An Updated Stable Primal-Dual Interior-Point Algorithm for Linear Programming

by

Ryan Hughes

A Masters Research Project
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Masters
in
Computational Mathematics

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

Supervisor: Henry Wolkowicz
Professor, Dept. of Combinatorics & Optimization,
University of Waterloo

Second Reader: Thomas Coleman
Professor, Dept. of Combinatorics & Optimization,
University of Waterloo

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We revisit the stable linear programming algorithm developed by Gonzalez-Lima, Wei, & Wolkowicz [13] and provide an update that performs favorably on well-conditioned dense and sparse linear programs. The algorithm follows the primal-dual interior-point framework by applying Newton's method to the perturbed optimality conditions. It efficiently finds the Jacobian for a reduced system that has been shown to remain well conditioned as you approach optimality. Once in a region of quadratic convergence, the algorithm converts to taking a purely Newton step (with no backtracking) if the Jacobian is not ill conditioned. Numerical results are presented with comparisons to Mosek, Matlab's LinProg and SDPT3 on random dense problems. A heuristic for creating random dense linear programs is presented and used in the testing of the updated stable method. Also, results for solving sparse problems from the Netlib problem set show favorable results.

Acknowledgements

I would like to thank my wife, Peggy, for her help and support while I completed this. Also, I would like to especially thank Henry W. for his countless hours of help and patience as I completed this work.

Table of Contents

List of Tables

List of Algorithms

List of Figures

Glossary

backward error 20

centering parameter 7

central path 7

condition number 20

dual linear program 3

dual program 3

dual value 3

dual variables 3

Dual-Simplex 21

ellipsoid method 2

error magnification factor 21

feasible 3

forward error 20

interior-point method 2

Lagrangian 18

linear program 1

List of Symbols

Chapter 1

Introduction

The purpose of this paper is to revisit the stable linear program, **LP**, path-following method given in [13]. This algorithm is a primal-dual path-following method that is based on an inexact Newton method. A reduced system is solved at each iteration and then a backsolve step is used to obtain the complete search direction. No ill-conditioning is introduced as in the traditional normal equations approach. Stability of this method is shown in Theorem 3.2.2. It assumes that the constraint matrix to the **LP** has a special structure described in Section 3.1.2. We include the crossover technique used in [13] but have a dynamic estimate for the crossover based on the conditioning of the problem. If conditioning is poor, then no crossover occurs and the algorithm continues without switching to a pure Newton method.

Numerical testing is performed on well-conditioned dense random problems and a heuristic is presented on how to generate such problems. This updated approach performs favourably against modern solvers on dense well-conditioned random problems and some nondegenerate NETLIB problems.

1.1 History

Linear Programming first appeared when G.B. Dantzig developed the simplex method. The simplex method for solving a linear program was introduced by Dantzig in the 1940's when he was working as a Mathematical Advisor to the United States Air Force Comptroller [5]. Along with T.C. Koopmans, Dantzig laid down a major part of the foundations of linear programming during this time and developed the details of the simplex method [6, 7, 5, 8], an extremely effective algorithm for solving an **LP**.

For decades, Dantzig's simplex method was the primary algorithm that was used to solve linear programs despite it being an exponential-time algorithm in the worst case. In practice (on average), it proved to be quite effective despite the worst case poor behaviour. In 1979, L.G. Khachian [18] showed that the ellipsoid method of Nemirovski-Shor [23], [4] could be used to solve **LP** in polynomial-time. However, this algorithm turned out to be a computational disappointment. It was not practical and generally not competitive with the simplex method.

A few years later, in 1984, Karmarkar [17] developed a polynomial-time algorithm, quite unlike the simplex method or the ellipsoid method. It solves an **LP** by travelling through the interior of the feasible region, an interior-point method. The simplex method iterates on the boundary of the feasible region and the ellipsoid method is an exterior method, so this was a revolutionary idea that sparked excitement in the field of linear programming. From this, the field of primal-dual methods were developed that iterate through both the primal feasible region and the dual feasible region simultaneously to reach optimality. Primal-dual methods have matured to elegantly solve many very large problems [3]. More details about primal-dual methods are given in Chapter 2.

LP's are widely used in many areas including profit maximization, optimal resource allocation in manufacturing, bridge design to minimize weight, aircraft flight plan selection to minimize fuel and computer chip design for maximal heat dissipation. This is certainly not an exhaustive list of uses of **LP**'s.

1.2 Linear Programming

Linear Programming is concerned with finding the optimal solution to a linear function, called the objective function, given in terms of decision variables with some imposed linear constraints to the problem. Depending on the problem, the objective function is to be either minimized or maximized subject to the constraints. The constraints are a combination of linear equalities, inequalities, or sign restrictions imposed on the decision variables.

The **LP** we consider in standard form is given by:

$$\begin{array}{rrlll} & p^* & := & \min & c^T x \\ (\mathbf{LP}) & & & \text{s.t.} & Ax = b \quad (\in \mathbb{R}^m) \\ & & & & x \geq 0 \quad (\in \mathbb{R}^n_+) \end{array} \tag{1.2.1}$$

where $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. A, b and c are given with x called the primal variable. This **LP** is called the primal problem.

1.3 Duality

Every **LP** has a corresponding problem called its dual program. The concept that every **LP** has a companion dual program is very important in optimization theory. This section on Duality is outlined in [12].

The corresponding dual linear program, **DLP**, to (1.2.1) is given by:

$$\begin{array}{rrlll} & d^* & := & \max & b^T y \\ (\mathbf{DLP}) & & & \text{s.t.} & A^T y + z = c \quad (\in \mathbb{R}^n) \\ & & & & z \geq 0 \qquad\quad (\in \mathbb{R}^n_+) \end{array} \tag{1.3.1}$$

where $y \in \mathbb{R}^m$, $z \in \mathbb{R}^n$ and c, x, b, A are as above and y, z are called the dual variables. Note that if $z \geq 0 \implies A^T y \leq c$ is an equivalent statement.

For an **LP**, $x^* \in \mathbb{R}^n$ is considered optimal if $c^T x^* \leq c^T x$ for all feasible values of x, namely $\forall x \geq 0$ such that $Ax = b$. Similarly for **DLP**, $y^* \in \mathbb{R}^m$ and $z^* \in \mathbb{R}^n$ are optimal if $b^T y^* \geq b^T y$ for all feasible values of y and z, that is $\forall y$ and $\forall z \geq 0$ such that $A^T y + z = c$. We assume that we have primal-dual strict feasibility with the primal-dual feasible and strictly feasible set given by, respectively,

$$\begin{array}{rcl} \Omega: & = & \left\{(x, y, z) : Ax = b, A^T y + z = c, x \geq 0, z \geq 0\right\}; \\ \Omega_s: & = & \left\{(x, y, z) : Ax = b, A^T y + z = c, x > 0, z > 0\right\}. \end{array}$$

The primal value, p^*, is the minimum value that the **LP** obtains, which is equal to $c^T x^*$. Similarly for **DLP**, the dual value, d^*, is the maximum value of the dual, namely $b^T y^*$.

