Large-Scale Computing for Complementarity and Variational Inequalities

By

Qian Li

A dissertation submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

(Mathematics)

at the

UNIVERSITY OF WISCONSIN – MADISON

2010

Abstract

Both complementarity problems and variational inequalities are tools for expressing the equilibrium conditions in diverse engineering and economic systems. They also play an important role in constrained optimization problems, encompassing the optimality conditions for linear and nonlinear programs and are necessary and sufficient conditions for convex problems. This thesis is concerned with enhancing and designing solvers with large-scale computing capability to process classes of the above two types of problems respectively.

One aspect of this research aims at enhancing the efficiency and reliability of PATH, the most widely used solver for mixed complementarity problems. A key component of the PATH algorithm is solving a series of linear complementary subproblems with a pivotal scheme. Improving the efficiency of the linear system routines (factor, solve, and update) required by the pivotal scheme is the critical computational issue. We incorporate two new options besides the default LUSOL package in PATH for such functionality. One of the options employs the UMFPACK package for factor and solve operations. UMFPACK is known to be both time and memory efficient in finding the solution of large sparse sets of unsymmetric linear equations and it has been incorporated as a built-in operator in MATLAB for this purpose. To provide the update operation to this option, we implement a stable and efficient block-LU updating scheme, which can also be combined with other factor and solve packages for more general use. This option leads to a significantly more effective version of PATH for solving many large-scale sparse systems. The other option exploits the COIN-OR utilities enhanced by adapting the linear refinement and scaling schemes used in the COIN-LP routines. This alternative is motivated by COIN-LP's better performance than the LUSOL-based MINOS/SNOPT in solving linear programs. The COIN option is effective in solving smaller-scale systems but less competitive on large-scale cases. These options are incorporated in the new GAMS/PATH distribution.

As more variational inequalities have been recognized within applications such as Nash equilibrium problems, electricity pricing and traffic equilibrium models, the need for a robust large-scale variational inequality solver has grown more pressing. This research therefore is also concerned with building a new large-scale affine variational inequality solver (PathAVI). Current methods for processing affine variational inequalities involve transforming the problem into a linear complementarity problem and solving the equivalent transformed problems using an existing complementarity solver. However often the original problem structure is lost during this transformation and in some cases, the resulting problem is hard or even unsolvable for existing solvers. PathAVI is designed to exploit the special structure of the underlying polyhedral set and is able to process a class of models (formulated as AVIs) whose equivalent complementarity reformulations cannot be processed by existing complementarity solvers. It constructs a piecewise linear path in the normal manifold associated with the polyhedral set to find a zero of the normal map induced by the AVI. Theoretical justifications are given to show that this algorithm is guaranteed to either find a solution or determine infeasibility in finite number of steps for problems involving matrices that are copositive-plus with respect to the recession cone of the polyhedral set. The central component of the algorithm is to employ a pivotal strategy to construct the piecewise linear path. Therefore it critically relies on linear system packages for its performance. Similar to the PATH case, sparse linear system packages and updating schemes are incorporated into the new solver and comparisons among different basis options are presented. Development of this new solver for solving affine cases is the key building block for a nonlinear variational inequality solver. Extension of the PathAVI scheme to solving nonlinear variational inequalities is proposed.

Finally, this research presents a MATLAB implementation for solving parametric monotone linear complementarity problems. The problem is to consecutively solve a sequence of linear complementarity problems as the parameter value varies over a feasible range determined by the solution of two linear programs. The algorithm is based on the Lemke's method and underlying theoretical justifications are also given. Application of this algorithm to portfolio selection and other problems is described in detail. Since linear complementarity problems are equivalent to affine variational inequalities, instead of solving a sequence of linear complementarity problems we can solve a parametric linear complementarity problem as a sequence of affine variational inequalities. For certain classes of problems, this alternative is potentially advantageous.

