence, the normal equations approach is embedded in the augmented system approach as one particular pivot strategy. Vanderbei[105] call this pivot strategy the conservative pivot strategy. Solving (4.2) and solving (4.3) are roughly equivalent under the conservative strategy.

Observing (4.9), when \( A \) has dense column \( a_j \), \( a_ja_j^T \) becomes a fully dense \( m \times m \) matrix. Consequently, \( AD^2A^T \) formed in the lower right hand corner is also fully dense. So if \( A \) has dense column, the conservative pivot strategy can dramatically impair the efficiency of solving the augmented system.
Chapter 4. Solving the Newton Equations

The success of the augmented system factorization depends highly on the efficiency of the pivot selection rule. More freedom in the pivot choice opens the possibility of getting sparser factors than in the normal equations approach (e.g., the degrading influence of dense columns can be avoided).

A popular strategy for pivot selection rules is to detect "dense" columns and then to pivot out the columns in the diagonal positions of $D^{-2}$ corresponding to the nondense columns of $A$. A difficulty arises with the choice of a threshold density used to group columns of $A$ into the sparse and the dense parts in (4.6). A fixed threshold value approach works well only in a case when dense columns are easily identifiable. Whenever more complicated sparsity structure appears in $A$, a more sophisticated heuristic is needed. Maros and Mészáros [70] give a detailed analysis of this issue. Instead of (4.6), they consider the following partition of the LP constraint matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \[ A_{21} & A_{22} \end{bmatrix},$$

where $A_{11}$ is supposed to be very sparse and additionally it is assumed to create a sparse adjacency structure $A_{11}A_{11}^T$, $A_{12}$ is a presumably small set of "difficult" columns, e.g., dense columns or columns referring to free variables, and $[A_{21}A_{22}]$ is a set of "difficult" rows. An efficient heuristic to find such a partition is given in [70]. Therefore (4.2) becomes

$$\begin{bmatrix} -D_1^{-2} & 0 & A_{11}^T & A_{21}^T \\ 0 & -D_2^{-2} & A_{12}^T & A_{22}^T \\ A_{11} & A_{12} & 0 & 0 \\ A_{21} & A_{22} & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta y_1 \\ \Delta y_2 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ h_1 \\ h_2 \end{bmatrix}.$$  

Elimination of $D_1^{-2}$ causes very limited fill-in and reduces the matrix to

$$\begin{bmatrix} -D_2^{-2} & A_{12}^T & A_{22}^T \\ A_{12} & A_{11}D_1^2A_{11}^T & A_{11}D_1^2A_{21}^T \\ A_{22} & A_{21}D_1^2A_{11}^T & A_{21}D_1^2A_{21}^T \end{bmatrix}.$$
The elimination of the $D_2^{-2}$ block should be delayed after all attractive pivot candidates from $A_{11}D_1^2A_{11}^T$ and $A_{21}D_1^2A_{21}^T$ blocks are exploited. Fill-in can be reduced dramatically by pivoting out the dense columns of $A$ as late as possible. The normal equations approach makes no such a distinction and pivots out both $D_1^{-2}$ and $D_2^{-2}$ blocks.

This technique for delaying the elimination of dense columns is algebraically equivalent to the well-known Schur-complement technique for handling dense columns. Observe that the normal equations

$$
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
D_1^2 & 0 \\
0 & D_2^2
\end{bmatrix}
\begin{bmatrix}
A_{11}^T & A_{12}^T \\
A_{12}^T & A_{22}^T
\end{bmatrix}
\begin{bmatrix}
\Delta y_1 \\
\Delta y_2
\end{bmatrix}
= 
\begin{bmatrix}
g_1 \\
g_2
\end{bmatrix}
$$

(4.15)

can be replaced with the following system

$$
\begin{bmatrix}
A_{11}D_1^2A_{11}^T & A_{11}D_1^2A_{21}^T & A_{12}D_2 \\
A_{21}D_1^2A_{11}^T & A_{21}D_1^2A_{21}^T & A_{22}D_2 \\
D_2A_{12}^T & D_2A_{22}^T & -I
\end{bmatrix}
\begin{bmatrix}
\Delta y_1 \\
\Delta y_2 \\
\Delta \nu
\end{bmatrix}
= 
\begin{bmatrix}
g_1 \\
g_2 \\
0
\end{bmatrix}
$$

(4.16)

in which all "difficult" columns are handled as a symmetric rank-k update of an "easy" part (cf.(4.7))

$$
\begin{bmatrix}
A_{11} \\
A_{12}
\end{bmatrix}
D_1^2[A_{11}^T & A_{12}^T] + 
\begin{bmatrix}
A_{12}D_2 \\
A_{22}D_2
\end{bmatrix}
[D_2A_{12}^T & D_2A_{22}^T].
$$

(4.17)

We can find that the matrix involved in the system (4.16) has exactly the same sparsity pattern as that in (4.14).

In the implementation of an augmented system approach, to save on the expensive analysis phase, the pivot order is reused in subsequent IPM iterations and only occasionally updated when the numerical properties of the Newton equation matrix have changed considerably. In every IPM iteration, the augmented system approach tends to require fewer arithmetic operations than the normal equations approach to solve for $(\Delta x, \Delta y)$, because it computes sparser factors. Summing up, the augmented system approach has the following advantages:

1. Good accuracy properties.
2. Easily generalizable to exploit the sparsity of KKT systems arising in nonseparable quadratic programming and linear complementarity problems.

3. Naturally handles free variables. If \( x_j \in (-\infty, +\infty) \), then \( D_j^{-2} \) in (4.2) in replaced by zero.

4. Dense columns of \( A \) do not degrade its efficiency, do not lead to significant fill-in.

It has two important disadvantages:

1. It is more complicated to implement than the normal equations approach.

2. The analysis and factorize phases cannot be separated and the factorization is more expensive than in the normal equations approach.

Computational practice [4] shows that both methods are important and have their own advantages. It would be beneficial to have both of them implemented as well as to have an analyzer that is able to determine which of them should be used [70].

### 4.2 The Adaptive Approach

Wang et al. [107] propose an efficient adaptive approach for solving the Newton equations in IPM. They use an adaptive automated procedure for determining whether to use a direct or iterative solver, whether to reinitialize or update the preconditioner, and how many updates to apply. This algorithm exploits the advantages of both the direct and the iterative methods and avoids the disadvantages of both of them.

This algorithm uses normal equations to solve the Newton equations. The ideas in this algorithm can be extended to the augmented system. Now we give a brief description of Wang's algorithm.

In the first iteration of the algorithm, the Newton equations are solved using a direct method. Starting from the second iteration, the algorithm uses a preconditioned conjugate gradient method. The preconditioner for each iteration is determined by factoring the current
matrix $K = AD^2A^T$ or by updating the current preconditioner. The determination of whether to update the current preconditioner or refactor the matrix $AD^2A^T$ to obtain a new preconditioner based on the cost of determining the previous search direction or the predicted cost of determining the current iteration. If either of the two costs is high, the algorithm reinitializes the preconditioner by factoring the current matrix $K = AD^2A^T$. Otherwise, it updates the current preconditioner. The number of Cholesky updates is changed adaptively over the course of the algorithm in order to improve efficiency. It is increased if the previous search direction took many iterations and decreased if the previous search direction took a very small number of iterations.

After computing the preconditioner, the algorithm solves the normal equations using the preconditioned conjugate gradient method. It gives a maximum number of iterations allowed. If the preconditioned conjugate gradient iteration number exceeds the maximum number, then the current preconditioner is abandoned and a new preconditioner is determined by Cholesky factorization. If this happens twice, the iterative method is not suitable and the algorithm switch to a direct method.

The iterative method will continue until the relative duality gap is small enough. In this situation, the iterates are close to the optimal solution and accuracy requirements are high. The elements in matrix $D$ vary significantly and make the matrix $K = AD^2A^T$ very ill-conditioned. The Cholesky factorization of $K$ may not generate a good preconditioner, even if stable methods are used. For all of these reasons, a direct method is used to determine the final search direction.

It also switches to a direct method when a Cholesky factorization has a zero on the diagonal (i.e. the diagonal of the preconditioner is singular).

The algorithm prefers to use the preconditioned conjugate gradient method in early iterations of IPM because of the following advantages.

1. The conjugate gradient method has better stability than the direct solver. Convergence can be achieved even if the factorization is quite inaccurate.

2. The storage requirement of the preconditioned conjugate gradient method is quite low,
Chapter 4. Solving the Newton Equations

amounting to a few vectors of length $m$.

3. Since the accuracy requirements for the search direction in the beginning iterations of the IPM are quite low, only a few conjugate gradient iterations are required.

4. A good preconditioner may dramatically accelerate the convergence rate and gain great computational saving.

Computational results in [107] demonstrated that this algorithm can enhance performance of interior point algorithms on large sparse problems.
Chapter 5

Presolve Analysis

Many LP problems can be simplified to a form that is faster to solve. It is beneficial to implement a presolve procedure in an LP solver to detect redundancy and to remove it. A properly formulated LP problem must contain as few variables, constraints, and nonzeros as possible, it must be well-scaled and the constraints must be linearly independent.

The issue of presolve analysis has been addressed by many authors [1], [17], [68], [3], [101]. Andersen [3] and Gondzio [44] proposed the presolve procedures for IPM.

We consider the primal and dual LP problems (2.50) and (2.52).

The computational complexity of IPM algorithms is dependent on the number of constraints \( m \) and variables \( n \) in (2.50). The sparsity of \( A \) is more important because only nonzeros in \( A \) are stored and the work is dependent on the number of nonzeros. These require that the presolve procedure should reduce the size of \( A \) without creating any new nonzeros in \( A \).

The presolve procedures for IPM can be categorized into three classes:

1. Reduction of problem size.
2. Improving sparsity of the problem.
3. Removing linear dependent rows and columns.

5.1 Reduction of Problem Size

5.1.1 Simple Presolve Methods

- An empty row:

  \[ \exists i : a_{ij} = 0, \forall j. \]  

  (5.1)
Chapter 5. Presolve Analysis

An empty row must either be redundant or infeasible.

- An empty column:
  \[ \exists j : a_{ij} = 0, \forall i. \]  
  \( (5.2) \)

Dependent on the bounds on variable \( x_j \) and its objective coefficient \( c_j \), variable \( x_j \) is fixed at one of its bounds or the problem is unbounded.

- An infeasible variable:
  \[ \exists j : l_j > u_j. \]  
  \( (5.3) \)

The problem is trivially infeasible.

- A fixed variable:
  \[ \exists j : l_j = u_j. \]  
  \( (5.4) \)

Variable \( x_j \) can be substituted out of the problem.