Theorem 1.3.1 (Weak Duality)**.** *If x^* is a solution to **LP** (1.2.1), and $\hat{y}$ is dual feasible, then*

$$c^T x^* \geq b^T \hat{y}$$

Proof. From dual feasibility, we know that $c \geq A^T \hat{y}$. Since x^* is a primal feasible solution, then we know that $x^* \geq 0$, $Ax^* = b$ and $(x^*)^T c \geq (x^*)^T A^T \hat{y}$. From this we get

$$c^T x^* = (x^*)^T c \geq (x^*)^T A^T \hat{y} = b^T \hat{y}$$

□

As a result of the Weak Duality Theorem, we have the following 3 corollaries:

Corollary 1.3.2. *If $\hat{x}$ is primal feasible and $\hat{y}$ dual feasible where $c^T\hat{x} = b^T\hat{y}$ then $\hat{x}$ and $\hat{y}$ are optimal solutions to **LP** and **DLP**, respectively.*

Proof. From Theorem 1.3.1 we know that for every optimal solution x^*, we have

$$c^T x^* \geq b^T \hat{y} = c^T \hat{x}$$

Thus, $\hat{x}$ is an optimal solution to **LP**. A Similar result is obtained for **DLP**. □

Corollary 1.3.3. *If **LP** is unbounded below, then **DLP** is infeasible.*

Proof. For any feasible solution, $\hat{y}$ of **DLP**, Theorem 1.3.1 indicates that $c^T x^*$ is bounded below by $b^T\hat{y}$. □

Using similar arguments, we have the following:

Corollary 1.3.4. *If **DLP** is unbounded above, then **LP** is infeasible.*

With these results, we can now obtain a stronger result called the *Strong Duality Theorem*:

Theorem 1.3.5 (Strong Duality for Linear Programming)**.**

1. *If either p^* or d^* obtains a finite value, then so does the other and $p^* = d^*$.*
2. *If either **LP** or **DLP** has an unbounded objective value then the other has no feasible solution.*

Strong Duality states that there is no duality gap between p^* and d^*. This is an important concept in primal-dual algorithms that is used to "measure" how close we are to the optimal solution as we sequentially iterate through the feasible region. Next we see that all active constraints in the optimal solution are tight.

Theorem 1.3.6 (Complementary Slackness)**.**

1. *If x^* and (y^*, z^*) are optimal solutions for **LP** and **DLP** respectively, then $(x^*)^T z^* = 0$;*

2. *If $\hat{x}$ feasible for **LP**, $(\hat{y}, \hat{z})$ feasible for **DLP** and $(x^*)^T z^* = 0$ then $\hat{x}$ and $(\hat{y}, \hat{z})$ are optimal for **LP** and **DLP** respectively.*

From Theorem 1.3.6 we see that once complementary slackness has been achieved for feasible x and (y, z), we have an optimal solution. Collecting all the criteria for optimal conditions, we have the following conditions to finding an optimal solution to **LP** and **DLP**.

Theorem 1.3.7 (KKT Optimality Conditions)**.** *Given **LP** and **DLP**, x is optimal for **LP**, and (y, z) are optimal for **DLP** if the following conditions are met:*

1. $Ax = b, \quad x \geq 0$ (primal feasibility)
2. $A^T y + z = c, \quad z \geq 0$ (dual feasibility)
3. $x^T z = 0$ (complementary slackness)

Theorem 1.3.7 outlines the conditions for an optimal solution. Our interior point method iterates to find an optimal solution by solving perturbed optimality conditions to converge to achieve the KKT conditions. More details on the Primal-Dual framework and the method used are given in Section 2.

1.4 Further Preliminaries

Throughout this paper we will use the additional notation: I is the identity matrix of appropriate dimension; for $x \in \mathbb{R}^n$, we let $X :=$Diag(x) be the diagonal square matrix with the elements of x on the diagonal; a vector of all ones with appropriate dimension is denoted by e. Norms used in the stopping criteria is the L_2 norm, denoted $\|\cdot\|$, and the Frobenius norm, denoted $\|\cdot\|_F$. Given $F : \mathbb{R}^n \to \mathbb{R}^n$, we let $F'(x)$ be the Jacobian of F at x. To help differentiate between block matrices and vectors throughout this paper we use square brackets, $[\cdot]$, to denote a matrix and round brackets, $(\cdot)$, to denote a vector.

1.5 Outline

The framework for primal-dual algorithms is given in Chapter 2. Our updated stable method is outlined in Chapter 3 with the proof of method stability shown in Theorem 3.2.2. Stopping Criteria, Backtracking, Starting Point and Crossover details are given in Sections 4.1, 4.2, 4.3 and 4.4, respectively. The results from the numerical experiments for both dense and sparse problems are given in Chapter 5. The areas of potential improvement for future work is given in Chapter 6. Concluding remarks are given in Chapter 7.

Chapter 2

Classical Primal-Dual Framework

We follow a primal-dual interior point framework, e.g., [27]. Primal-dual interior point methods solve both the primal and dual problems simultaneously, but keep $x, z > 0$, interiority, satisfied at every iteration. We can rewrite and perturb the optimality conditions for problems (1.2.1) and (1.3.1) as a mapping $F_\mu : \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ in the following way

$$F_\mu(x, y, z) := \begin{pmatrix} A^T y + z - c \\ Ax - b \\ ZXe - \mu e \end{pmatrix} = 0, \qquad X, Z > 0, \tag{2.0.1}$$

where the barrier parameter $\mu > 0$, $X := \text{Diag}(x)$, $Z := \text{Diag}(z)$, and both $x, z \in \mathbb{R}^n_{++}$. The third term is to ensure complementary slackness, i.e., $x_i z_i = 0, \forall i$ in the limit as $\mu \downarrow 0$. The target, μ, is updated at the current iterate using the complementarity equation

$$\mu = \frac{1}{n}\mu e^T e = \frac{1}{n} e^T ZXe = \frac{1}{n} z^T x. \tag{2.0.2}$$

Algorithm 2.0.1 outlines the framework for primal-dual interior point methods. We allow for $\sigma \in (0, 1]$ to be changed adaptively. And we backtrack *safely* to maintain positivity. The details for these steps are given in Chapter 4 for the stable method.

At each step, Newton's method generates the next iteration towards optimality by finding the Newton search direction, Δs, as the solution of the following system

$$F'(x, y, z)\Delta s = -F(x, y, z), \quad \text{where } \Delta s := \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix}, \tag{2.0.3}$$

or equivalently with the three residuals, r_d, r_p, r_c

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = - \begin{pmatrix} A^T y + z - c \\ Ax - b \\ ZXe - \mu e \end{pmatrix} \quad =: - \begin{pmatrix} r_d \\ r_p \\ r_c \end{pmatrix} =: r_s. \tag{2.0.4}$$

If the current point is strictly feasible, $s \in \Omega_s$, then (2.0.4) becomes

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ r_c \end{pmatrix}. \tag{2.0.5}$$

However, taking a full Newton step does not guarantee that the strict feasibility condition $x, z > 0$ is maintained. Therefore we use a damped Newton method and choose primal and dual steplengths α_p, $\alpha_d \in (0, 1]$ to obtain a subsequent point,

$$x + \alpha_p \Delta x, \quad y + \alpha_d \Delta y, \quad z + \alpha_d \Delta z. \tag{2.0.6}$$

It is desirable to have $\alpha_p \approx 1$ and $\alpha_d \approx 1$ so we take a large step towards optimality, but this is not always possible because we exit the feasible region. Often, only small steps where $\alpha_p \approx 0$ and $\alpha_d \approx 0$ are allowed to stay feasible but this results in only taking small steps toward optimality. The steplength used in this updated stable method is adaptive and for large α_d and α_p we increase the steplength to > 1 to approach optimality more aggressively.

To address the issue of potentially losing feasbility at each iteration, we stay close to the central path of the feasible region. If Ω_s is non-empty then the central path is well-defined, [27, Chapter 2].

We attempt to maintain that our current iterates stay within a region of the central path while moving towards the optimal solution. We introduce a centering parameter, $\sigma \in (0, 1]$ such that (2.0.5) becomes

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = - \begin{pmatrix} 0 \\ 0 \\ ZXe - \sigma \mu e \end{pmatrix}. \tag{2.0.7}$$

Here $\mu > 0$ is the so-called log-barrier parameter. A small value of σ means that we aggressively move towards the optimum; while a value close to 1 means that we are conservative

and move towards the central path. By solving for Δs in (2.0.7), there is a way to find a Newton direction that attempts to move toward optimality while staying within a region of the central path.