Acknowledgements

I would like to express my sincere gratitude and thanks to my advisor Professor Michael Ferris for his guidance, counsel and encouragement at every stage of this research. I am grateful to Steven Dirkse and Todd Munson who provided numerous insightful comments on the work related to the PATH solver. I have also benefitted from interaction with Tim Davis, John Forrest and Michael Saunders who provided valuable suggestions related to the linear system solvers they maintain.

I would like to thank Professors Shi Jin, Stephen Robinson, James Rossmanith and Stephen Wright for serving on my final examination committee, and for their invaluable input and advice.

My stay at UW-Madison was made both enjoyable and memorable by my friends and officemates - Weiwei Chen, Geng Deng, Jesse Holzer, Mahdi Namazifar, Christina Oberlin, Meng Shi, Lisa Tang. I enjoyed having technical discussions as well as social get-togethers with them.

I am most grateful to my husband Howard for his encouragement, understanding, patience and assistance during the significant part of this research work. In addition, I express thanks to my parents, for their efforts, inspiration and love, which makes everything possible. I affectionately dedicate this work to them.

The work in this thesis was supported by AFOSR Grant FA9550-07-1-0389 and NSF Grant CMMI-0928023.

Madison, Wisconsin Qian Li

December, 2009

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Complementarity and variational inequality are often recognized as the first order optimality conditions for linear and nonlinear programs. In fact the study of complementarity originated from the Karush-Kuhn-Tucker conditions in linear programming and quadratic programming [54, 56]. Besides this feature, complementarity problems and variational inequalities are closely related and are prominently used to express system equilibrium. Some of the earliest studies of such properties and algorithm development for solving these systems are associated with game theory [11, 57, 58], following which the scope of complementarity and variational inequality expanded extensively to numerous applications from engineering, economics and finance. A number of review documents of these applications and developments have appeared over the years [10, 12, 28, 29, 36, 49]. Therefore there is a great deal of practical interests in developing robust and efficient algorithms for processing these problems.

This research is mainly concerned with two classes of problems from the above categories, namely mixed complementarity problems and affine variational inequalities.

For mixed complementarity problems, a number of environments have been made available for general users and researchers to express and solve such problems. For example, modeling languages such as AMPL [41] and GAMS [3] provide natural facilities for expressing mathematical programs including mixed complementarity problems; other tools such as MATLAB [60] and NEOS [13, 33] can also be used to represent complementarity relationships. The most widely used mixed complementarity solver, capable of supporting all of the above environments, is called the PATH solver [22, 34]. PATH is a generalized Newton method for solving mixed complementarity problems and has been effective at solving relatively difficult problems (see, for example, [2, 69]). For larger problems, however, the numerical linear algebra is inadequate to obtain good performance. Therefore one aspect of this research is focused on enhancing the efficiency and reliability of PATH. Since PATH essentially solves a linear complementarity problem using Lemke's method at each generalized Newton step, a key computational issue is the efficiency of the linear algebra routines (factor, solve and update) required by the pivotal method. We thus enhance PATH by incorporating new basis options containing these necessary linear algebra routines based on efficient linear system packages and updating schemes.

Current methods for processing an affine variational inequality problem are generally based on a transformation of the original problem into an equivalent linear complementarity problem. The resulting problem is then processed by complementarity solvers. This method encounters difficulties for certain classes of affine variational inequalities because the transformation to some extent ignores the underlying structure of the original problem and generates an equivalent linear complementarity problem which is much harder for existing complementarity solvers to process. Variational inequalities (or equivalently generalized equations) are closely related to the well-known minimum principle optimality conditions of linear and nonlinear programming. The advantage of using variational inequalities to express the optimality conditions is clear if we consider the constraint set as a geometric object, in that, a specifically designed variational inequality solver will exploit this object explicitly. In the case of a complementarity reformulation, extra parameters are introduced. Therefore the transformation changes the feasible set of the problem and expresses the original structure implicitly. From a primal-dual perspective, a variational inequality solver only searches in the primal space while a complementarity solver looks for solutions in an augmented primal-dual space. Besides its use in expressing the optimality conditions of a single problem, variational inequality is especially useful for applications concerning system equilibrium, where a set of objectives need to be optimized at the same time, for example the Nash game and electricity pricing models. Due to the above advantages of (affine) variational inequalities, more interest has been shown for a new solver specifically designed to exploit the structure of such problems. Cao and Ferris proposed a scheme for solving affine variational inequalities [5] based on a realization of a more general scheme due to Eaves [25], which can also be considered as a generalization of Lemke's pivotal method. This method is further extended by Cao and Ferris in [4] to remove certain invertibility assumptions made upon the problem matrix. Unfortunately, due to the reduction schemes employed by this method, the solver, which is an implementation in MATLAB, is inefficient as the problem size becomes larger. Therefore this thesis is also concerned with the development of a robust large-scale affine variational inequality solver, which is proven to be able to process as wide a class of affine variational inequality problems as the Cao & Ferris method does.