- A singleton row:
  \[ \exists (i, k) : a_{ij} = 0, \forall j \neq k, a_{ik} \neq 0. \]  
  \( (5.5) \)

An equality type singleton row fixes variable \( x_k \) at level \( b_i/a_{ik} \). If \( b_i/a_{ik} < l_k \) or \( b_i/a_{ik} > u_k \), then the problem is infeasible. The elimination of an equality type singleton row may create new singleton rows and the process is repeated.

An inequality type singleton row introduces a new bound on an appropriate variable. Such a bound can be redundant, tighter than one already existing or infeasible.

5.1.2 Forcing and Redundant Constraints

The simple bounds on the primal variables (2.50) are exploited to detect forcing and redundant constraints.

We define the following lower and upper limits for a constraint \( i \)

\[ b_i = \sum_{j \in N_i} a_{ij} u_j, \]  

\( (5.6) \)
and
\[ \bar{b}_i = \sum_{j \in P_i} a_{ij} u_j, \quad (5.7) \]
where \( P_i = \{ j : a_{ij} > 0 \} \) and \( N_i = \{ j : a_{ij} < 0 \} \).

We have
\[ b_i \leq \sum_j a_{ij} x_j \leq \bar{b}_i, \quad (5.8) \]
for any \( x_j \) that satisfies \( 0 \leq x_j \leq u_j \).

We now consider the "less than or equal to" constraint
\[ \sum_j a_{ij} x_j \leq b_i. \quad (5.9) \]
A similar analysis applies to a "greater than or equal to" constraint.

The following four possibilities may occur.

- An infeasible constraint:
  \[ \exists i : b_i < b_{\bar{i}}. \quad (5.10) \]
  The problem is infeasible.

- An redundant constraint:
  \[ \exists i : \bar{b}_i \leq b_i. \quad (5.11) \]
  Constraint \( i \) is redundant, row \( i \) can be eliminated.

- A forcing constraint:
  \[ \exists i : \bar{b}_i = b_i. \quad (5.12) \]
  If \( \bar{b}_i = b_i \) then due to linearity the only feasible value of \( x_j \) is 0 if \( j \in P_i (j \in N_i) \).
  Therefore, we can fix all variables in the \( i \)th constraint.

- An nonconstructive constraint:
  \[ \exists i : b_{\bar{i}} < b_i < \bar{b}_i. \quad (5.13) \]
  Constraint \( i \) cannot be eliminated. This case is nonconstructive.
5.1.3 Tightening Variable Bounds and Implied Free Variables

The nonconstructive case

\[ b_i < b_i < \bar{b}_i \]  \hspace{1cm} (5.14)

can often be used to generate implied bounds on variables involved by (5.9).

Equation (5.6) and (5.9) imply

\[ x_k \leq u'_k = (b_i - b_i)/a_{ik}, \forall k \in P_{i*}, \]  \hspace{1cm} (5.15)

and

\[ x_k \geq l'_k = u_k + (b_i - b_i)/a_{ik}, \forall k \in N_{i*}. \]  \hspace{1cm} (5.16)

These new bounds can be constructive (tighten appropriate bound), redundant or contradictory to the existing bound.

When an implied finite bound is imposed on a free variable, the free variable does not have to be split and represented as the difference of two nonnegative variables.

If an upper bound in (5.6) is infinite, i.e. \( u_k = +\infty \) for some \( k \in N_{i*} \), then \( b_i \) is infinite. There is still an implied bound on \( x_k \). Gondzio[44] gives

\[ x_k \geq l'_k = (b_i - \sum_{j \in N_{i*} - \{k\}} a_{ij}u_j)/a_{ik}, \]  \hspace{1cm} (5.17)

• Implied free variables:

If \( 0 \leq l'_k \leq u'_k \leq u_k \), then the original bound \( 0 \leq x_k \leq u_k \) can be removed and \( x_k \) becomes free.

• A free singleton column:

\[ \exists (k,j) : (a_{ij} = 0, \forall i \neq k, a_{kj} \neq 0) \land (l_j = -\infty) \land (u_j = +\infty). \]  \hspace{1cm} (5.18)

For equality constraints, substitute \( x_j = (b_k - \sum_{q \neq j} a_{kj}x_q)/a_{kj} \) in to the objective function, and the constraint \( k \) can be eliminated. Thus one constraint and one variable are removed from the problem without generating any fill-ins in \( A \), although the objective function is modified.
Chapter 5. Presolve Analysis

- An implied free singleton column:

\[
\exists (k, j) : (a_{ij} = 0, \forall i \neq k, a_{kj} \neq 0) \land (0 \leq l'_j \leq u'_j \leq u_j)
\]  \hspace{1cm} (5.19)

where \( l'_j \) and \( u'_j \) are computed according to (5.15) and (5.16) or (5.17). Then the variable \( x_j \) can be treated as a free column singleton and the constraint \( k \) and variable \( x_j \) are removed from the problem.

5.1.4 Bounds on Shadow Prices and Dominated Variables

All presolve procedures discussed in the previous section are concerned with the analysis of a primal formulation (2.50). The similar analysis can be used on the dual problem (2.52).

If the upper bound of a primal variable is infinite, we get

\[
\text{if } u_j = +\infty \text{ then } w_j = 0. \hspace{1cm} (5.20)
\]

The associated constraint in (2.52) becomes an inequality

\[
a^T_{*j} y \leq c_j. \hspace{1cm} (5.21)
\]

If \( a_{*j} \) is a singleton column with an entry \( a_{kj} \), that is

\[
\exists (k, j) : (a_{ij} = 0, \forall i \neq k, a_{kj} \neq 0), \hspace{1cm} (5.22)
\]

then the inequality (5.21) become a singleton row for the dual problem (2.52) and can be used to produce a bound on the shadow price \( y_k \).

Assume all dual variables \( y \) have explicit (possibly infinite) bounds

\[
p_i \leq y_i \leq q_i, \quad i = 1, 2, \cdots, m. \hspace{1cm} (5.23)
\]

Define the lower and upper limits for all dual constraints as follows

\[
c_j = \sum_{i \in P_{*j}} a_{ij} p_i + \sum_{i \in N_{*j}} a_{ij} q_i, \hspace{1cm} (5.24)
\]
and
\[
\overline{c}_j = \sum_{i \in P^*_{j}} a_{ij} q_i + \sum_{i \in N^*_{j}} a_{ij} p_i,
\]
where \(P^*_{j} = \{ i : a_{ij} > 0 \}\) and \(N^*_{j} = \{ i : a_{ij} < 0 \}\).

Any feasible \(y\) satisfies
\[
c_j \leq \sum_i a_{ij} y_i \leq \overline{c}_j.
\] (5.26)

The following possibilities may occur:

1. \(c_j < \overline{c}_j\) (5.21) cannot be satisfied, the problem is dual infeasible.

2. \(\overline{c}_j < c_j\) The induced cost \(z_j\) is strictly positive, so the variable \(x_j\) is dominated. It can be fixed at its lower bound \(x_j = 0\) and eliminated from the primal problem. The dual constraint \(j\) becomes redundant.

3. \(\overline{c}_j = c_j\) (5.21) is always satisfied with \(z_j \geq 0\).

4. \(c_j \leq c_j < \overline{c}_j\) The variable \(x_j\) cannot be eliminated.

If the third case \(\overline{c}_j = c_j\) occurs, the variable \(x_j\) is called weakly dominated. It is claimed in [3] and [44] that under some additional conditions the variable can be eliminated.

If \(c_j < c_j < \overline{c}_j\), we can apply the same technique as in last section to derive new bounds on the dual variables from (5.21),(5.24) and (5.26), we get:
\[
y_k \leq q_k' = p_k + (c_j - \overline{c}_j)/a_{kj}, \quad \forall k \in P^*_{j},
\]
and
\[
y_k \geq p_k' = q_k + (c_j - \overline{c}_j)/a_{kj}, \quad \forall k \in N^*_{j},
\]
for \(a_{kj} \neq 0\).

If the new bound improves the older one, we call it constructive. If it is redundant, we omit it. Finally, if it contradicts the older one, the problem is dual infeasible.

The new bounds on \(y\) are used to recompute \(c_j\) and \(\overline{c}_j\) which in turn are used to detect more dominated columns.
5.2 Improving Sparsity of the Problem

5.2.1 Making the Matrix \( A \) Sparser

For solving the Newton equation system, both the normal equations and the augmented system are strongly influenced by the sparsity structure of \( A \). Therefore, we want to make the matrix \( A \) as sparse as possible. This Sparsity Problem (SP) can be described as: try to find a nonsingular matrix \( M \in \mathbb{R}^{m \times m} \) such that the matrix \( MA \) is the sparsest possible. Unless some simplifying assumptions are added, this problem is NP-hard[54]. Some efficient heuristics are proposed in [21]. These methods are relatively expensive to reduce the number of nonzeros of \( A \).

Gondzio[44] gives an considerably simpler heuristic for improving the sparsity of \( A \). He analyze every equality type row, say, \( a_i \), and to look for all LP constraints with the sparsity pattern being the superset of \( a_i \). If \( a_{k*} \) is a superset of \( a_i \), then for any real \( \alpha \), \( a'_{k*} = a_{k*} + \alpha a_i \) has the same sparsity pattern as \( a_{k*} \). If \( a_{k*} \) is replaced by \( a'_{k*} \) and \( b_k \) is replaced by \( b'_k = b_k + \alpha b_i \), the changed linear program is clearly equivalent to the original one.

An appropriate choice of \( \alpha \) can cause elimination of at least one nonzero entry from \( a_{k*} \), or more, thus producing a sparser \( a'_{k*} \).

Adler et al.[1] also gives a very simple and fast heuristic for improving the sparsity of \( A \).

5.2.2 Splitting Dense Columns

Sometimes even very sparse \( A \) may produce relative dense factors in the normal equations or the augmented system. If dense columns exist in \( A \), the factors of normal equations can have a lot of fill-ins. Therefore if the number of dense columns is not excessive, we can split them into shorter pieces.

A dense column \( d \) of \( A \) with \( n_d \) nonzero elements is replaced with a set of columns \( d_1, d_2, \ldots, d_k \) such that

\[
d = \sum_{i=1}^{k} d_i.
\]  

(5.29)
Adding the linking constraints

\[ x_d^i - x_d^{i+1} = 0, \quad i = 1, 2, \ldots, k - 1, \quad (5.30) \]

to force the variables \( x_d^1, x_d^2, \ldots, x_d^k \) associated with all the \( d \) to be the same. We have no longer large dense block of size \( n_d \times n_d \) created by the dense column \( d \). Instead, the new adjacency structure \( \tilde{A} \tilde{\Theta} \tilde{A}^T \) of the transformed problem contains \( k \) smaller dense windows resulting from pieces \( d_i \), for which we pay by adding \( k - 1 \) constraints (5.30).