Algorithm 2.0.1 Primal-Dual Framework for Linear Progams [27, p. 8]

1: Given initial point $(x^0, y^0, z^0), x^0 > 0, z^0 > 0,$
2: **for** k=0,1,2,... **do**
3: solve the following for $\left(\Delta x^k, \Delta y^k, \Delta z^k\right)^T$

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z^k & 0 & X^k \end{bmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta z^k \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -X^k Z^k e + \sigma_k \mu_k e \end{pmatrix}$$

where $\sigma_k \in (0, 1]$ and $\mu_k = \frac{(x^k)^T z^k}{n}$

4: set

$$(x^{k+1}, y^{k+1}, z^{k+1}) \leftarrow (x^k, y^k, z^k) + \alpha_k(\Delta x^k, \Delta y^k, \Delta z^k)$$

choosing α_k so that $(x^{k+1}, z^{k+1}) > 0$
5: **end for**

Chapter 3

Stable Method

The stable method is a modification of the normal equations approach. Here we discuss the normal equations approach for solving the Newton equations for the search direction, and we highlight where numerical difficulties can arise. Details of the stable approach is presented and the stability of the method is shown.

3.1 Search Directions

3.1.1 Normal Equations

The normal equations approach is a standard method used to solve a linear programs, e.g., [13]. At each iteration of the primal-dual interior point algorithm, the Newton equation (2.0.3) is solved for the Newton direction Δs. Solving for the Newton direction without exploiting the structure is too expensive; so block elimination is employed to solve it more efficiently. Typically, Δz is eliminated first. This is equivalent theoretically to left-multiplying by the elementary block-pivoting matrix P_z, defined as

$$P_z := \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -X & 0 & I \end{bmatrix}.$$

So our Newton system becomes

$$K := P_z F'_\mu = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -X & 0 & I \end{bmatrix} \begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix} = \begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & -XA^T & 0 \end{bmatrix}, \tag{3.1.1}$$

with the corresponding right-hand side

$$-P_z F_\mu = -\begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -X & 0 & I \end{bmatrix} \begin{pmatrix} A^T y + z - c \\ Ax - b \\ ZXe - \mu e \end{pmatrix} = -\begin{pmatrix} r_d \\ r_p \\ -Xr_d + ZXe - \mu e \end{pmatrix}. \tag{3.1.2}$$

The next step in the normal equations approach is to solve for Δx. This is equivalent to further left-multiplying the system by the transformation P_x, defined as

$$P_x := \begin{bmatrix} I & 0 & 0 \\ 0 & I & -AZ^{-1} \\ 0 & 0 & Z^{-1} \end{bmatrix}$$

to get

$$P_x P_z F'_\mu = \begin{bmatrix} I & 0 & 0 \\ 0 & I & -AZ^{-1} \\ 0 & 0 & Z^{-1} \end{bmatrix} \begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & -XA^T & 0 \end{bmatrix} = \begin{bmatrix} 0 & A^T & I_n \\ 0 & AZ^{-1}XA^T & 0 \\ I_n & -Z^{-1}XA^T & 0 \end{bmatrix}, \tag{3.1.3}$$

with the right-hand side becoming

$$\begin{aligned} -P_x P_z F_\mu &= -\begin{bmatrix} I & 0 & 0 \\ 0 & I & -AZ^{-1} \\ 0 & 0 & Z^{-1} \end{bmatrix} \begin{pmatrix} r_d \\ r_p \\ -Xr_d + ZXe - \mu e \end{pmatrix} \\ &= \begin{pmatrix} -r_d \\ -r_p + A(-Z^{-1}Xr_d + x - \mu Z^{-1}e) \\ Z^{-1}Xr_d - x + \mu Z^{-1}e \end{pmatrix}. \end{aligned} \tag{3.1.4}$$

We can now solve for Δy and then backsolve. After appropriate rearrangement we get the equivalent system

$$\begin{bmatrix} I_n & 0 & A^T \\ 0 & I_n & -Z^{-1}XA^T \\ 0 & 0 & AZ^{-1}XA^T \end{bmatrix} \begin{pmatrix} \Delta z \\ \Delta x \\ \Delta y \end{pmatrix} = \begin{pmatrix} -r_d \\ Z^{-1}Xr_d - x + \mu Z^{-1}e \\ -r_p + A(-Z^{-1}Xr_d + x - \mu Z^{-1}e) \end{pmatrix}. \tag{3.1.5}$$

The last block (normal equations) can be solved directly for Δy. Then we can backsolve to obtain Δx and finally Δz. Though this system can be solved efficiently, this system becomes ill-conditioned as we approach the optimum, even though the normal equations for Δy can remain well-conditioned. An example showing that this system becomes ill-conditioned near the optimum is given in [13, Example 2.3]. P_x is an ill-conditioned transformation near the optimum and this results in the entire system becoming ill-conditioned.

3.1.2 Stable Approach

The stable approach does not have a transformation that makes the problem ill-conditioned. The stable reduction step in this approach assumes that A has a special block structure (after a permutation if needed), namely

$$A = \begin{bmatrix} B & E \end{bmatrix}, \quad B \in \mathbb{R}^{m \times m} \text{ invertible.} \tag{3.1.6}$$

We assume that B is well-conditioned, and that the triangular factorization $B = LU$ is inexpensive. In the large, sparse case, we also assume that both L and U are sparse. This stable approach is discussed in more detail and computational results are given in [13]. Generally, the number of variables n is significantly greater than the number of constraints m.

To take advantage of the special structure of A, we partition x and z appropriately to

$$x = \begin{pmatrix} x_m \\ x_v \end{pmatrix} \text{ and } z = \begin{pmatrix} z_m \\ z_v \end{pmatrix}$$

with lengths m and $n-m$, respectively, i.e., $x_m, z_m \in \mathbb{R}^m$ and $x_v, z_v \in \mathbb{R}^{n-m}$. Subsequently, the diagonal matrices X_m, X_v, Z_m, and Z_v are similarly defined square matrices. We expand the block matrix K in (3.1.1) and define the elementary matrix for block Gaussian elimination P_{stable} to be

$$P_{stable} := \begin{bmatrix} I_n & 0 & 0 & 0 \\ 0 & B^{-1} & 0 & 0 \\ 0 & -Z_m B^{-1} & I_m & 0 \\ 0 & 0 & 0 & I_v \end{bmatrix}.$$

We define the resulting system after the block elimination, F_{stable}, to be

$$\begin{aligned} F_{stable} := P_{stable} P_z F'_\mu &= \begin{bmatrix} I_n & 0 & 0 & 0 \\ 0 & B^{-1} & 0 & 0 \\ 0 & -Z_m B^{-1} & I_m & 0 \\ 0 & 0 & 0 & I_v \end{bmatrix} \begin{bmatrix} 0 & 0 & A^T & I_n \\ B & E & 0 & 0 \\ Z_m & 0 & -X_m B^T & 0 \\ 0 & Z_v & -X_v E^T & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & A^T & I_n \\ I_m & B^{-1}E & 0 & 0 \\ 0 & -Z_m B^{-1} E & -X_m B^T & 0 \\ 0 & Z_v & -X_v E^T & 0 \end{bmatrix}, \end{aligned} \tag{3.1.7}$$

where the right-hand side from (3.1.2) becomes

$$\begin{aligned} -P_{stable} P_z F_\mu &= -P_{stable}(P_z F_\mu) \\ &= -P_{stable} \begin{pmatrix} r_d \\ r_p \\ -X_m (r_d)_m + Z_m X_m e - \mu e \\ -X_v (r_d)_v + Z_v X_v e - \mu e \end{pmatrix} \\ &= \begin{pmatrix} -r_d \\ -B^{-1} r_p \\ Z_m B^{-1} r_p + X_m (r_d)_m - Z_m X_m e + \mu e \\ X_v (r_d)_v - Z_v X_v e + \mu e \end{pmatrix}. \end{aligned} \tag{3.1.8}$$

Putting equations (3.1.7) and (3.1.8) together, we solve the following system

$$\begin{bmatrix} 0 & 0 & A^T & I_n \\ I_m & B^{-1}E & 0 & 0 \\ 0 & -Z_m B^{-1} E & -X_m B^T & 0 \\ 0 & Z_v & -X_v E^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x_m \\ \Delta x_v \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} -r_d \\ -B^{-1} r_p \\ Z_m B^{-1} r_p + X_m (r_d)_m - Z_m X_m e + \mu e \\ X_v (r_d)_v - Z_v X_v e + \mu e \end{pmatrix}. \tag{3.1.9}$$