This chapter begins with the definition of the mixed complementarity problems together with a list of special cases in Section 1.1.1. The definition of affine variational inequalities follows in Section 1.1.2 and the relationship between these two classes of problems is then discussed in Section 1.1.3. A bimatrix game is given in Section 1.2. It serves as an example for both classes of problems and also gives a more concrete illustration to their relationship. Both methods for processing mixed complementarity problems and affine variational inequality problems rely heavily on their equivalent representation as an unconstrained system of nonsmooth equations using the normal map [24], and on the normal manifold [66] associated with their underlying set structures. These notions are the topics of Section 1.3 and Section 1.4 respectively. Finally, Section 1.5 outlines the remainder of this thesis.

Before proceeding, we briefly explain some notation used in this thesis. We use I to denote an index set. To avoid confusion, an identity matrix will be denoted by $\mathbb{I}$. Index sets named $\mathcal{I}$ and $\mathcal{A}$ are related to inactive and active constraint sets respectively.

1.1 Definitions

1.1.1 Complementarity

The **Mixed Complementarity Problem** (MCP) is defined by the set $\mathbf{B} := \{z \in \mathbb{R}^n | \, l \leq z \leq u\}$ with bounds $l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{\infty\}$ such that $l_i \leq u_i$ for all $i = 1, \dots, n$, and a function $F : \mathbf{B} \mapsto \mathbb{R}^n$. A vector $z \in \mathbb{R}^n$ is a solution of $MCP(F, [l, u])$ if and only if one of the following holds for each $i = 1, \dots, n$:

$$\begin{aligned} l_i \leq z_i \leq u_i \quad &\text{and} \quad F_i(z) = 0 \\ z_i = l_i \quad &\text{and} \quad F_i(z) > 0 \\ z_i = u_i \quad &\text{and} \quad F_i(z) < 0. \end{aligned} \tag{1.1.1}$$

Note that if $l_i = u_i$, then $z_i = l_i = u_i$ is a fixed value and any $F_i(z)$ satisfies (1.1.1).

We denote this relationship in short by

$$l \leq z \leq u \perp F(z),$$

where $\perp$ is used to express orthogonality.

Special cases of mixed complementarity problems include:

- **System of Equations** $F(z) = 0$, when $\mathbf{B} = \mathbb{R}^n$;
- **Nonlinear Complementarity Problem**, when $\mathbf{B} = \mathbb{R}^n_+$;
- **Linear Complementarity Problem**, when $\mathbf{B} = \mathbb{R}^n_+$ and F is affine, that is

$$F(z) = Mz + q$$
$$0 \leq z \perp F(z) \geq 0$$

  with $M \in \mathbb{R}^{n \times n}$ and $q \in \mathbb{R}^n$.

Here $\mathbb{R}^n_+$ is the positive orthant $\{z | z_i \geq 0\,,\, i = 1, \cdots, n\}$.