5.3 Removing Linearly Dependent Rows and Columns

5.3.1 Linearly Dependent Rows

Linearly dependent rows in \( A \) are undesirable because in an interior-point algorithm a certain linear system must be solved in each iteration. If \( A \) contains linearly dependent rows, the linear system has a singular matrix and does not have a unique solution. To find all the linearly dependent rows in \( A \) is computational too expensive. Therefore, most IPM codes try to find only simple linear dependent rows.

One simple form of linear dependent rows is called duplicate row by Tomlin and Welch[101]. The definition is

\[ \exists(i, k) : a_{ij} = \nu a_{kj}, \quad i \neq k. \quad (5.31) \]

The heuristic of section 3.6 to make \( A \) sparser naturally detects this situation if at least one of the constraints \( i \) and \( k \) is of the equality type.

It must be noticed that due to primal degeneracy, even if the original \( A \) has full row rank, the normal equations matrix becomes rank deficient when the optimum is approached. Therefore, it is necessary to correct small rank deficiency of \( A \) in the optimization procedure, not only in the presolve procedure.
5.3.2 Duplicate Columns

We define columns $j$ and $k$ to be duplicate if they satisfy

$$
\exists(j,k) : a_{ij} = \nu a_{ik}, \quad j \neq k, \forall i,
$$

where $\nu$ is a scalar multiplier. We have that

$$
z_j^* - w_j^* = c_j - \sum_i y_i^* a_{ij}
= c_j - \nu \sum_i y_i^* a_{ik}
= c_j + \nu (z_k^* - w_k^* - c_k)
= c_j - \nu c_k + \nu (z_k^* - w_k^*).
$$

- Fixing a duplicate column:

$$
c_j - \nu c_k \neq 0
$$

In this case, under some additional assumptions, one of the columns dominates the other. If $c_j > \nu c_k$ then

$$
z_j^* - w_j^* > \nu (z_k^* - w_k^*).
$$

Now, if $u_k = +\infty$ (i.e. $w_k^* = 0$) and $\nu > 0$, or $l_k = -\infty$ (i.e. $z_k^* = 0$) and $\nu < 0$ then (5.35) yields

$$
z_j^* - w_j^* > 0.
$$

Variable $x_j$ can then be fixed to its lower bound $l_j$. If $l_j = -\infty$ then the problem is dual infeasible.

An analogous analysis applies to the opposite case $c_j < \nu c_k$.

- Replacing two duplicate columns by one:

$$
c_j - \nu c_k = 0
$$
Let us study the primal problem (2.50), we have

\[
\begin{align*}
\text{minimize} & \quad \cdots + c_j x_j + c_k x_k + \cdots, \\
\text{subject to} & \quad \cdots + a_j x_j + a_k x_k + \cdots = b, \\
& \quad l_j \leq x_j \leq u_j, \\
& \quad l_k \leq x_k \leq u_k. 
\end{align*}
\]

(5.38)

Define

\[
x_k = -\nu x_j + x'_k,
\]

(5.39)

substitute (5.39) into (5.38) we obtain

\[
\begin{align*}
\text{minimize} & \quad \cdots + c_k x'_k + \cdots, \\
\text{subject to} & \quad \cdots + a_k x'_k + \cdots = b, \\
& \quad l_j \leq x_j \leq u_j, \\
& \quad l_k \leq x'_k - \nu x_j \leq u_k. 
\end{align*}
\]

(5.40)

Bounds of \(x'_k\) depend on the sign of \(\nu\):

\[
\begin{align*}
\text{if} \quad \nu > 0 & \quad \text{then} \quad l'_k = l_k + \nu l_j \quad \text{and} \quad u'_k = u_k + \nu u_j, \\
\text{if} \quad \nu < 0 & \quad \text{then} \quad l'_k = l_k + \nu u_j \quad \text{and} \quad u'_k = u_k + \nu l_j.
\end{align*}
\]

(5.41) (5.42)

Therefore, the duplicate column procedure removes variable \(x_j\) from the problem and modifies the bounds on variable \(x_k\) according to (5.41) and (5.42).

5.4 Postsolve

After presolving the original problem (2.50), the reduced problem is solved by an IPM algorithm. In general, the optimal solution to the reduced problem is in different variables than the original problem. Therefore, a restoration procedure is needed to “undo” the presolving. To perform the “undo” procedure efficiently, one has to store some trace information on all presolve modifications done to the primal and dual variables, respectively. Andersen[3] and Gondzio [44] gave detailed description about the postsolve procedure.
Chapter 6

Solving the Fractional \((g,f)\)-factor Problem

A \((g,f)\)-factor of a graph \(G\) is a subgraph of \(G\) whose valencies are bounded between \(g\) and \(f\). More formally, let \(G = (V(G), E(G))\) be a graph on \(m\) vertices with multiple edges and loops allowed, let \(\lambda_{ij}\) denote the multiplicity of the edge \((i,j) \in E(G)\) and let \(d_G(i)\) denote the degree of vertex \(i\) in \(G\). A loop counts 2 towards the degree. Let \(g = (g_i : i \in V(G))\) and \(f = (f_i : i \in V(G))\) be vectors of positive integers satisfying

\[
\forall i \in V(G): \quad 0 \leq g_i \leq f_i \leq d_G(i).
\] (6.1)

A \((g,f)\)-factor is a subgraph \(H\) of \(G\) with

\[
\forall i \in V(G): \quad g_i \leq d_H(i) \leq f_i.
\] (6.2)

If we allow the subgraph \(H\) to have fractional edges, i.e., a real vector \(\chi = (\chi_{ij} : (i,j) \in E(G))\) satisfying \(0 \leq \chi_{ij} \leq \lambda_{ij}\), then \(d_H(i) = \sum_{(i,j) \in E(H)} \chi_{ij}\). In this case \(H\) is a fractional \((g,f)\)-factor of \(G\).

The fractional \((g,f)\)-factor problem can be solved using network flow algorithms[6]. In this chapter, we use IPM to solve the fractional \((g,f)\)-factor problem. We will exploit the special structure of the fractional \((g,f)\)-factor problem and propose two IPM algorithms to take advantage of the special structure of the fractional \((g,f)\)-factor problem. In Section 6.1 we develop Choi’s parallel interior point algorithm for network flow problem[23] to the algorithm for the fractional \((g,f)\)-factor problem. In Section 6.2 we propose a preconditioned conjugate gradient based implementation of IPM which uses an efficient preconditioner for the fractional \((g,f)\)-factor problem.
There are several different ways to formulate the fractional \((g, f)\)-factor problem as a linear programming problem. In order to make the corresponding algorithm in Section 6.1 and Section 6.2 more efficient, we use two different linear programming models of the fractional \((g, f)\)-factor problem in these two sections. The model we use in Section 6.1 has relatively fewer variables and is good for the parallel algorithm. The model we use in Section 6.2 has more variables, but for this model we can use a very efficient preconditioner and therefore make the algorithm more efficient for solving the fractional \((g, f)\)-factor problem.

### 6.1 The Parallel IPM for the Fractional \((g, f)\)-factor Problem

The fractional \((g, f)\)-factor problem can be formulated as the following linear program:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{m} l_i + \sum_{i=1}^{m} h_i, \\
\text{subject to} & \quad -\sum_{(i,j) \in E(G)} \chi_{ij} - l_i \leq -g_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad \sum_{(i,j) \in E(G)} \chi_{ij} - h_i \leq f_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad 0 \leq \chi_{ij} \leq \lambda_{ij}, \quad \text{for } (i,j) \in E(G), \\
& \quad l_i \geq 0, \quad \text{for } i = 1, \ldots, m, \\
& \quad h_i \geq 0, \quad \text{for } i = 1, \ldots, m,
\end{align*}
\]

(6.3)

where \(\chi = (\chi_{ij} : (i,j) \in E(G)) \in \mathbb{R}^n\), \(g, f \in \mathbb{R}^m\), \(l = (l_i : i \in V(G)) \in \mathbb{R}^m\) and \(h = (h_i : i \in V(G)) \in \mathbb{R}^m\) (\(l_i\) and \(h_i\) represent the lower and upper deficiency of vertex \(i\) respectively). Here we try to reduce the deficiency to zero.

In matrix notation, the above LP problem can be formulated as the following form:

\[
\begin{align*}
\text{minimize} & \quad c^T x, \\
\text{subject to} & \quad A x \leq b, \\
& \quad \chi_{ij} \leq \lambda_{ij}, \quad \text{for } (i,j) \in E(G), \\
& \quad x \geq 0,
\end{align*}
\]

(6.4)
where

\[
\tilde{A} = \begin{bmatrix}
-A & -I & 0 \\
A & 0 & -I
\end{bmatrix} \in \mathcal{R}^{2m \times (n+2m)},
\]

\[
b = \begin{bmatrix}
-g^T & f^T
\end{bmatrix}^T \in \mathcal{R}^{2m},
\]

\[
x = \begin{bmatrix}
\chi^T & l^T & h^T
\end{bmatrix}^T \in \mathcal{R}^{n+2m},
\]

\[
c = \begin{bmatrix}
0_n^T & e_{2m}^T
\end{bmatrix}^T \in \mathcal{R}^{n+2m},
\]

$0_n$ denotes the vector of all zeros in $\mathcal{R}^n$, $e_{2m}$ denotes the vector of all ones in $\mathcal{R}^{2m}$, and $\tilde{A}$ is the $m \times n$ node-arc incidence matrix of the graph $G = (V(G), E(G))$, i.e., for each arc $(i, j) \in E(G)$, there is an associated column in matrix $\tilde{A}$ with exactly two nonzero entries: an entry 1 in row $i$ and an entry 1 in row $j$.

Choi and Goldfarb propose a parallel interior point algorithm in [23]. They show that their algorithm can exploit various special structures of linear programs. We develop this algorithm for our fractional $(g, f)$-factor problem by taking advantage of the special structure of the fractional $(g, f)$-factor problem. For convenience, we reproduce Choi's resultant algorithm in Section 6.1.1 and 6.1.2. In Section 6.1.2, we suggest using George's algorithm[34] to solve the tridiagonal linear equations system in order to save the required storage for the algorithm. This suggestion is not mentioned in [23]. In Section 6.1.3, the layering routine is proposed by Choi and Goldfarb[23] for the network flow problem. We add a modifying routine to construct the staircase matrix $\tilde{A}$ for our fractional $(g, f)$-factor problem and make Choi's algorithm suitable for the fractional $(g, f)$-factor problem.