The stable reduction step in solving this system is obtaining Δx_v and Δy from the bottom two rows of (3.1.9). This step which is the most computationally expensive operation, is solving the following reduced system

$$J_s \begin{pmatrix} \Delta x_v \\ \Delta y \end{pmatrix} = F_s, \tag{3.1.10}$$

where the Jacobian and right hand side are, respectively,

$$J_s := \begin{bmatrix} Z_m B^{-1} E & -X_m B^T \\ Z_v & -X_v E^T \end{bmatrix}, \text{ and } F_s := \begin{pmatrix} Z_m B^{-1} r_p + X_m (r_d)_m - Z_m X_m e + \mu e \\ X_v (r_d)_v - Z_v X_v e + \mu e \end{pmatrix}.$$

Next, the second row of (3.1.9) is used to backsolve for Δx_m. Finally, another backsolve of the top row is done to solve for Δz. In performing these backsolves, the matrix B^{-1} is never evaluated but instead the required product is evaluated using system solves and the LU factorization. For instance, in the second row of (3.1.9), the matrix product $B^{-1}(Ex_v)$ is evaluated using a system solve and the LU factorization when solving for Δx_m.

3.2 Stability and Convergence

One important aspect of this stable method is that it remains numerically stable as you approach optimality. Here we show that the Jacobian maintains numerical stability since it is nonsingular.

The stable reduction step is based on finding a null space representation of A. The step is also known as a null space method [2, Chapter 6]. The method has two key assumptions. It assumes that a particular solution $\hat{x}$ of $Ax = b$ is available, and that a matrix $N \in \mathbb{R}^{n \times (n-m)}$ is available such that $AN = 0$, that is that the columns of N spans the null space of A.

Given the initial solution $\hat{x}$, we use the substitution $x = \hat{x} + Nv$, for some $v \in \mathbb{R}^{n-m}$. Then

$$\begin{aligned} b = Ax \quad &\iff \quad b = A(\hat{x} + Nv) \\ &\iff \quad b = A\hat{x} + ANv = A\hat{x}, \quad \text{since } AN = 0 \\ &\iff \quad b = A\hat{x}. \end{aligned}$$

As stated earlier, this paper assumes that A has the form (3.1.6) and the system $Bv = d$ has a unique solution, is well-conditioned, and is inexpensive to solve. Furthermore, we assume that N has the form

$$N = \begin{bmatrix} -B^{-1}E \\ I_{n-m} \end{bmatrix}. \tag{3.2.1}$$

We note that these properties may only be available for B after a permutation of the rows and columns of A.

The stable reduction for the Newton search direction obtained from solving (3.1.9) is a linearization of the perturbed Newton equation given in (3.2.4), below. We first find the principal submatrix B of A. We now continue with the stable linearization [13].

We can now eliminate the first two linear blocks of equations in (2.0.1) to obtain a single block of equations for optimality.

Theorem 3.2.1. *Suppose $x = \hat{x} + Nv \geq 0$, $z = c - A^T y \geq 0$ and $A\hat{x} = b$. Also, suppose that $\text{Null}(A) = \text{Range}(N)$, then x, y, z are optimal for (1.2.1) and (1.3.1) if and only if*

$$\text{Diag}(c - A^T y)\,\text{Diag}(\hat{x} + Nv)e = 0 \tag{3.2.2}$$

We can see that (3.2.2) is equivalent to the complementary slackness condition, $ZXe = 0$. The perturbed optimality conditions for our primal-dual method is given by

$$\mathcal{F}_\mu(v, y) := \text{Diag}(c - A^T y)\,\text{Diag}(\hat{x} + Nv)e - \mu e = 0 \tag{3.2.3}$$

This is a nonlinear system of two variables, v and y. The linearization of this system for $\Delta\hat{s} := \begin{pmatrix} \Delta v \\ \Delta y \end{pmatrix}$ is given by

$$\mathcal{F}'_\mu(v, y)\Delta\hat{s} = -\mathcal{F}_\mu(v, y) \tag{3.2.4}$$

where $\mathcal{F}'_\mu(v, y)$ is the Jacobian defined as

$$\begin{aligned} \mathcal{F}'_\mu(v, y) &:= \begin{bmatrix} \text{Diag}(c - A^T y)N & -\text{Diag}(\hat{x} + Nv)A^T \end{bmatrix} \\ &= \begin{bmatrix} ZN & -XA^T \end{bmatrix}. \end{aligned} \tag{3.2.5}$$

Therefore, from (3.2.4) and (3.2.5) our linearized system is

$$\begin{bmatrix} ZN & -XA^T \end{bmatrix} \begin{pmatrix} \Delta v \\ \Delta y \end{pmatrix} = -\mathcal{F}_\mu(v,y)$$
$$ZN\Delta v - XA^T\Delta y = -\mathcal{F}_\mu(v,y). \tag{3.2.6}$$

The first $n-m$ variables is often the larger, more difficult part of (3.2.6) to solve but if E is sparse then it is inexpensive. The first $n-m$ variables corresponds to solving for Δv. The second, usually smaller part is the m variables corresponding to solving for Δy. The next theorem shows that the Jacobian matrix $\mathcal{F}'_\mu$ at optimality does not get ill-conditioned as μ approaches 0.

Theorem 3.2.2. *Suppose we have an LP given by (1.2.1) and its dual given by (1.3.1). Also suppose A is onto, full-rank and $\text{Null}(A) = \text{Range}(N)$. Further suppose that N has full column rank and $(\bar{x}, \bar{y}, \bar{z})$ is the unique optimal solution. Then the Jacobian matrix $\mathcal{F}'_\mu$ from (3.2.5) is nonsingular.*

Proof. Suppose $\mathcal{F}'_\mu \Delta\hat{s} = 0$, so if we show $\Delta\hat{s} = \begin{pmatrix} \Delta v \\ \Delta y \end{pmatrix} = 0$ then we are done. Let $\mathcal{B} = \{i : x_j = \hat{x}_j + (Nv)_j > 0\}$ and $\mathcal{N} = \left\{i : z_i + c_i - (A^T y)_i > 0\right\}$. Since A is onto and full-rank, then $\mathcal{B} \cap \mathcal{N} = \emptyset$. Also $A_\mathcal{B}$, the matrix formed by the columns of A from indices in $\mathcal{B}$, is nonsingular.

Since $\mathcal{F}'_\mu \Delta\hat{s} = 0$ and (3.2.6) we have

$$0 = \mathcal{F}'_\mu \Delta\hat{s} = \left(c - A^T y\right)(Nv) - (\hat{x} + Nv)\left(A^T \Delta y\right). \tag{3.2.7}$$

From complementary slackness, definitions and (3.2.7), the following conditions must be satisfied

$$\begin{aligned} c_j - (A^T y)_j = 0, \quad & \hat{x}_j + (Nv)_j > 0, \quad & (A^T\Delta y)_j = 0, \quad \forall j \in \mathcal{B} \\ c_i - (A^T y)_i > 0, \quad & \hat{x}_i + (Nv)_i = 0, \quad & (N\Delta v)_i = 0, \quad \forall i \in \mathcal{N}. \end{aligned} \tag{3.2.8}$$

The first line of (3.2.8) shows that $(A^T\Delta y)_j = 0$ for all $j \in \mathcal{B}$, but $A_\mathcal{B}$ is nonsingular, therefore $\Delta y = 0$.