1.1.2 Variational Inequality

A **Variational Inequality** can be defined as follows. Consider a nonempty set $\mathcal{C} \subset \mathbb{R}^n$ and a function $F := \mathbb{R}^n \mapsto \mathbb{R}^n$, $VI(F, \mathcal{C})$ is to find a $z \in \mathcal{C}$ such that

$$\langle F(z), y - z \rangle \geq 0, \quad \forall y \in \mathcal{C}. \tag{1.1.2}$$

In this thesis we will only consider $VI(F, \mathcal{C})$ with a continuously differentiable F over a nonempty, closed and convex set $\mathcal{C}$. In particular, we are mostly interested in solving **Affine Variational Inequality** $AVI(M, q, \mathcal{C})$ on a polyhedral set $\mathcal{C}$, namely $F(z) = Mz - q$ is an affine map on set $\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b, Hz = h\}$. The study of the affine case is important since it is the key building block of the nonlinear cases.

The variational inequality defined in (1.1.2) is equivalent to a **Generalized Equation**:

$$0 \in F(z) + N_{\mathcal{C}}(z), \tag{1.1.3}$$

where $N_{\mathcal{C}}(z)$ is the **Normal Cone** to $\mathcal{C}$ at z.

$$N_{\mathcal{C}}(z) := \begin{cases} \{x \in \mathbb{R}^n | \text{ for each } y \in \mathcal{C}, \langle x, y - z \rangle \leq 0\} & \text{if } z \in \mathcal{C} \\ \emptyset & \text{if } z \notin \mathcal{C} \end{cases}$$

Take a simple example with the box constraints $\mathbf{B} := \{z \in \mathbb{R}^n | \, l \leq z \leq u\}$, the normal cone to the set is then comprised of a Cartesian product:

$$N_{\mathbf{B}}(z) = \prod_{i=1}^{n} N_i(z_i).$$

where each component depends upon the values of z_i vis-a-vis its bounds l_i and u_i:

$$N_i(z_i) = \begin{cases} \mathbb{R} & \text{if } l_i = z_i = u_i \\ \mathbb{R}_- & \text{if } l_i = z_i < u_i \\ \{0\} & \text{if } l_i < z_i < u_i \\ \mathbb{R}_+ & \text{if } l_i < z_i = u_i \\ \emptyset & \text{otherwise .} \end{cases}$$

Here $\mathbb{R}_-$ and $\mathbb{R}_+$ denote the negative and positive half lines respectively.

Let us consider a slightly more complicated example with a polyhedral set $\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b\}$. A **Face** $\mathcal{F}_i$ of set $\mathcal{C}$ is described by a set of **Active Constraints** associated with some particular z value:

$$\mathcal{A} := \{j \in \{1 \cdots n\} \,|\, B_{j,\cdot}\, z = b_j\},$$

where the subscript $(j, \cdot)$ means the j^{th} row of the matrix. Therefore face i of the set $\mathcal{C}$ can be written as:

$$\mathcal{F}_i = \{z | B_{\mathcal{A}} z = b_{\mathcal{A}}, B_{\mathcal{I}} z \geq b_{\mathcal{I}}\},$$

where $\mathcal{I}$ is the complement of $\mathcal{A}$. The normal cone to the relative interior of this face can be described as:

$$N_{\mathcal{F}_i} = \{-B^\top u | u_{\mathcal{A}} \geq 0, u_{\mathcal{I}} = 0\}.$$

The relative interiors of the faces form a partition of $\mathcal{C}$. The normal cone to the set $\mathcal{C}$ at a particular point z is thus

$$N_{\mathcal{C}}(z) = N_{\mathcal{F}_i}(z),$$

where $\mathcal{F}_i$ is the face that contains z in its relative interior.

1.1.3 Relationship Between AVI and MCP

An affine variational inequality problem can be reformulated as an LCP, which is a special case of the mixed complementarity problem listed above. To illustrate, we use an affine variational inequality defined with $F(z) = Mz - q$ and a polyhedral set $\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b\}$. An equivalent linear complementarity problem to $AVI(M, q, \mathcal{C})$ can be constructed by introducing a variable u:

$$\begin{aligned} 0 = Mz - q - B^\top u \quad &\perp \quad z \text{ free} \\ 0 \leq Bz - b \quad &\perp \quad u \geq 0. \end{aligned}$$

This linear complementarity problem can be written in the form of an augmented generalized equation:

$$0 \in \begin{bmatrix} M & -B^\top \\ B & \end{bmatrix} \begin{bmatrix} z \\ u \end{bmatrix} - \begin{bmatrix} q \\ b \end{bmatrix} + N_{\mathbb{R}^n \times \mathbb{R}^m_+}\left(\begin{bmatrix} z \\ u \end{bmatrix}\right),$$

with correspondingly an augmented set $\mathbb{R}^n \times \mathbb{R}^m_+$.