### 6.1.1 The Interior Point Algorithm

We consider a linear program of the following form:

\[
\begin{align*}
\text{minimize} & \quad c^T x, \\
\text{subject to} & \quad Ax = b, \\
& \quad Hx \leq u, \\
& \quad x \geq 0,
\end{align*}
\]
where \( c, x \in \mathcal{R}^n, b \in \mathcal{R}^m, u \in \mathcal{R}^k, A \in \mathcal{R}^{m \times n} \) and \( H \in \mathcal{R}^{k \times n} \).

Comparing (6.5) with (2.50), we can see the only difference is: in (6.5) we change the upper bound constraints \( x \leq u \) in (2.50) into \( Hx \leq u \). We study problem in this form because it captures the special structures of the fractional \((g,f)\)-factor problem. We will give detailed description in Section 6.1.3.

Adding slack variables to \( Hx \leq u \), we get

\[
\begin{align*}
\text{minimize} \quad & c^T x, \\
\text{subject to} \quad & Ax = b, \\
& Hx + s = u, \\
& x \geq 0, \\
& s \geq 0,
\end{align*}
\]

(6.6)

where \( c, x \in \mathcal{R}^n, s, u \in \mathcal{R}^k, b \in \mathcal{R}^m, A \in \mathcal{R}^{m \times n} \) and \( H \in \mathcal{R}^{k \times n} \).

The corresponding dual problem is

\[
\begin{align*}
\text{maximize} \quad & b^T y - u^T w, \\
\text{subject to} \quad & A^T y - H^T w + z = c, \\
& z \geq 0, \\
& w \geq 0,
\end{align*}
\]

(6.7)

where \( y \in \mathcal{R}^m, z \in \mathcal{R}^n \) and \( w \in \mathcal{R}^k \).

Using the same procedure as in Section 2.6, we can get the Newton search directions:

\[
\begin{align*}
\Delta y &= (AQ^{-1}A^T)^{-1}AQ^{-1}\rho(\mu), \\
\Delta x &= Q^{-1}(A^T\Delta y - \rho(\mu)), \\
\Delta z &= \mu X^{-1}e_n - Z e_n - X^{-1}Z\Delta x, \\
\Delta w &= S^{-1}(\mu e_k - SW e_k + WH\Delta x), \\
\Delta s &= -H\Delta x,
\end{align*}
\]

(6.8)

where

\[
Q = X^{-1}Z + H^T S^{-1} WH,
\]

(6.9)
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

and

\[
\rho(\mu) = \mu H^T S^{-1} e_k - \mu X^{-1} e_n - H^T W e_k + Z e_n.
\] (6.10)

Using the Sherman-Morrison-Woodbury updating formula\[41\], we can write \(Q^{-1}\) as

\[
Q^{-1} = (X^{-1} Z + H^T S^{-1} WH)^{-1} = X Z^{-1} - X Z^{-1} H^T K^{-1} H Z^{-1} X,
\] (6.11)

where

\[
K = W^{-1} S + H X Z^{-1} H^T.
\] (6.12)

In the predictor-corrector IPM, the search directions can be computed the same way.

Now we consider the case where the matrix \(H\) has a block diagonal structure. In particular, let \(H\) have \(t\) blocks

\[
H = \begin{bmatrix}
H_1 \\
& H_2 \\
& & \ddots \\
& & & H_t
\end{bmatrix},
\]

where \(H_i \in \mathbb{R}^{k_i \times n_i}, \ i = 1, \ldots, t, \ \sum_{i=1}^t k_i = k\) and \(\sum_{i=1}^t n_i = n\). Then, the matrices \(H^T S^{-1} WH\) and \(Q\) are square block diagonal matrices with \(n_i \times n_i\) diagonal blocks and \(K\) is a square block diagonal matrix with \(k_i \times k_i\) diagonal blocks.

Because of the block diagonal structure of \(Q\) and \(K\), operating involving \(Q^{-1}\) requires the solution of several smaller systems of equations, all of which can be done in parallel. Moreover, when \(H\) has a large number of rows relative to \(A\), i.e. \(k \gg m\), the speed-up obtainable by using parallel computation can be substantial.

6.1.2 Staircase Structure

IPMs work well for staircase linear programs. If matrix \(A\) is staircase then the matrix \(A A^T\) is block tridiagonal so that no fill-in occurs outside of the diagonal and off-diagonal blocks during Cholesky factorization. Thus, the search direction for such problems can be computed efficiently.
The algorithm described in Section 6.1.1 can take advantage of a constraint matrix $A^*$ that has the staircase structure

$$
A^* = \begin{bmatrix}
A_1 & A_{12} & & \\
& A_{21} & A_2 & A_{23} & \\
& & A_{32} & A_3 & A_{34} \\
& & & \ddots & \\
& & & & A_{2q+1,2q} & A_{2q+1}
\end{bmatrix} \in \mathbb{R}^{(m+k) \times n}.
$$

We assume that $A^*$ has an odd number of row blocks. We also assume that all the constraints are equalities and we shall use $m_i$ to denote the number of rows in the $i$th row block, i.e., in submatrices of the form $A_{i,..}$.

We decompose the constraint matrix into (\ref{eq:6.12}) as follows: let $A$ consist of the odd blocks of rows of $A^*$ ($A \in \mathbb{R}^{m \times n}$ where $m = \sum_{i=0}^{q} m_{2i+1}$) and let $H$ be the even blocks of rows of $A^*$ ($H \in \mathbb{R}^{k \times n}$ where $k = \sum_{i=1}^{q} m_{2i}$).

Obviously, both $A$ and $H$ are block diagonal. Therefore, $K$ and $Q^{-1}$ are both square block diagonal matrices which can be easily obtained by (6.12) and (6.11). Let the diagonal matrix $D = XZ^{-1}$ be symmetrically partitioned to conform with the column partition (block structure) of $A^*$, i.e., $D = diag\{D_1, D_{12}, D_2, D_{23}, D_3, \ldots, D_{2q,2q+1}, D_{2q+1}\}$, and let $W, S$ and $K$ be symmetrically partitioned to conform with the row partition of $H$. Then, the $q$ diagonal blocks of $K$ and the $2q + 1$ diagonal blocks of $Q^{-1}$ are

$$
K_i = W_i^{-1}S_i + H_i \bar{D}_i \bar{H}_i^T, \quad i = 1, \ldots, q,
$$

$$
(Q^{-1})_{2i+1} = D_{2i+1}, \quad i = 0, 1, \ldots, q,
$$

and

$$
(Q^{-1})_{2i} = \bar{D}_i - \bar{D}_i \bar{H}_i^T K_i^{-1} H_i \bar{D}_i, \quad i = 1, \ldots, q,
$$

where $H_i = [A_{2i,2i-1} \quad A_{2i}, \quad A_{2i,2i+1}]$ and $\bar{D}_i = diag\{D_{2i-1,2i}, D_{2i}, D_{2i,2i+1}\}$ for $i = 1, \ldots, q$.

Moreover, $AQ^{-1}A^T$ is block tridiagonal with diagonal blocks

$$
(AQ^{-1}A^T)_{i+1} = A_{2i+1,2i}D_{2i,2i+1}(I - A_{2i,2i+1}K_i^{-1}A_{2i,2i+1}D_{2i,2i+1})A_{2i+1,2i}^T
$$

for $i = 1, \ldots, q$. 


Chapter 6. Solving the Fractional (g, f)-factor Problem

\[ \begin{align*} 
+ A_{2i+1}D_{2i+1}A_{2i+1}^T + A_{2i+1,2i+2}D_{2i+1,2i+2} \\
\times (I - A_{2i+2,2i+1}K^{-1}_{i+1}A_{2i+2,2i+1}D_{2i+1,2i+2})A_{2i+1,2i+2}, 
\end{align*} \]

for \( i = 0, \ldots, q \), where the first and last term is vacuous for the cases \( i = 0 \) and \( i = q \), and off-diagonal blocks

\[ (AQ^{-1}A^T)_{i,i+1} = (AQ^{-1}A^T)_{i+1,i} = A_{2i-1,2i}D_{2i-1,2i}A_{2i,2i-1}^T A_{2i+1,2i+1}^T A_{2i+1,2i}, \quad i = 1, \ldots, q. \]

We see that splitting the staircase matrix \( A^* \) into \( A \) and \( H \) so that we can apply our algorithm allows us to compute a step direction by factorizing an \( m \times m \) symmetric block tridiagonal matrix (with diagonal blocks of size \( m_{2i+1} \times m_{2i+1}, i = 0, \ldots, q \) and off-diagonal blocks of size \( m_{2i-1} \times m_{2i+1}, i = 1, \ldots, q \)) and \( q \) smaller matrices \( K_i \) of size \( m_{2i} \times m_{2i} \). We note that the \( K_i \) factorizations can all be done in parallel. This compares with the straightforward approach which involves factorizing an \( (m + k) \times (m + k) \) symmetric block tridiagonal matrix of the form \( A^*Q^{-1}A^T \) where \( Q \) is diagonal. This matrix has \( 2q + 1 \) diagonal blocks of size \( m_i \times m_i, i = 1, \ldots, 2q + 1 \) and off-diagonal blocks of size \( m_i \times m_{i+1}, i = 1, \ldots, q \). Certainly, when \( m \) is much smaller than \( k \), substantial savings in computational effort are afforded by our approach. For this purpose, we can control the size of \( AQ^{-1}A^T \), making it small at the expense of making the blocks of \( H \) large.