We now show that $\Delta v = 0$. Since the range of N is the null space of A then $AN = 0$, so

$$0 = \begin{bmatrix} A_{\mathcal{B}} & A_{\mathcal{N}} \end{bmatrix} \begin{bmatrix} (N\Delta v)_{\mathcal{B}} \\ (N\Delta v)_{\mathcal{N}} \end{bmatrix} = A_{\mathcal{B}}(N\Delta v)_{\mathcal{B}} + \underbrace{A_{\mathcal{N}}(N\Delta v)_{\mathcal{N}}}_{=0} = A_{\mathcal{B}}(N\Delta v)_{\mathcal{B}} \tag{3.2.9}$$

From (3.2.8), we see that $A_N(N\Delta v)_N = 0$ since $(N\Delta v)_i = 0,\ \forall i \in \mathcal{N}$. This gives us

$$\begin{array}{rcccl} & A_{\mathcal{B}}(N\Delta v)_{\mathcal{B}} & = & 0 & \\ \implies & (N\Delta v)_{\mathcal{B}} & = & 0, & \text{since } A_{\mathcal{B}} \text{ is nonsingular} \\ \implies & \Delta v & = & 0, & \text{since } N \text{ is full-rank.} \end{array}$$

□

We use (3.2.3) and the linearization (3.2.6) to develop the stable primal-dual algorithm.

Chapter 4

Implementation Details

Components to the implementation of the updated stable algorithm is presented here. Steplength, stopping criteria, starting point and crossover criteria are given and the complete updated stable algorithm is given in Algorithm 4.4.2.

4.1 Stopping Criteria

Stopping criteria used in our algorithm is checking for a small relative gap [26] if we do not crossover to take full Newton steps to optimality. If there is no crossover into a region of quadratic convergence, then the precision must be less than

$$\frac{x^T z}{1 + \max\left(|c^T x|, |b^T y|\right)}.$$

However, if we do cross over into a region of quadratic convergence then our stopping criteria changes so our precision must be less than

$$\frac{\|F_s\|}{\|A\|_F + \|b\|}.$$

Complete details for checking the crossover are given in Section 4.4.

4.2 Backtracking & Steplength

To prevent ill-conditioning, backtracking is done at each step to stay away from the boundary of the feasible region before the crossover. Steplength is adaptive with steplengths greater than 1 allowed for large α_d and α_d.

It is common to use a constant value for the centering parameter, σ. Here, an adaptive approach is used before the crossover to allow for greater flexibility and potentially better algorithm performance. If the crossover is taken then centering is not needed and $\sigma = 0$. The Mehrotra's Predictor-Corrector (MPC) method [19] is not used in this algorithm.

4.3 Starting Point

Having a poor choice of starting point can lead to non-convergence. Only satisfying positivity of starting points x^0 and z^0 can still lead to convergence issues so a better choice has a significant affect on the robustness of the algorithm. Here we use a heuristic for finding a good starting point given in [21].

The first step is to solve two minimization problems finding the minimum norms for $\hat{x}$ and $\hat{z}$ given the primal and dual constraints. That is, we are solving the following two problems

$$\begin{aligned} \min_{\hat{x}} \quad & \frac{1}{2}\hat{x}^T\hat{x} \\ \text{s.t.} \quad & A\hat{x} = b \end{aligned} \tag{4.3.1}$$

and

$$\begin{aligned} \min_{(\hat{y},\hat{z})} \quad & \frac{1}{2}\hat{z}^T\hat{z} \\ \text{s.t.} \quad & A^T\hat{y} + \hat{z} = c. \end{aligned} \tag{4.3.2}$$

Finding the Lagrangian of (4.3.1) gives us

$$L_1(\hat{x}, \hat{y}) = \min_{\hat{x}} \frac{1}{2}\hat{x}^T\hat{x} + \hat{y}^T(A\hat{x} - b). \tag{4.3.3}$$

Now (4.3.3) is an unconstrained minimization problem so we can set the partial derivatives to zero to find optimal value $\hat{x}$

$$\nabla_{\hat{x}} L_1(\hat{x}, \hat{y}) = \hat{x} + \hat{y}^T A = \hat{x} + A^T \hat{y}$$
$$\nabla_{\hat{x}} L_1(\hat{x}, \hat{y}) = 0 \implies \hat{x} + A^T \hat{y} = 0 \implies \hat{x} = -A^T \hat{y} \tag{4.3.4}$$

Now we find the partial derivative with respect to $\hat{y}$ and set it to zero

$$\begin{aligned} & \nabla_{\hat{y}} L_1(\hat{x}, \hat{y}) && = 0 \\ \implies & A\hat{x} - b && = 0 \\ \implies & A\hat{x} && = b \\ \implies & A\left(-A^T \hat{y}\right) && = b \\ \implies & \hat{y} && = -\left(AA^T\right)^{-1} b \end{aligned}$$

Substituting $\hat{y}$ into (4.3.3) give us $\hat{x} = A^T \left(AA^T\right)^{-1} b$. Performing the similar steps for (4.3.2), we acquire $\hat{z} = c - A^T \hat{y}$. To be a feasible point we need $\hat{x}, \hat{z} \geq 0$, but this first step cannot guarantee non-negativity.

The second step adds values to both $\hat{x}$ and $\hat{z}$ so the resulting vector is positive. The minimum values of both $\hat{x}$ and $\hat{z}$ are used that the resulting vector is above zero. The resulting values are given by

$$\tilde{x} := \hat{x} + \max\left(0, -\frac{3}{2}\min \hat{x}_i\right) e, \quad \text{and} \quad \tilde{z} := \hat{z} + \max\left(0, -\frac{3}{2}\min \hat{z}_i\right) e.$$

To further ensure that the starting point are not too close to zero and not too dissimilar, we add another term to achieve this. This term is the average element size of $\tilde{x}$, weighted by the elements of $\tilde{z}$ given by

$$x^0 = \tilde{x} + \frac{1}{2}\frac{\tilde{x}^T \tilde{z}}{e^T \tilde{z}_i}$$

Similarly for $\tilde{z}$ we get

$$z^0 = \tilde{z} + \frac{1}{2}\frac{\tilde{x}^T \tilde{z}}{e^T \tilde{x}_i}$$

The complete starting point algorithm used is given in Algorithm 4.3.1.

Algorithm 4.3.1 Generating starting point [21, Section 14.2]

Inputs:
$A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$
$y^0 \leftarrow (AA^T)^{-1}Ac$ ▷ Step 1
$\hat{x} \leftarrow A^T(AA^T)^{-1}b$
$\hat{z} \leftarrow c - A^T y^0$
▷ Step 2
$\tilde{x} \leftarrow \hat{x} + \max\left(0, -\frac{3}{2}\min \hat{x}_i\right)e$
$\tilde{z} \leftarrow \hat{z} + \max\left(0, -\frac{3}{2}\min \hat{z}_i\right)e$
▷ Step 3
$x^0 \leftarrow \tilde{x} + \frac{1}{2}\frac{\tilde{x}^T\tilde{z}}{e^T\tilde{z}_i}$
$z^0 \leftarrow \tilde{z} + \frac{1}{2}\frac{\tilde{x}^T\tilde{z}}{e^T\tilde{x}_i}$
return (x^0, y^0, z^0)

4.4 Crossover

When the current iterate is close enough to the optimal solution, then a Newton type approach implies that a pure Newton method attains quadratic convergence, i.e., we set *Newton's method free* and take step lengths of one with no backtracking to stay positive. The crossover boundary where quadratic convergence is attained not only depends on the relative gap but also on the condition number of the reduced Jacobian matrix, J_s.

The condition number of any square matrix $\mathcal{A}$ is given by

$$\text{cond}\,(\mathcal{A}) = \|\mathcal{A}\| \cdot \|\mathcal{A}^{-1}\|$$

where $\|\cdot\|$ is a vector norm. Given an approximate solution x_a to the linear system $Ax = b$, the backward error and relative backward error are defined to be

$$\|b - Ax_a\|, \text{ and } \frac{\|b - Ax_a\|}{\|b\|}$$

respectively. Additionally, forward error and relative forward error are defined to be

$$\|x - x_a\|, \text{ and } \frac{\|x - x_a\|}{\|x\|}$$

respectively. The error magnification factor is the ratio between the relative forward error and the relative backward error, namely

$$\frac{\dfrac{\|x - x_a\|}{\|x\|}}{\dfrac{\|b - Ax_a\|}{\|b\|}}.$$

The condition number returns the maximum possible error magnification for solving a linear system $Ax = b$ over all possible b's [25].