This transformation allows problems originally formulated as affine variational inequalities to be processed by existing linear or mixed complementarity solvers. However it may sometimes be less desirable because the original structure of the polyhedral set is lost and the transformed problem with an augmented and unbounded set may be harder to solve. Therefore an algorithm specifically designed to exploit the original underlying structure of affine variational inequalities is studied in this thesis.

1.2 A Bimatrix Game

An interesting application arises from a simple case of Nash games, which is also known as the bimatrix game: two players have I and J pure strategies and they determine the probability with which each of their pure strategies is played to minimize their costs (or maximize their profits), a so-called mixed strategy. The player's mixed strategies are denoted by p and q that belong to unit simplex $\triangle_I$ and $\triangle_J$ respectively. That is

$$\triangle_I = \{p \in \mathbb{R}^I \mid \sum_{i=1}^{I} p_i = 1, p_i \geq 0 \ \forall i\}$$

and similarly for $\triangle_J$. Furthermore, let their loss matrices be $A \in \mathbb{R}^{J\times I}$ and $B \in \mathbb{R}^{I\times J}$, where A_{ji} is the cost incurred by the first player if he/she chooses pure strategy i and the second player chooses pure strategy j, and B_{ij} is the cost incurred by the second player. The expected costs for the first and the second players are $q^\top Ap$ and $p^\top Bq$ respectively.

A Nash equilibrium is reached by the pair of strategies (p^*, q^*) if and only if

$$p^* \in \arg \min_{p\in\triangle_I} \langle Aq^*, p\rangle \quad \text{and} \quad q^* \in \arg \min_{q\in\triangle_J} \langle B^\top p^*, q\rangle.$$

Combining the KKT conditions of both systems gives a mixed linear complementarity problem as follows:

$$\begin{aligned} 0 \leq Aq - e_I u \quad &\perp \quad p \geq 0 \\ e_I^\top p = 1 \quad &\perp \quad u \text{ free} \\ 0 \leq B^\top p - e_J v \quad &\perp \quad q \geq 0 \\ e_J^\top q = 1 \quad &\perp \quad v \text{ free} \end{aligned}$$

where e_I and e_J are vectors of ones and u and v are scalar parameters.

A reduction method is generally used in the literature for solving this special instance, which considers the following standard linear complementarity problem:

$$\begin{aligned} 0 \le Aq - e_I \quad &\perp \quad p \ge 0 \\ 0 \le B^\top p - e_J \quad &\perp \quad q \ge 0. \end{aligned}$$

The solution to the original game (p^*, q^*) is recovered by normalizing the solution (p°, q°) obtained from the above reduced system:

$$p^* = \frac{p^\circ}{e_I^\top p^\circ}, \quad q^* = \frac{q^\circ}{e_J^\top q^\circ}.$$

The AVI formulation of the optimality conditions on the other hand arises naturally from this bimatrix game. We express the optimality conditions of the above set of linear programs by generalized equations:

$$-Aq^* \in N_{\triangle_I}(p^*) \text{ and } -B^\top p^* \in N_{\triangle_J}(q^*).$$

Therefore the corresponding VI is affine and can be written as:

$$0 \in \begin{bmatrix} 0 & A \\ B^\top & 0 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} + N_{\triangle_I \times \triangle_J}\left(\begin{bmatrix} p \\ q \end{bmatrix}\right). \tag{1.2.1}$$