To solve the linear equations system \( AQ^{-1}A^T \Delta y = t \), where \( t = AQ^{-1}p(\mu) \), we do not need to factorize the matrix \( AQ^{-1}A^T \). We know \( R = AQ^{-1}A^T \) is a block tridiagonal matrix. George[34] point out that the factorization of such a matrix \( R \) can be written in the form

\[
\begin{bmatrix}
R_{11} & R_{12} \\
R_{21} & R_{22} & R_{23} \\
\vdots & \ddots & \ddots & \ddots \\
R_{m-1} & \ddots & \ddots & \ddots & R_{mm}
\end{bmatrix}
= \begin{bmatrix}
I & R_{11}P_1^{-1} & \ldots & R_{m-1}P_{m-1}^{-1} & R_m
\end{bmatrix}
\begin{bmatrix}
P_1 & R_{12} \\
P_2 & R_{23} \\
P_3 & \ldots & \ldots & \ldots & R_m
\end{bmatrix}
\]

where

\[
P_i = R_{ii} - R_{i,i-1}P_{i-1}^{-1}R_{i-1,i}, \quad i = 2, 3, \ldots, m.
\]
Chapter 6. Solving the Fractional \((g,f)\)-factor Problem

To use this form to solve a set of equations \(R\Delta y = t\) requires the forward substitution steps

\[
\begin{align*}
    v_1 &= t_1, \\
    v_i &= t_i - R_{i,i-1}P^{-1}_{i-1}v_{i-1}, \quad i = 2, 3, \ldots, m, \\
\end{align*}
\] (6.13)

followed by the back substitution steps

\[
\begin{align*}
    \Delta y_m &= P^{-1}_m v_m, \\
    \Delta y_i &= P^{-1}_i(v_i - R_{i,i+1}\Delta y_{i+1}), \quad i = m - 1, \ldots, 1. \\
\end{align*}
\] (6.14)

It therefore suffices to keep the off-diagonal blocks of \(R\) unmodified and the diagonal blocks \(P_i\) in factorized form

\[P_i = L_i U_i, \quad i = 1, 2, \ldots, m.\]

Therefore, all fill-in is avoided in off-diagonal blocks and the required storage is saved overall.

6.1.3 The Fractional \((g,f)\)-factor Problem

To apply the IPM to the fractional \((g,f)\)-factor problem, we want to construct staircase structured constraints in (6.4). We use the following layering routine on the graph \(G = (V(G), E(G))\), to reorder the node-arc incident matrix \(\hat{A}\) into a staircase matrix.

Layering routine:

Step 1 Starting from a chosen set of nodes, say \(N_1\), perform undirected breadth first search to construct a layered graph, say with \(2q + 1\) layers.

Step 2 Let \(R_i\) be the set of arcs in layer \(i\), and \(R_{i,i+1}\) be the set of arcs between layer \(i\) and \(i + 1\). Furthermore, let \(N_i\) be the set of nodes in layer \(i\).

Step 3 Permute the node-arc incidence matrix \(\hat{A}\) of the graph \(G = (V(G), E(G))\) so that the rows of \(\hat{A}\) are ordered as \(N_1, N_2, \ldots, N_{2q+1}\), and the columns of \(\hat{A}\) are ordered as \(R_1, R_{1,2}, R_2, R_{2,3}, \ldots, R_{2q+1}\).
The node-arc incidence matrix $A$ produced by the above routine has a staircase structure, since this routine creates a layered graph, i.e., there are no arcs between layer $i$ and layer $j$ for $j \neq i + 1$ or $j \neq i - 1$. Note that the choice of $N_1$ greatly affects the number of layers. We would like to choose $N_1$ so that $q$ is large and $m = \sum_{i=0}^{q} |N_{2i+1}|$ is small. For given $N_1$, we can find the layered structure by performing breadth first search. But finding the best layered graph with respect to the above criteria is difficult because of the exponentially large number of possible starting sets $N_1$. Therefore, we will use a heuristic method to find a better (not necessary best) $N_1$.

Now we use the following modifying routine to reorder $\hat{A}$ of (6.4) into a staircase matrix based on $\hat{A}$.

**Modifying routine**

**Step 1** Order $\begin{bmatrix} -\hat{A} \\ \hat{A} \end{bmatrix}$ in (6.4) into a staircase by moving each row in the bottom matrix $\hat{A}$ to the position right after the corresponding row in the upper matrix $-\hat{A}$.

**Step 2** For each vector $R_i : i = 1, \ldots, 2q + 1$, move all the columns corresponding to $(l_k, h_k) : \forall k \in N_i$ to the position at end of each vector $R_i$.

After this modifying routine we get a new staircase matrix $\hat{A}$.

For example, the staircase matrix $\hat{A}$ of the graph in Figure 6.1 is shown in Figure 6.2.

Now applying the method described in Section 6.1.2, we decompose the staircase constraint matrix $\hat{A}$ into $\begin{bmatrix} A \\ H \end{bmatrix}$. Adding slack variables $\tau$ in the constraints corresponding to matrix $A$, therefore (6.4) becomes

minimize $\quad \tau^T \bar{x}$,

subject to $\begin{bmatrix} A & I \end{bmatrix} \bar{x} = b_1,$

$\begin{bmatrix} H & 0 \end{bmatrix} \bar{x} \leq b_2,$

$x_{ij} \leq \lambda_{ij}, \quad \text{for } (i,j) \in E(G),$

$\bar{x} \geq 0,$

(6.15)
Figure 6.1: The layered graph $G$. 

where $\bar{x} = [\chi^T \; I^T \; h^T \; r^T]^T \in \mathcal{R}^{\bar{n}+\bar{m}}$ with $\bar{n} = n + 2m$ and $\bar{m} = 2m$, $A \in \mathcal{R}^{\bar{m} \times \bar{n}}$, $H \in \mathcal{R}^{k \times \bar{n}}$, and $r \in \mathcal{R}^{\bar{m}}$. We can write (6.15) in the following form:

$$\begin{align*}
\text{minimize} & \quad c^T \bar{x}, \\
\text{subject to} & \quad \bar{A} \bar{x} = b_1, \\
& \quad H \bar{x} \leq \bar{b}_2, \\
& \quad \bar{x} \geq 0,
\end{align*}$$

(6.16)

where

$$\begin{align*}
\bar{A} &= [A \; I] \quad \in \mathcal{R}^{\bar{m} \times (\bar{n}+\bar{m})} \\
\bar{H} &= \begin{bmatrix} H & 0 \\ I & 0 \end{bmatrix} \quad \in \mathcal{R}^{(k+n) \times (\bar{n}+\bar{m})} \\
\bar{b}_2 &= [b_2^T \; \lambda_2^T] \quad \in \mathcal{R}^{k+n} \\
\bar{x} &\in \mathcal{R}^{\bar{n}+\bar{m}} \\
b_1 &\in \mathcal{R}^{\bar{m}}
\end{align*}$$

To get Newton search directions for (6.16), we need to compute

$$Q^{-1} = [\bar{X}^{-1} Z + \bar{H}^T \bar{S}^{-1} \bar{W} \bar{H}]^{-1},$$

(6.17)
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R12</th>
<th>R2</th>
<th>R23</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>N2</td>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>N3</td>
<td>6</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>1</td>
<td>-1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Figure 6.2: The staircase matrix \( \tilde{A} \) of the layered graph \( G \).

We can see

\[
\tilde{H}^T S^{-1} \tilde{W} \tilde{H} = \begin{bmatrix} H^T \begin{bmatrix} I \\ 0 \end{bmatrix} & S_1^{-1} W_1 & 0 \\ 0 & S_2^{-1} W_2 & 0 \end{bmatrix} \begin{bmatrix} H & 0 \\ I & 0 \end{bmatrix} = \begin{bmatrix} H^T S_1^{-1} W_1 H + S_2^{-1} W_2 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} H^T S_1^{-1} W_1 H + \Omega & 0 \\ 0 & 0 \end{bmatrix},
\]

where

\[
\Omega = \begin{bmatrix} S_2^{-1} W_2 & 0 \\ 0 & 0 \end{bmatrix}.
\]
Therefore
\[
Q^{-1} = \left[ \begin{bmatrix} \tilde{X}_1^{-1}Z_1 & 0 \\ 0 & \tilde{X}_2^{-1}Z_2 \end{bmatrix} + \begin{bmatrix} H^T S_1^{-1}W_1 H + \Omega & 0 \\ 0 & 0 \end{bmatrix} \right]^{-1}
\]
\[
= \begin{bmatrix} [\tilde{X}_1^{-1}Z_1 + H^T S_1^{-1}W_1 H + \Omega]^{-1} & 0 \\ 0 & [\tilde{X}_2^{-1}Z_2]^{-1} \end{bmatrix}.
\]

Let \( Q_1 = \tilde{X}_1^{-1}Z_1 + H^T S_1^{-1}W_1 H + \Omega \), thus
\[
Q_1^{-1} = \Theta - \Theta H^T K^{-1} H \Theta,
\]
where \( \Theta = (\tilde{X}_1^{-1}Z_1 + \Omega)^{-1} \) and \( K = W_1^{-1} S_1 + H \Theta H^T \).

Note that \( \Theta \) is a diagonal matrix and that \( K \) is a block diagonal matrix, therefore \( Q_1^{-1} \) is a block diagonal matrix.

We have
\[
\tilde{A}Q_1^{-1}A^T = [A \ I] \begin{bmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{bmatrix} \begin{bmatrix} A^T \\ I \end{bmatrix} = AQ_1^{-1}A^T + Q_2^{-1},
\]
where \( Q_2^{-1} = [\tilde{X}_2^{-1}Z_2]^{-1} \) is diagonal matrix. Therefore, \( AQ_1^{-1}A^T \) and \( \tilde{A}Q_1^{-1}A^T \) are block tridiagonal matrices with dimension \( \tilde{m} \). We can use the parallel algorithm which is described in Section 6.1.2 to solve the Newton search directions (6.8).

### 6.2 The Preconditioned Conjugate Gradient Algorithm

In this section, we first use the result of [7] and [8] to reduce the fractional \((g, f)\)-factor problem to a minimum cost network flow (MCNF) problem. Then we use a preconditioned conjugate gradient (PCG) based implementation of IPM to solve the MCNF problem. Resende and Veiga[89] have shown that the PCG based IPM makes the IPM very efficient for MCNF problem. The preconditioner we suggest is a combination of Mehrotra and Wang's preconditioner for MCNF problem[77] and Wang and O'Leary's adaptive preconditioner [107].

According to the results of [7] and [8], the network \( \mathcal{G} = (V, \mathcal{E}) \) for computing the fractional \((g, f)\)-factor problem is constructed as follows:
Let the nodes

\[ V = \{\text{dummy}\} \cup \{R_i : i \in V\} \cup \{S_i : i \in V\}, \]  

and the set \( E \) of arcs given as follows

\[ \forall (i,j) \in E(G) (R_i, S_j), (R_j, S_i) \in E_1, \text{ both with capacity } \lambda_{ij} \text{ and cost } 0, \]

\[ \forall i \in V(G) (\text{dummy}, R_i), (S_i, \text{dummy}) \in E_2, \text{ both with capacity } g_i \text{ and cost } -1, \]

\[ (\text{dummy}, R_i), (S_i, \text{dummy}) \in E_3, \text{ both with capacity } f_i - g_i \text{ and cost } 0, \]

\[ (\text{dummy}, R_i), (S_i, \text{dummy}) \in E_4, \text{ both with capacity } \infty \text{ and cost } 1. \]

Therefore, the set of arcs \( E = E_1 \cup E_2 \cup E_3 \cup E_4 \).