In order to be efficient computationally, the condition number is estimated by a 1-norm condition number estimate using the `condest` function is MATLAB. The condition number estimate is an efficient way of finding a lower bound for the condition number of a matrix [15, 16]. To remain competitive the condition number estimate cannot be calculated at every iteration so this method only performs the calculation once the relative gap reaches a desired tolerance. If the condition number is small then there is a crossover to a pure Newton method. However, if the condition number remains large, there is no crossover and continues iterating as before. The crossover check is only performed once and not checked again. If the crossover occurs then the algorithm stopping criteria changes for a precision less than

$$\frac{\|F_s\|}{\|A\|_F + \|b\|}$$

The computation time of the estimate is equivalent to approximately 1-2 iterations of computation so it is therefore important that it is not calculated at each iteration.

During testing of the algorithm, a large condition number for J_s caused the crossover to fail and the system ultimately diverged. Once the condition number was taken into consideration, the results stabilized. Indications show that the condition number of J_s depend on the condition number of the original basis to the **LP**. The complete modified stable algorithm is given in Algorithm 4.4.2.

Mosek also uses a crossover technique to improve performance time. However, Mosek's crossover is not marking the region of quadratic convergence, but rather performs a Dual-Simplex method once within a region around the optimal value. The Dual-Simplex method

keeps dual feasibility and complementary slackness while seeking for primal feasibility at the optimum.

Algorithm 4.4.2 Modified stable primal-dual interior-point algorithm

Initialize:
  iter $\leftarrow 0$
  crossover $\leftarrow$ **false**
  testcrossover $\leftarrow$ **true**
  crosstol $\leftarrow$ tol
  $x \leftarrow x^0, \quad y \leftarrow y^0, \quad z \leftarrow z^0$
  stopcrit $\leftarrow \dfrac{x^T z}{1 + \max(|c^T x|, |b^T y|)}$
while stopcrit$<10^{-\text{digits}}$ & iter<maxIters **do**
  Find Newton direction, $\Delta s = \begin{pmatrix} \Delta x & \Delta y & \Delta z \end{pmatrix}^T$ by solving (3.1.9)
  if (stopcrit<crosstol) & (testcrossover) & (**NOT** crossover) **then**
    testcrossover $\leftarrow$ **false** ▷ Only gets checked once
    if cond $(J_s) < \frac{1000}{\text{crosstol}}$ **then** ▷ Testing for Crossover
      $\sigma \leftarrow 0, \quad \mu \leftarrow 0, \quad \alpha_p \leftarrow 1, \quad \alpha_d \leftarrow 1, \quad$ crossover $\leftarrow$ **true**
    end if
  else
    if NOT crossover **then**
      $\alpha_p \leftarrow 0.997 \cdot \min\left(1, \min\limits_{i:\Delta x_i<0} -\dfrac{x_i}{\Delta x_i}\right)$
      $\alpha_d \leftarrow 0.997 \cdot \min\left(1, \min\limits_{i:\Delta z_i<0} -\dfrac{z_i}{\Delta z_i}\right)$
      $\mu \leftarrow x^T z / n$
    end if
  end if
  $x \leftarrow x + \alpha_p \Delta x, \quad y \leftarrow y + \alpha_d \Delta y, \quad z \leftarrow z + \alpha_d \Delta z$ ▷ Taking step
  if crossover **then** ▷ Performing Updates
    $\sigma \leftarrow 0, \quad \mu \leftarrow 0, \quad \alpha_p \leftarrow 1, \quad \alpha_d \leftarrow 1$
    stopcrit $\leftarrow \dfrac{\|F_s\|}{\|A\|_F + \|b\|}$ ▷ Stopping criteria changes
  else
    Adjust σ, α_p and α_d adaptively
    stopcrit $\leftarrow \dfrac{x^T z}{1 + \max(|c^T x|, |b^T y|)}$
  end if
  $r_p \leftarrow Ax - b, \quad r_d \leftarrow A^T y + z - c, \quad r_c \leftarrow x^T z - \sigma\mu$
  iter $\leftarrow$ iter+1
end while
return $\begin{pmatrix} x & y & z \end{pmatrix}^T$

Chapter 5

Numerical Experiments

Numerical testing was performed on well-conditioned random problems and on some sparse problems from the NETLIB library. Random problems were generated such that they were well-conditioned and already in the form satisfying (3.1.6). The method for generating well-conditioned random problems are given in Section 5.1.

All computations were done in MATLAB 2019a on a Macbook Pro containing a 2.2 GHz Intel i7 CPU with 16GB RAM. The NETLIB problems used were taken from the University of Florida Sparse Matrix Laboratory [9] in MATLAB `.mat` file format.

Solver comparisons were done with Mosek [1] and MATLAB's `linprog` function for interior point methods. Initially comparisons were also done with SDPT3 [26] but it consistently took 2-3 times longer than the slowest of the other solvers so its results were excluded in the tables. The default settings were used for the 3 solvers except that the interior point algorithm was chosen for the LinProg solver. Performance profiles [10, 14] were used to make comparisons between solvers.

5.1 Random (Dense) Problems

To perform numerical experiments on random problems, we construct our problems to ensure that the sub-matrix B is well-conditioned and that the optimal solution is unique. Our algorithm expects the constraint matrix to have the special structure in (3.1.6). Therefore, we construct B and E appropriately, and then concatenate to form A.

To construct well-conditioned B, we use Algorithm 5.1.1, below. That is we generate two $m \times m$ orthogonal matrices U_B, V_B and diagonal matrix Σ_B, where $\text{cond}\,(\Sigma_B) \approx \mathcal{C}_m$.

For this work we chose $\mathcal{C}_m = \log(m)$ because the expected L_2 norm condition number of real $m \times m$ matrices generated from a standard normal distribution is approximately $\log(m) + 1.537$ as $m \to \infty$ [11]. The choice of $\mathcal{C}_m = \log(m)$ is a minor simplification.

To ensure a matrix has a custom condition number, the singular values are constructed so that the largest singular value is $\mathcal{C}_m$ and the smallest is 1. The desired singular values are

$$\begin{pmatrix} \mathcal{C}_m \\ \vdots \\ 1 \end{pmatrix}.$$

Given a set of descending singular values $(\sigma_1, \ldots, \sigma_m)^T$, we want to find constants ω and ξ such that

$$\omega\sigma_1 + \xi = \mathcal{C}_m, \text{ and} \tag{5.1.1}$$
$$\omega\sigma_m + \xi = 1. \tag{5.1.2}$$

Substituting (5.1.2) into (5.1.1) and solving for ω, we obtain

$$\omega = \frac{\mathcal{C}_m - 1}{\sigma_1 - \sigma_m}, \text{ and } \xi = 1 - \omega\sigma_m.$$

This transformation is applied to the diagonal entries of Σ_B to obtain the desired condition number.

We next define $B := U_B \Sigma_B V_B^T$. To construct E we proceed in a similar fashion. We generate orthogonal matrices $U_E \in \mathbb{R}^{m\times m}$, $V_E \in \mathbb{R}^{(n-m)\times(n-m)}$ and diagonal $\Sigma_E \in \mathbb{R}^{m\times m}$ where $\text{cond}\,(\Sigma_E) \approx \mathcal{C}_m$. V_E is cropped to $\mathbb{R}^{(n-m)\times m}$ so that E has the correct size. All randomly generated problems have $n \geq 2m$ to perform this step. We then define $E := U_E \Sigma_E V_E^T$.

Next, we generate the optimal solution vectors. To guarantee strict complementarity, the primal vector x^* is generated with m strictly positive values, and dual vector z^* has $n-m$ strictly positive values with all remaining entries zero. The dual vector y^* is randomly generated with no restrictions. To obtain the complementary slackness holds, elements of x^* and z^* are permuted so that $(x^*)^T z^* = 0$.