Note that in more general settings, for example

$$\begin{array}{ll} \min\limits_p \theta^1(p,q) & \qquad \min\limits_q \theta^2(p,q) \\ \text{s.t. } p \in K^1 & \qquad \text{s.t. } q \in K^2 \end{array}$$

where the cost function θ^1 contains other linear function terms that are only associated with variable p (and similarly for θ^2 with variable q) and K^1, K^2 are intersections of the unit simplex with other polyhedral constraint sets. The optimality conditions expressed as a KKT system is an augmented linear complementarity problem over an augmented feasible set, to which the reduction method cannot be applied. The AVI formulation on the other hand still works on the original feasible set $K^1 \times K^2$. More generally still, θ^1 and θ^2 could be convex functions of p and q respectively and K^1 and K^2 are convex sets, and then solving $VI(F, \mathcal{C})$ with

$$F := \begin{bmatrix} \nabla_p \, \theta^1(p,q) \\ \nabla_q \, \theta^2(p,q) \end{bmatrix}, \qquad \mathcal{C} = K^1 \times K^2$$

is equivalent to solving the game. Note that ∇ denotes the gradient.

When the number of players in a Nash game increases to 3 or more, the corresponding complementarity and variational inequality formulations of its optimality conditions are no longer affine. In general, Nash games are proven to be hard problems with complexity PPAD-Complete [7, 8, 15, 16].

1.3 Normal Map

The **Normal Map** is defined as:

$$F_{\mathcal{C}}(x) := F(\pi_{\mathcal{C}}(x)) + (x - \pi_{\mathcal{C}}(x)) \tag{1.3.1}$$

where $F := \mathbb{R}^n \to \mathbb{R}^n$ and $\mathcal{C}$ is a non-empty, closed, convex set. Note that $\pi_{\mathcal{C}}(\cdot)$ is the Euclidean projection onto $\mathcal{C}$, defined by

$$\pi_{\mathcal{C}}(x) = arg \min_{x} \{||x - z||_2 \,|\, z \in \mathcal{C}\}.$$

Both algorithms considered in this thesis process MCP and AVI problems by exploiting their reformulations via normal maps, due to the following results.

Solving an $MCP(F, [l, u])$ is equivalent to finding a zero of the normal map with $\mathcal{C} = [l, u]$ a box set, in particular we have

Theorem 1.3.1. *Let $MCP(F, [l, u])$ be given, then the following hold:*

- *if $x \in \mathbb{R}^n$ is a zero of the normal map such that*

$$F_{[\mathbf{l},\mathbf{u}]}(x) = 0,$$

  then $z := \pi_{[l,u]}(x)$ solves $MCP(F, [l, u])$;

- *if z solves $MCP(F, [l, u])$, then $x = z - F(z)$ is a zero of the normal map (1.3.1).*

Solving an $AVI(M, q, \mathcal{C})$ or its corresponding generalized equation (1.1.3) is equivalent to finding a zero of its normal map with a polyhedral set $\mathcal{C}$, in that

Theorem 1.3.2. *Let $AVI(M, q, \mathcal{C})$ be given, then the following hold:*

- *if $x \in \mathbb{R}^n$ is a zero of normal map, then $z := \pi_{\mathcal{C}}(x)$ solves $AVI(M, q, \mathcal{C})$;*

- *if z solves $AVI(M, q, \mathcal{C})$, then $x = z - (Mz - q)$ is a zero of the normal map (1.3.1).*

Note that the domain of the normal map (1.3.1) is the whole space, whereas the original problem is defined over a subset in the whole space.

1.4 Normal Manifold

A key feature of the algorithms we employ in this thesis is their ability to exploit the geometric structure of the underlying box constraint set in the mixed complementarity problem case and polyhedral set in the affine variational inequality case. An important notion is the **Normal Manifold** [66].