**Proposition 6.2.1** *(Anstee[7])* A minimum cost flow in \( G \) yields a minimum deficiency fractional subgraph \( H \) using

\[ \forall (i,j) \in E(G) \chi_{ij} = \frac{1}{2}(f(R_i, S_j) + f(R_j, S_i)), \]

where \( f(p,q) \) is the flow on the arc \((p,q) \in E\).

Let

\[ \chi_{ij} = \text{ the flow on arc } (R_i, S_j) \in E_1, \]

\[ lr_i = \text{ the flow on arc } (\text{dummy}, R_i) \in E_2, \]

\[ ls_i = \text{ the flow on arc } (S_i, \text{dummy}) \in E_2, \]

\[ mr_i = \text{ the flow on arc } (\text{dummy}, R_i) \in E_3, \]

\[ ms_i = \text{ the flow on arc } (S_i, \text{dummy}) \in E_3, \]

\[ hr_i = \text{ the flow on arc } (\text{dummy}, R_i) \in E_4, \]

\[ hs_i = \text{ the flow on arc } (S_i, \text{dummy}) \in E_4, \]

then the minimum cost network flow problem of \( G \) can be formulated as the following LP
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

problem,

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{m}(-lr_i + hr_i), \\
\text{subject to} & \quad \sum_{(i,j) \in E_1} x_{ij} - lr_i - mr_i - hr_i = 0, \quad \text{for } i = 1, \ldots, m, \\
& \quad \sum_{(i,j) \in E_1} x_{ij} - ls_i - ms_i - hs_i = 0, \quad \text{for } i = 1, \ldots, m, \\
& \quad \sum_{i=1}^{m} (lr_i + mr_i + hr_i) - \sum_{i=1}^{m} (ls_i + ms_i + hs_i) = 0, \\
& \quad 0 \leq x_{ij} \leq \lambda_{ij}, \quad \text{for } (i, j) \in E(G), \\
& \quad 0 \leq lr_i \leq g_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad 0 \leq ls_i \leq g_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad 0 \leq mr_i \leq f_i - g_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad 0 \leq ms_i \leq f_i - g_i, \quad \text{for } i = 1, \ldots, m, \\
& \quad hr_i \geq 0, \quad \text{for } i = 1, \ldots, m, \\
& \quad hs_i \geq 0, \quad \text{for } i = 1, \ldots, m.
\end{align*}
\]

(6.19)

To solve (6.19), if the optimal objective value is equal to \(\sum_{i=1}^{m} -g_i\), we get a fractional \((g, f)\)-factor \(H\) with the edge values given as Proposition 6.2.1. Otherwise, we get a minimum deficiency fractional subgraph \(H\).

The network flow problem (6.19) can be transformed to the following form:

\[
\begin{align*}
\text{minimize} & \quad \tilde{c}^T \tilde{x}, \\
\text{subject to} & \quad \tilde{A} \tilde{x} = \tilde{b}, \\
& \quad \tilde{x} \leq u, \\
& \quad \tilde{x} \geq 0,
\end{align*}
\]

(6.20)

where \(\tilde{A}\) is the \((2m + 1) \times (2n + 6m)\) node-arc incidence matrix of the network \(G = (\mathcal{V}, \mathcal{E})\), i.e. for each arc \((i, j) \in \mathcal{E}\) there is an associated column in matrix \(\tilde{A}\) with exactly two nonzero entries: an entry 1 in row \(i\) and an entry \(-1\) in row \(j\). \(\tilde{x}_{ij}\) denotes the flow on edge \((i, j) \in \mathcal{E}\), \(\tilde{c}_{ij}\) is the cost of transporting one unit of flow on edge \((i, j) \in \mathcal{E}\) and \(u_{ij}\) is the upper bound on flow (i.e. capacity) on edge \((i, j) \in \mathcal{E}\). For each vertex \(j \in \mathcal{V}\), \(\tilde{b}_j\) denotes the flow produced or
consumed at vertex \( j \). If \( \tilde{b}_j > 0 \), vertex \( j \) is a source. If \( \tilde{b}_j < 0 \), vertex \( j \) is a sink. Otherwise \( \tilde{b}_j = 0 \), vertex \( j \) is a transshipment vertex. In the formulation (6.20), \( \tilde{b}_j = 0 \) for all \( j \in \mathcal{V} \).

As we stated in Chapter 4, the major work in a single iteration of any IPM is related to build and update the matrix \( \tilde{A}D^2\tilde{A}^T \) and to solve the Newton equations system

\[
(\tilde{A}D^2\tilde{A}^T)\Delta y = \sigma
\]

that determines the search direction at each iteration of IPM. There are two methods to solve the Newton equations system: the \textit{direct} method and the \textit{iterative} method.

Resende and Veiga\cite{89} have shown that the \textit{direct} method performs poorly on even small instances of network flow problems. This is because that even though the node-arc incidence matrix \( \tilde{A} \) is sparse, the factorization of \( \tilde{A}D^2\tilde{A}^T \) can produce considerable fill-in, which will cause prohibitively high computational effort and storage requirements during Cholesky factorization in direct method. Therefore, the most fundamental requirement of an interior point implementation for network flows is an efficient implementation of an \textit{iterative} method to solve the Newton equations system. We pay our attention to the conjugate gradient (CG) method.

Because the matrices \( \tilde{A}D^2\tilde{A}^T \) in IPMs show a tendency to be extremely ill conditioned, a pure conjugate gradient algorithm shows hopelessly slow convergence. Additionally, the use of finite precision results in round-off errors which sometimes make it impossible for the algorithm to converge.

To remedy this we use a preconditioner, i.e., a symmetric positive definite matrix \( M \), such that the preconditioned matrix

\[
M^{-1}(\tilde{A}D^2\tilde{A}^T)
\]

is less ill-conditioned than \( \tilde{A}D^2\tilde{A}^T \). For this purpose, \( M \) should satisfy that \( M^{-1} \approx (\tilde{A}D^2\tilde{A}^T)^{-1} \) and \( M^{-1} \) is easy to compute. The preconditioner improves the efficiency of the conjugate gradient algorithm by reducing the number of iterations it takes to find a satisfiable direction. In the preconditioned conjugate gradient (PCG) algorithm, we use the determined preconditioner
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

82

Procedure \(\text{PCG}(\tilde{A}, D, \sigma, \epsilon, \Delta y)\)
1. \(\Delta y_0 = \Delta y_i\)
2. \(r_0 = \sigma - (\tilde{A}D^2\tilde{A}^T)\Delta y_0;\)
3. \(v_0 = M^{-1} r_0;\)
4. \(p_0 = v_0;\)
5. \(i = 0;\)
6. While the stopping criterion is not satisfied do:
7. \(q_i = (\tilde{A}D^2\tilde{A}^T)p_i;\)
8. \(\alpha_i = v_i^T r_i / p_i^T q_i;\)
9. \(\Delta y_{i+1} = \Delta y_i + \alpha_i p_i;\)
10. \(r_{i+1} = r_i - \alpha_i q_i;\)
11. \(v_{i+1} = M^{-1} r_{i+1};\)
12. \(\beta_i = v_{i+1}^T r_{i+1} / v_i^T r_i;\)
13. \(p_{i+1} = v_{i+1} + \beta_i p_i;\)
14. \(i = i + 1;\)
15. end do;
16. \(\Delta y = \Delta y_i\)
end \(\text{PCG};\)

Table 6.1: The preconditioned conjugate gradient algorithm to solve the following system of linear equations.

\[
M^{-1}(\tilde{A}D^2\tilde{A}^T)\Delta y = M^{-1} \sigma.
\]

The pseudo-code of PCG algorithm is given in Table 6.1.

Resende and Veiga[89] have shown that the special structure of network flow problems can be exploited to determine an easily invertible preconditioner for the PCG algorithm. They proposed a preconditioner which is a combination of diagonal and maximum weighted spanning tree preconditioners. In the earlier iterations of IPM they use the diagonal preconditioner \(M = \text{diag}(\tilde{A}D^2\tilde{A}^T)\), and switch to the maximum weighted spanning tree preconditioner when the diagonal preconditioner becomes ineffective.

- The spanning tree preconditioner
A basis in the network simplex method correspond to a spanning tree of the graph $G$. A nondegenerate minimum cost network flow problem has a unique optimal solution and a corresponding optimal spanning tree $T^*$. Let $\bar{A} = [\bar{A}_T \quad \bar{A}_N]$, where $N^* = G \setminus T^*$. It is easy to see that $De \rightarrow (D_T e, 0)$ and the matrix $\bar{A}D^2\bar{A}^T \rightarrow \bar{A}_T D_T^2 \bar{A}_T^T$, when the IPM goes to the optimal solution. Hence one may expect that $\bar{A}_TD_T^2 \bar{A}_T^T$, where $T$ is a maximum weighted spanning tree, would serve as a good preconditioner as the algorithm converges. The maximum weighted spanning tree can be identified using as weights the diagonal elements of the current scaling matrix $w = De$ where $e$ is a unit vector.

Based on Resende and Veiga’s preconditioner, Mehrotra and Wang[77] proposed a improved preconditioner as follows:

Let $T$ be a maximum weighted spanning tree of $G$, then

$$\tilde{A}D^2\tilde{A}^T = [\tilde{A}_T D_T^2 \tilde{A}_T^T + \tilde{A}_N D_N^2 \tilde{A}_N^T] \approx \tilde{A}_T D_T^2 \tilde{A}_T^T + \Lambda,$$

where $\Lambda$ is a nonnegative diagonal matrix. A simple choice of $\Lambda$ which has worked well in practice is to take

$$\Lambda = \rho \times \text{diag}(\tilde{A}_N D_N^2 \tilde{A}_N^T),$$

where $\rho$ is computed adaptively.

Mehrotra and Wang[77] proved that the Cholesky factor of the preconditioner $\tilde{A}_T D_T^2 \tilde{A}_T^T + \Lambda$ can be computed in $O(m)$ computational steps. Actually, the Cholesky factor of the preconditioner is an incomplete Cholesky factor of $\tilde{A}D^2\tilde{A}^T$. The computational results given in [77] show that Mehrotra and Wang’s preconditioner is better than Resende and Veiga’s preconditioner.