We need to ensure that the final optimal basis remains well-conditioned. The primal optimal solution, x^*, is already known so subsequently the columns in A that form the

optimal basis is also known. Let β be the set of indices of columns of A that form the optimal basis with $|\beta| = m$. Also let

$$\beta_B := \{i \in \beta : i \leq m\}, \tag{5.1.3}$$
$$\gamma_B := \{i \notin \beta : i \leq m\}, \tag{5.1.4}$$
$$\beta_E := \{i \in \beta : i > m\}, \tag{5.1.5}$$
$$\gamma_E := \{i \notin \beta : i > m\}, \text{ and} \tag{5.1.6}$$
$$|\beta_B| := a. \tag{5.1.7}$$

A QR decomposition is performed on B_{basis}, the matrix consisting of basis columns in B, to obtain orthogonal matrix Q and upper triangular matrix R. Equivalently in MATLAB notation

$$B_{basis} := B[:, \beta_B].$$

We construct a new upper triangular block matrix, $\tilde{R}$, that consists of the first a rows of R, along with a well-conditioned diagonal matrix D. $\tilde{R}$ is given by

$$\tilde{R} := \begin{bmatrix} R[1:a,:]^{\S} & 0 \\ 0 & D \end{bmatrix}$$

where $D \in \mathbb{R}^{(m-a)\times(m-a)}$ is a randomly generated diagonal matrix with $\text{cond}(D) \approx \mathcal{C}_m$. The optimal basis of the random problem is given by $\Gamma := Q\tilde{R}$. The non-basis columns in E are then replaced by the better-conditioned columns in Γ. The complete construction of well-conditioned random problems are given in Algorithm 5.1.1.

5.2 NETLIB Problems

We performed numerical tests on a subset of the well-known NETLIB problems to evaluate effectiveness on sparse matrices. Since the modified stable algorithm is getting good performance on random dense matrices, we would expect even better performance if sparsity can be taken advantage of. The NETLIB problems are highly degenerate where 71% of

[§] In this work we use MATLAB notation to denote row and column indexing

Algorithm 5.1.1 Construction of well-conditioned dense random problems

▷ Well-condtioned B

Generate $U_B, V_B \in \mathbb{R}^{m\times m}$, orthogonal matrices
Generate $\Sigma_B \in \mathbb{R}^{m\times m}$, diagonal with $\text{cond}\,(\Sigma_B) \approx \mathcal{C}_m$
$B \leftarrow U_B \Sigma_B V_B^T$

▷ Well-condtioned E

Generate $U_E \in \mathbb{R}^{m\times m}$, orthogonal
Generate $V_E \in \mathbb{R}^{(n-m)\times(n-m)}$, orthogonal
Crop V_E to $\mathbb{R}^{(n-m)\times m}$
Generate $\Sigma_E \in \mathbb{R}^{m\times m}$, diagonal with $\text{cond}\,(\Sigma_E) \approx \mathcal{C}_m$
$E \leftarrow U_E \Sigma_E V_E^T$
$A \leftarrow \begin{bmatrix} B & E \end{bmatrix}$

▷ Ensuring optimality

Generate $x^* \in \mathbb{R}^n$ with m strictly positive values and $n-m$ zeroes
Generate $y^* \in \mathbb{R}^m$
Generate $z^* \in \mathbb{R}^n$ with $n-m$ strictly positive values and m zeroes
Permute elements of x^* and z^* such that $(x^*)^T z^* = 0$

▷ Ensuring optimal solution stays well-conditioned

Find $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{m\times a}$ such that $QR = B_{basis}$, where $B_{basis} \in \mathbb{R}^{m\times a}$ is the matrix of the basis columns in B. a is defined in (5.1.7)
Generate $D \in \mathbb{R}^{(m-a)\times(m-a)}$, diagonal with $\text{cond}\,(D) \approx \mathcal{C}_m$
$\Gamma \leftarrow Q\tilde{R}$, where $\tilde{R} := \begin{bmatrix} R[1:a,:]^{\S} & 0 \\ 0 & D \end{bmatrix}$
$A[:, \gamma_E] \leftarrow \Gamma[:, (a+1):n]^{\S}$, where γ_E is defined in (5.1.6)
$E \leftarrow A[:, (m+1):n]$
$b \leftarrow A^T x^*$
$c \leftarrow A^T y^* + z^*$
return A, b, c

the problems have infinite condition number [22]. This stable method performs better on non-degenerate problems.

To ensure that these problems were in the appropriate format given by (3.1.6), the `licols` function was used to find the linearly independent columns of A and then moved to the first m columns. This step accounted for less than 1% of total computation time. For the NETLIB comparisons we used a stopping criteria of 10^{-12} and a crossover tolerance of 10^{-3}.

Specifications		Stable			Mosek			LinProg		
m	n	CPU (s)	Itr	R-G	CPU (s)	Itr*	R-G	CPU (s)	Itr*	R-G
500	1000	0.465	19.4	3.58e-16	0.927	5.0	1.38e-10	3.183	7.0	3.15e-08
1000	2000	2.387	21.0	1.14e-15	4.728	5.0	2.84e-10	11.245	8.0	1.56e-08
1500	3000	7.230	23.2	4.20e-15	13.534	5.2	2.91e-09	36.543	8.6	8.84e-10
2000	4000	14.961	23.0	6.95e-16	29.297	5.2	3.37e-09	81.522	9.0	7.06e-13
2500	5000	29.705	24.0	1.81e-15	54.655	5.2	5.10e-10	152.018	9.0	8.36e-09
5000	10000	212.604	25.2	3.38e-15	447.508	5.4	2.00e-10	-	-	-
7500	15000	669.434	26.0	7.97e-15	1488.815	5.2	2.44e-09	-	-	-

Table 5.1: Average results over 5 dense random problems with $\mathcal{C}_m = \log(m)$. *Mosek and LinProg should have double number of iterations since they are using Mehrotra's Predictor Corrector method

5.3 Results

Table 5.1 compares averaged results for time, number of iterations and relative gap, labeled as CPU, Itr and R-G respectively. The updated stable method results are compared to the Mosek and LinProg solvers. Currently, Mosek is able to solve the widest variety of problems in the shortest time [20]. The random problems were generated using Algorithm 5.1.1 for different problem sizes. For each size the updated stable method solved the problem in the shortest time. The stable method performed over twice the number of iterations but could still find the optimal solution in the fastest time. Note that the number of iterations listed for Mosek and LinProg should be double since they employ Mehrotra's Predictor-Corrector. The results of the 2 larger problems for LinProg are not shown because it took a considerable amount of time to solve and were stopped before completion.

The relative gap for the stable method is several orders of magnitude smaller than both Mosek and LinProg. This is likely due to the default parameters for Mosek and LinProg. The stable method stopping criteria was less than 10^{-12} and when pure Newton steps were taken, the relative gap shrunk by several orders of magnitude at each step.

One thing we noticed is that when the condition number of A was significantly lower than $\mathcal{C}_m$, the stable method was able to solve the dense problem even faster despite the size of the matrix. This indicates that for large, very well-conditioned problems, the stable method would be able to solve dense problems very quickly.

Figure 5.1 shows performance profiles for solving time to solve 20 random dense problems of size 1100×2500. The stable method solved all the problems in the shortest amount

	Stable			Mosek			LinProg		
Problem	CPU (s)	Itr	R-G	CPU (s)	Itr*	R-G	CPU (s)	Itr*	R-G
afiro	0.012	16	8.682e-13	0.304	9	3.024e-12	0.012	7	1.902e-13
adlittle	0.164	24	9.099e-14	0.292	13	2.005e-10	0.015	12	1.147e-13
agg	0.404	44	3.991e-13	0.290	19	3.915e-09	0.037	20	7.293e-13
stocfor1	0.043	24	5.482e-13	0.205	8	3.441e-12	0.012	10	1.414e-12

Table 5.2: Comparisons of NETLIB problems for the 3 different solvers. *Mosek and LinProg should have double number of iterations since they are using Mehrotra's Predictor Corrector method

Problem	m	n	Crossed Over	cond (J_s)	Original cond (B)	Adjusted cond (B)
afiro	27	51	N	2.384e+07	∞	4.800e+00
adlittle	56	138	N	6.548e+23	∞	2.170e+06
agg	488	615	N	2.590e+14	∞	1.415e+05
stocfor1	117	165	N	1.460e+07	∞	2.266e+02

Table 5.3: Comparisons of NETLIB problems

of time and thus always had a value of 1 in the performance profiles (indicating the fastest solve time). SDPT3 is included in the figure but slower than the other solvers.