Wang and O’Leary[107] proposed an adaptive approach to determine a preconditioner. Following their approach, we use the scheme in which we first perform a procedure to determine whether to update the current preconditioner or to recompute a new preconditioner $\tilde{A}_T D_T^2 \tilde{A}_T^T + \Lambda$ for the matrix $\tilde{A}D^2\tilde{A}^T$. Next, we use the determined preconditioner to implement a PCG method.
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

The updated preconditioner is computed by updating the previous preconditioner by a small-
rank change, when \(\mu\) changes. Let \(\tilde{D}\) be the current diagonal matrix and \(D\) be the one for which
we have a preconditioner \(\tilde{A}T \tilde{D}^2 \tilde{A}T + \Lambda\) and the Cholesky factor \(L\) of the preconditioner. Define
\(\Delta D = \tilde{D}^2 - D^2\) and let \(a_i\) be the \(i\)-th column of matrix \(\tilde{A}\). Since
\[
\tilde{A} D^2 \tilde{A}^T = \tilde{A} \tilde{D}^2 \tilde{A}^T + \tilde{A} \Delta D \tilde{A}^T
\]
\[
\approx \tilde{A} T \tilde{D}^2 \tilde{A} T + \Lambda + \tilde{A} \Delta D \tilde{A}^T
\]
\[
= L L^T + \sum_{i=1}^{2n+6m} \Delta d_i a_i a_i^T,
\]
we may obtain an improved preconditioner \(\tilde{L} \tilde{L}^T\) by applying a rank-\(\alpha\) update to \(LL^T\). This
update may be computed as in [15] and [26].

The determination of whether to use the updated preconditioner or to recompute a new
preconditioner is based on the approximate cost of the previous iteration including the cost of
any updates that were made to the preconditioner. If the cost of the previous search direction
was high, we recompute a new preconditioner for the current matrix \(\tilde{A} D^2 \tilde{A}^T\). If the cost of the
previous search direction was not high, the preconditioner is obtained by updating the previous
one.

Wang and O'Leary[107] demonstrate that the adaptive preconditioner can enhance the
performance of interior point algorithms on large sparse problems.

In the PCG algorithm, the initial direction \(\Delta y_0\) is the direction \(\Delta y\) which is produced in
the previous call to the PCG algorithm, i.e., during the previous interior point iteration. This
is done with the expectation that \(D\) and \(\sigma\) change little between consecutive interior point
iterations, and consequently the direction produced in an iteration should be close to the one
produced in the previous iteration. When first time PCG is called, let \(\Delta y_0 = (0, \ldots, 0)\).

In this algorithm, we use the stopping criterion suggested by Karmarkar and Ramakrishnan
[59] and Resende and Veiga [89].

Let \(\Delta \tilde{y}\) be an approximate solution of (6.21) and
\[
\cos \theta = \frac{| \sigma^T (\tilde{A}D^2 \tilde{A}^T) \Delta \tilde{y} |}{\| \sigma \| \cdot \| (\tilde{A}D^2 \tilde{A}^T) \Delta \tilde{y} \|}.
\]
where $\theta$ is the angle between $(\bar{A}D^2\bar{A}^T)\Delta \bar{y}$ and $\sigma$.

We terminate PCG iterations when $|1 - \cos \theta| < \epsilon$, i.e., $\theta \approx 0$, where $\epsilon$ is some small tolerance (typically $\epsilon = 10^{-8}$).

This stopping criterion effectively halts the conjugate gradient algorithm when a good enough direction is on hand.

### 6.3 Computational Implementation

We use Gondzio's code HOPDM to solve the fractional $(g, f)$-factor problem. HOPDM stands for Higher Order Primal Dual Method. It is an implementation of Gondzio's multiple centrality corrections method which has been described in Section 3.3.

We use the formulation (6.3) to solve the fractional $(g, f)$-factor problem. The objective function of (6.3) represents the total deficiency for the current $x = (x_{ij} : (i, j) \in E(G)) \in \mathbb{R}^n$ value. If we can find an optimal solution in which the objective value is less than 1, then the current $x$ is a fractional $(g, f)$-factor. Otherwise the objective value will be an integer $K$ which is equal to or larger than 1. In this case, we can get a $(g, f)$-barrier (see below) and a minimum deficiency fractional subgraph $x$.

To introduce the definition of $(g, f)$-barrier, we cite the following theorem.

**Theorem 6.3.1 (Anstee[6])** There exists a fractional $(g, f)$-factor in $G$ if and only if

$$
\sum_{i \in S} f_i + \sum_{i \in T, j \in U} \lambda_{ij} \geq \sum_{i \in T} g_i \quad (6.22)
$$

for all partitions $S, T, U$ of vertex $1, 2, \ldots, m \in V(G)$.

If there is a partition $S, T, U$ violating (6.22), this partition is a $(g, f)$-barrier.

After executing HOPDM, If the objective value is equal to or larger than 1, we can get a $(g, f)$-barrier $S, T, U$ as follows:

$$
T = \{i : l_i > 0, \ \forall i \in V(G)\},
$$

$$
S = \{i : h_i > 0, \ \forall i \in V(G)\},
$$
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

\[
U = V(G) \setminus (T \cup S).
\]

In this case, the objective value \(K\) will be

\[
K = \sum_{i \in T} g_i - \sum_{i \in T, j \in T \cup U} X_{ij} - \sum_{i \in S} f_i.
\]

Actually, for all \((i, j) : i \in T, j \in T \cup U, X_{ij} \approx 1\). Therefore, \(\sum_{i \in T, j \in T \cup U} X_{ij}\) is approximately the number of edges \((i, j) : i \in T, j \in T \cup U\).

### 6.3.1 Starting Point

All interior point methods are sensitive to the starting point. The best choice of a default starting point is still an open question.

In this section we use three different starting point approaches to run the HOPDM code for fractional \((g, f)\)-factor problem and compare the results. Now we give a detailed description of these three starting point approaches.

- **Approach 1**

  For LP (2.51) and (2.52), the only requirement for the starting point of Gondzio’s multiple centrality corrections method is \(x^0 > 0, s^0 > 0, z^0 > 0, \) and \(w^0 > 0\). Gondzio generates the starting point by using the solution of a primal least squares problem,

  \[
  \begin{align*}
  \text{minimize} & \quad |x^Ts|, \\
  \text{subject to} & \quad Ax = b, \\
  & \quad x + s = u,
  \end{align*}
  \]

  and a dual least squares problem,

  \[
  \begin{align*}
  \text{minimize} & \quad |z^Tw|, \\
  \text{subject to} & \quad A^Ty + z - w = c.
  \end{align*}
  \]

  The Lagrangian for (6.23) is then

  \[
  L(x, s, y, w) = |x^Ts| - v^T(Ax - b) + t^T(x + s - u).
  \]
Chapter 6. Solving the Fractional \((g,f)\)-factor Problem

The first order optimality conditions (KKT conditions) for (6.23) are

\[ s - A^T v + t = 0, \]
\[ x + t = 0, \]
\[ A x - b = 0, \]
\[ x + s - u = 0. \]  \hfill (6.26)

We can easily solve the linear equations system (6.26).

\[ v = (AA^T)^{-1}(Au - 2b), \]
\[ x = (u - A^Tv)/2, \] \hfill (6.27)
\[ s = u - x. \]

Similarly, we can find the solution of (6.24) as

\[ y = (AA^T)^{-1}Ac, \]
\[ z = (A^Ty - c)/2, \] \hfill (6.28)
\[ w = -z. \]

The stepsizes in primal space \(d_P\) and dual space \(d_D\) are determined by

\[ \hat{d}_P = \max(-1.5 \times \min\{x_j\}, -1.5 \times \min\{s_j\}, 0), \]

and

\[ \hat{d}_D = \max(-1.5 \times \min\{z_j\}, -1.5 \times \min\{w_j\}, 0), \]

then

\[ d_P = \hat{d}_P + 0.5 \times \frac{(x + \hat{d}_P e)^T(z + \hat{d}_D e) + (s + \hat{d}_P e)^T(w + \hat{d}_D e)}{\sum_{j=1}^{n}(x_j + w_j)}, \]

and

\[ d_D = \hat{d}_D + 0.5 \times \frac{(x + \hat{d}_P e)^T(z + \hat{d}_D e) + (s + \hat{d}_P e)^T(w + \hat{d}_D e)}{\sum_{j=1}^{n}(x_j + s_j)}. \]
The starting point is then given by

\[
\begin{align*}
    x^0 &= x + dpe, \\
    s^0 &= s + dpe, \\
    z^0 &= z + dDe, \\
    w^0 &= w + dDe, \\
    y^0 &= y, \quad \text{(unaltered).}
\end{align*}
\]  

(6.29)

We can prove that the starting point satisfies \(x^0 > 0, s^0 > 0, z^0 > 0, \text{ and } w^0 > 0\).

- **Approach 2**

  Anstee gives a method for choosing the starting point in [9].

  For every vertex \(i \in V(G)\), the interval \(I_i\) and \(\bar{g}_i, \bar{f}_i\) are defined as follows

  \[
  I_i = \left[ \frac{g_i}{d_G(i)} , \frac{f_i}{d_G(i)} \right] = [\bar{g}_i, \bar{f}_i].
  \]

  We can see if a fractional subgraph \(\chi\) has \(\chi_{ij} \in I_i \cap I_j, \forall (i, j) \in E(G)\) then \(\chi\) is a fractional \((g, f)\)-factor. Therefore if \(I_i \cap I_j \neq \emptyset\), we choose \(\chi_{ij} \in I_i \cap I_j\). If \(I_i \cap I_j = \emptyset\) and \(\bar{f}_i < \bar{g}_j\), we choose \(\chi_{ij} = \bar{f}_i + \frac{1}{2}d(I_i, I_j)\), where \(d(I_i, I_j)\) is the distance between \(I_i\) and \(I_j\).

  In (6.3), \(l_i : i \in V(G)\) are determined by \(g_i - \sum_{(i,j) \in E(G)} \chi_{ij}\), and \(h_i : i \in V(G)\) are determined by \(\sum_{(i,j) \in E(G)} \chi_{ij} - f_i\).

  Now we have determined \(x^0 = [\chi^0, l^0, h^0]^T\) of (6.4).

  The starting value of the primal upper bound slack variables \(s\) of (6.4) is determined by \(s^0 = u - x^0\), where \(u\) is the upper bound vector.

  We give a threshold value \(\delta\), and adjust all \(x^0\) and \(s^0\) that are below this threshold value to the threshold value \(\delta\). This guarantees that all \(x^0\) and \(s^0\) are sufficiently bounded away from zero. This threshold value \(\delta\) is also used to bound dual variables \(z^0\) and \(w^0\) away from zero.