The results for the sparse NETLIB problems are given in Table 5.2. The modified stable algorithm performs very well on the sparse NETLIB problems despite the problems being ill-conditioned. For the sparse problems, LinProg was able to solve all the problems in the shortest amount of time. The number of iterations by the stable method is approximately the same for all 3 solvers (except stocfor1 problem) once the iterations are doubled due to MPC method. Figure 5.2 shows the performance profile for the NETLIB problems.

None of the NETLIB problems successfully crossed over into the region of quadratic convergence due to high condition numbers of J_s. Table 5.3 shows the condition numbers of J_s for the NETLIB problems when the crossover is checked. The condition number calculation accounted for 10-50% of the CPU solve time where the larger contribution was on the smaller problems. It also shows the condition number of the submatrix B before and after the columns were rearranged to obtain a full rank B matrix.

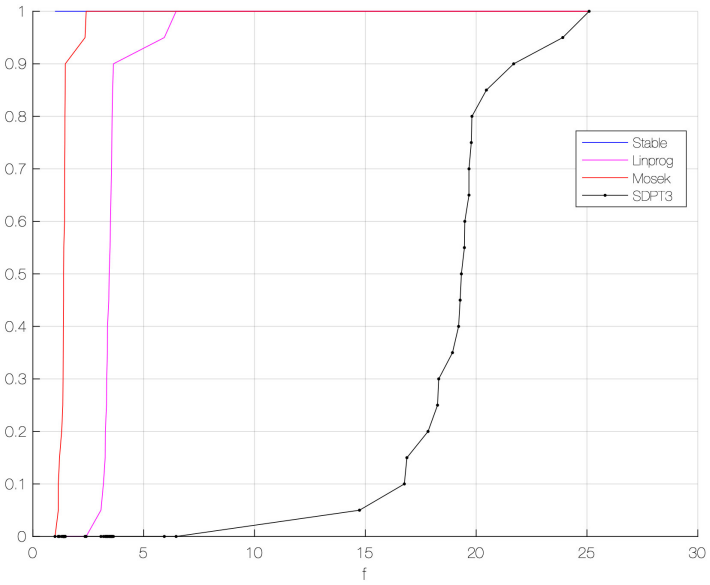

Figure 5.1: Performance profiles for 4 solvers for 20 random (dense) problems of size 1100x2500 with $\mathcal{C}_m = \log(m)$ for time to solve. The Stable method is a horizontal line at 1.0, indicating it solved the problems in the smallest amount of time

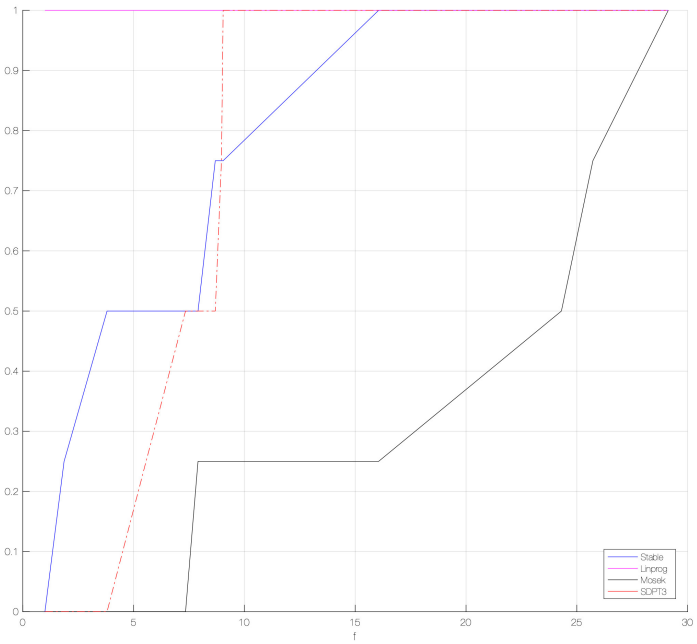

Figure 5.2: Performance profiles for 4 solvers for the sparse Netlib problems

Chapter 6

Future Work

Some potential areas where this work could be improved is to include exhaustive presolving to the method [24] to reduce the size of the problem before the iterative algorithm begins. Also, incorporating Mehrotra's Predictor-Corrector [19] into the algorithm could improve the stable method's performance.

Chapter 7

Conclusion

The update to the stable method has been shown numerically to be quite competitive with modern solvers on this class of nondegenerate problems. The method is computationally efficient since a reduced system is solved at each iteration and Newton steps are taken if the Jacobian remains well-conditioned. The updated stable method can take advantage of sparse problems and solve quickly without losing sparsity. An algorithm for creating well-conditioned random dense problems with a custom condition number has been presented and used in testing of this method. Our tests show that this method can take advantage of sparsity for large well-conditioned problems.

References

[1] Erling D. Andersen and Knud D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm*, pages 197–232. Springer US, Boston, MA, 2000.

[2] M. Benzi, G.H. Golub, and J. LIESEN. Numerical solution of saddle point problems. *Acta Numer.*, 14:1–137, 2005.

[3] R.E. Bixby. A brief history of linear and mixed-integer programming computation. *Doc. Math.*, (Extra vol.: Optimization stories):107–121, 2012.

[4] Robert G Bland, Donald Goldfarb, and Michael J Todd. The ellipsoid method: A survey. *Operations research*, 29(6):1039–1091, 1981.

[5] G. Dantzig. Reminiscences about the origins of linear programming. *Operations Research Letters*, 1:43–48, 1982.

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

[7] G.B. Dantzig. Origins of the simplex method. In *A history of scientific computing (Princeton, NJ, 1987)*, ACM Press Hist. Ser., pages 141–151. ACM, New York, 1990.

[8] G.B. Dantzig. Linear programming. In *History of Mathematical Programming: A Collection of Personal Reminiscences*. CWI North-Holland, Amsterdam, 1991.

[9] T. Davis. Lpnetlib, 2018 (accessed October 2018). https://www.cise.ufl.edu/research/sparse/matrices/LPnetlib/.

[10] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[11] Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM Journal on Matrix Analysis and Applications*, 9(4):543–560, 1988.

[12] Shu-Cherng Fang and Sarat Puthenpura. *Linear Optimization and Extensions: Theory and Algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[13] M. Gonzalez-Lima, H. Wei, and H. Wolkowicz. A stable primal-dual approach for linear programming under nondegeneracy assumptions. *Comput. Optim. Appl.*, 44(2):213–247, 2009.

[14] Nicholas Gould and Jennifer Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software (TOMS)*, 43(2), 2016.

[15] W. Hager. Condition estimates. *SIAM Journal on Scientific and Statistical Computing*, 5(2):311–316, 1984.

[16] N. Higham and F. Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1185–1201, 2000.

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

[18] L.G. Khachian. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 244:1093–1096, 1979.

[19] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optim.*, 2(4):575–601, 1992.

[20] H. Mittelmann. Benchmark of barrier lp solvers, 2019 (accessed September 2019). http://plato.asu.edu/ftp/lpbar.html.

[21] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[22] F. Ordonez and R. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM Journal on Optimization*, 14(2):307–333, 2003.

[23] Steffen Rebennack. *Ellipsoid method*, pages 890–899. Springer US, Boston, MA, 2009.

[24] Vishnu V. Sadhana. Efficient presolving in linear programming. Master's thesis, University Of Florida, Florida, 2002.

[25] Timothy Sauer. *Numerical Analysis*. Pearson, 2018.

[26] R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using sdpt3. *Mathematical Programming*, 95(2):189–217, Feb 2003.

[27] S. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, Pa, 1996.