  For the dual problem, we set \(y^0 = 0\). \(z^0\) and \(w^0\) are determined as follows
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

For \(x_j\) unbounded:

\[
x_j^0 = \begin{cases} 
   c_j & \text{if } c_j \geq \delta \\
   \delta & \text{if } c_j < \delta
\end{cases}
\]  

(6.30)

For \(x_j\) bounded:

\[
\begin{align*}
   w_j^0 &= \delta, \quad x_j^0 = c_j + w_j^0 & \text{if } c_j \geq 0 \\
   x_j^0 &= \delta, \quad w_j^0 = x_j^0 - c_j & \text{if } c_j < 0
\end{align*}
\]  

(6.31)

- Approach 3

We give a threshold value \(\delta\). For primal problem, we set \(x^0 = \delta\) and \(s^0 = \delta\). The dual variables are determined the same way as approach 2.

6.3.2 Test Problems

A merely random \((g, f)\)-factor problem is not good. To show this, let us review some available results. We consider a random graph \(G\) on \(n\) vertices where each vertex chooses randomly a set of \(r\) neighbors.

Pósa [86] proved that a random graph \(G\) with \(f(n) = cn\log n\) edges is Hamiltonian with probability tending to 1 if \(c > 3\). Obviously, if a graph of even order is Hamiltonian, then the graph has two disjoint 1-factors. In [93] Shamir and Upfal found a sufficient condition for a 1-factor. They proved that if the minimum vertex degree reaches 6, almost every random graph \(G\) has a 1-factor. Komlós and Szemerédi[64] proved that when a random graph is chosen by choosing edges independently with the same probability, Hamiltonian cycles appear (in a probabilistic sense) at the same time that the minimum vertex degree reaches 2. This requires about \(1/2n\log n + n\log\log n\) edges. This means that if in a random graph \(G\) each valency is at least 2, then the graph \(G\) contains a Hamiltonian cycle “almost surely”.

From these results, we get the conclusion that a random graph with enough large number of edges almost surely has a 1-factor.

We expect that in a similar way a random graph \(G\) with “large enough” degree almost surely has a \((g, f)\)-factor. A proof could involve showing that the cut conditions (6.22) are almost
Chapter 6. Solving the Fractional \((g,f)\)-factor Problem

surely satisfied. Therefore in order to generate "interesting" \((g,f)\)-factor problems, we do not think it is appropriate to use a wholly random graph.

We have generated problems that either contain a \((g,f)\)-barrier or something close a \((g,f)\)-barrier, i.e. a "small" cut. To do this we choose a partition \(S,T,U\) of the vertex set \(V(G)\) at random and then introduce edges and the data values \(f_i, g_i\) for every vertex \(i\) in \(V(G)\) to make the partition \(S,T,U\) either violate or nearly violate (6.22), namely equality in (6.22).

6.3.3 Computational Results

According to [9], we allow an error of size \(< 1\) for our fractional \((g,f)\)-factor problem. In our algorithm, when the objective value \(< 1\), we stop the algorithm and get a fractional \((g,f)\)-factor. Otherwise, for a given optimality tolerance \(\varepsilon\), when the relative duality gap satisfies

\[
\frac{c^T x - b^T y + u^T w}{1 + |b^T y - u^T w|} < \varepsilon,
\]

we stop the algorithm. In this case, the algorithm converge to an optimal solution with an integral objective value \(K > 1\) and we get a \((g,f)\)-barrier.

In this section, we test fourteen \((g,f)\)-factor problems using the three starting point approaches and compare the computational results. Problem 1 to Problem 10 are feasible (having fractional \((g,f)\)-factor) or infeasible problems with small size. Problem 11, 12 and 13 are disjoint union of Problem 1 and Problem 2, Problem 3 and Problem 4, and Problem 3 and Problem 5, respectively. They are the following three types of union: infeasible and infeasible, feasible and infeasible, and feasible and feasible. Problem 14 is a disjoint union of Problem 1 and Problem 2 plus 9 edges, so that there is a cut of capacity 0, i.e. a cut satisfying equality in (6.22).

Table 6.2 gives the sizes of the fourteen \((g,f)\)-factor problems and their optimal objective values. Column 1 and Column 2 of this table indicate the numbers of vertices and edges in the original graph. Column 3 indicates the number of constraints in each problem. Column 4 indicates the number of variables in (6.4) and column 5 indicates the number of variables after adding slack variables to (6.4). Column 6 gives the number of nonzero elements in the
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

Table 6.2: Original problem size

| Problems | \(|V(G)|\) | \(|E(G)|\) | Rows | Structural cols. | Columns | Nonzeros | Opt. values |
|----------|----------|----------|------|-----------------|---------|----------|-------------|
| 1        | 24       | 55       | 48   | 103             | 151     | 316      | 5           |
| 2        | 27       | 70       | 54   | 124             | 178     | 388      | 4           |
| 3        | 22       | 45       | 44   | 89              | 133     | 268      | 0           |
| 4        | 30       | 70       | 60   | 130             | 190     | 400      | 1           |
| 5        | 24       | 55       | 48   | 103             | 151     | 316      | 0           |
| 6        | 27       | 70       | 54   | 124             | 178     | 388      | 0           |
| 7        | 9        | 14       | 18   | 32              | 50      | 92       | 1           |
| 8        | 9        | 14       | 18   | 32              | 50      | 92       | 0           |
| 9        | 11       | 19       | 22   | 41              | 63      | 120      | 0           |
| 10       | 8        | 11       | 16   | 27              | 43      | 76       | 0           |
| 11       | 51       | 125      | 102  | 227             | 329     | 704      | 9           |
| 12       | 52       | 115      | 104  | 219             | 323     | 668      | 1           |
| 13       | 46       | 100      | 92   | 192             | 284     | 584      | 0           |
| 14       | 51       | 134      | 102  | 236             | 338     | 740      | 0           |

Table 6.2: Original problem size

coefficient matrix after adding slack variables. Column 7 gives the optimal objective value of every problem.

Table 6.3 shows the number of iterations of IPM using several different threshold value \(\delta\) in the starting point approach 2. It shows the results for two different optimality tolerance \(\epsilon\). From the total iterations in Table 6.3, we can see that the best threshold value of approach 2 is \(\delta = 0.4\).

Table 6.4 compares the iteration numbers of the three starting point approaches using two different optimality tolerances \(\epsilon\). From the computational results of approach 2 in Table 6.3, we know that the best threshold value is \(\delta = 0.4\), and the second best threshold value is \(\delta = 0.5\). Therefore, we test \(\delta = 0.4\) and \(\delta = 0.5\) in approach 3 to see the difference between approach 2 and approach 3. From the total iterations in Table 6.4, we can see that among the three starting point approaches, the best approach is approach 2 with \(\delta = 0.4\).

In Table 6.3 and Table 6.4, the iteration counts for all the feasible problems are relatively small. This is because as soon as the objective value is smaller than one, we stop the algorithm
and get a fractional $(g, f)$-factor; otherwise, we stop the algorithm only when the relative duality gap is smaller than the optimality tolerance $\epsilon$.

From the computational results in Table 6.3 and Table 6.4, we see that doubling the problem size for both feasible and infeasible problems has little effect on the iteration count. The effect is unpredictable. In some cases the iteration count decreases, and in some cases it increases. But doubling the problem size does not affect our conclusion that the $(g, f)$-factor problem-specific starting point (approach 2 with $\delta = 0.4$) is a good starting point and makes the IPM more efficient for solving the fractional $(g, f)$-factor problem.

All the test problems were solved extremely fast (in a few seconds for the largest problems), and so these methods might apply to problems much larger than these tested here. Nonetheless we imagine a direct network flow algorithm would outcompete these methods for all but very large problems. It is shown in [89], [85] and [88] that for many classes of large scale problems interior point methods are currently the most efficient computational solution approaches. Therefore, interior point methods provide, in addition to the direct network flow algorithm, new ways to solve large scale fractional $(g, f)$-factor problems.
Chapter 6. Solving the Fractional \((g, f)\)-factor Problem

<table>
<thead>
<tr>
<th>Problem</th>
<th>(\delta =)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.05</td>
</tr>
<tr>
<td>(\varepsilon = 1D - 8)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>71</td>
</tr>
</tbody>
</table>

| \(\varepsilon = 1D - 6\) |       |      |      |      |      |      |      |      |      |      |
| 1       | 6     | 8    | 6    | 6    | 6    | 6    | 6    | 6    | 7    | 7    |
| 2       | 9     | 8    | 7    | 8    | 7    | 8    | 8    | 8    | 8    | 8    |
| 3       | 3     | 5    | 3    | 3    | 2    | 2    | 2    | 2    | 2    | 2    |
| 4       | 7     | 8    | 8    | 7    | 6    | 7    | 8    | 7    | 7    | 7    |
| 5       | 4     | 5    | 4    | 3    | 3    | 3    | 3    | 3    | 3    | 3    |
| 6       | 4     | 5    | 4    | 3    | 3    | 2    | 2    | 2    | 2    | 2    |
| 7       | 6     | 6    | 5    | 6    | 6    | 6    | 6    | 6    | 6    | 6    |
| 8       | 1     | 3    | 2    | 2    | 2    | 2    | 2    | 2    | 2    | 2    |
| 9       | 1     | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 2    |
| 10      | 1     | 1    | 1    | 1    | 1    | 2    | 2    | 2    | 2    | 2    |
| 11      | 7     | 8    | 7    | 6    | 7    | 7    | 8    | 7    | 8    | 8    |
| 12      | 8     | 8    | 8    | 7    | 8    | 8    | 8    | 8    | 8    | 8    |
| 13      | 5     | 5    | 4    | 3    | 3    | 3    | 3    | 3    | 3    | 3    |
| 14      | 4     | 6    | 5    | 4    | 3    | 4    | 4    | 4    | 4    | 4    |
| Total   | 66    | 77   | 65   | 59   | 57   | 61   | 61   | 62   | 62   | 63   |

Table 6.3: Iteration counts for approach 2
Chapter 6. Solving the Fractional \((g,f)\)-factor Problem

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach 1</th>
<th>Approach 2</th>
<th>Approach 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \delta = 0.4 )</td>
<td>( \delta = 0.4 )</td>
<td>( \delta = 0.5 )</td>
</tr>
<tr>
<td>( \varepsilon = 1D - 8 )</td>
<td>( \varepsilon = 1D - 6 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>73</td>
<td>62</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>65</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4: Iteration counts for different starting point approaches
Bibliography


[22] S. Chen. Computational experience with the Karmarkar algorithm. Talk held at the ORSA/TIMS Joint National Meeting in Los Angeles, CA, USA, AT & T Bell Laboratories, Holmdel, NJ 07733, USA, April 1986.


Bibliography