Let $\mathcal{C} \in \mathbb{R}^n$ be a nonempty polyhedral convex set. As before, let $\{\mathcal{F}_i | i \in \mathbb{F}\}$ be the nonempty faces of $\mathcal{C}$. The relative interiors of these faces $\mathcal{F}_i$ form a partition of $\mathcal{C}$. For $i \in \mathbb{F}$, the normal cone $N_{\mathcal{F}_i}$ is constant on the relative interior of face $\mathcal{F}_i$. Let

$\sigma_i := \mathcal{F}_i + N_{\mathcal{F}_i}$, the normal manifold $\mathcal{N}_{\mathcal{C}}$ of $\mathcal{C}$ is comprised of the collection of these polyhedral sets σ_i and it covers $\mathbb{R}^n$. Each σ_i is called a cell of $\mathcal{N}_{\mathcal{C}}$ and $\mathcal{N}_{\mathcal{C}}$ is a subdivided piecewise linear manifold of dimension n.

For example, when the feasible set is the positive orthant $\mathbb{R}^n_+$, $\mathcal{N}_{\mathbb{R}^n_+}$ is comprised of the orthants of $\mathbb{R}^n$. In the case of a polyhedral set $\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b\}$, the normal manifold $\mathcal{N}_{\mathcal{C}}$ covers the whole space, and the relative interiors of the faces $\mathcal{F}_i$ index a partition of the normal manifold. In particular, each cell σ_i is generated corresponding to the active set $\mathcal{A}_i$ that defines F_i and $N_{\mathcal{F}_i}$:

$$\mathcal{F}_i = \{z | B_{\mathcal{A}_i} z = b_{\mathcal{A}_i}, B_{\mathcal{I}_i} z \geq b_{\mathcal{I}_i}\},$$

$$N_{\mathcal{F}_i} = \{-B^\top u | u_{\mathcal{A}_i} \geq 0, u_{\mathcal{I}_i} = 0\}.$$

In addition, when the function F is affine, the normal map $F_{\mathcal{C}}$ will agree in each cell of the normal manifold with an affine map. Both algorithms for dealing with the complementarity problems and affine variational inequalities essentially traverse the normal manifold, as each pivot corresponds to entering a new cell and therefore changing the affine map that currently represents the normal map. We show in the sequel the form of these changes and how to implement them efficiently.

1.5 Outline

We have presented the definitions of the mixed complementarity problems and the affine variational inequality problems and discussed their relationships, together with a bimatrix game example from the application point of view. We also illustrated that a method for processing these classes of problems is to solve their corresponding normal map equations, based on the properties of their associated normal manifolds.

The remainder of this thesis is focused on the solvers for the two classes of problems, namely the existing PATH solver for mixed complementarity problems and the new PathAVI solver for affine variational inequalities.

Both solvers essentially involve a sequence of pivots, so a key component of the solvers is a basis package which contains all the linear system routines (factor, solve and update) required by the pivotal method. In order to enhance the performance of both solvers, we consider a variety of basis package options supported by different linear system packages. This is motivated by the work in enhancing the PATH solver, which is the topic of Chapter 3. The descriptions of the basis package and the different options we consider are contained in Chapter 2. In particular, LUSOL and two new basis options (UMFPACK and COIN) for sparse systems, together with a Dense option for dense systems are presented.

Chapter 3 gives a description of the PATH algorithm in more detail and documents our work in enhancing the efficiency and reliability of PATH. As mentioned before, improving the efficiency of the basis package routines required by the pivotal method is the critical computational issue. We compare the default LUSOL option in PATH with two new options (UMFPACK and COIN). Computational results show that PATH/UMFPACK option is more efficient than PATH/LUSOL at solving most large-scale complementarity problems, whereas PATH/COIN is effective at solving smaller-scale systems but surprisingly is less effective than the other options in solving large-scale problems.

The new solver (PathAVI) for processing affine variational inequalities is the subject of Chapter 4. It employs a path-following method originally proposed by Cao and Ferris [5]. A review of the original Cao & Ferris method together with the underlying theory is presented, which motivates the development of a new solver that is more appropriate for solving large-scale instances. As mentioned before, the solution path is generated by a pivotal strategy similar to Lemke's method. Therefore the new solver also relies on the same basis packages for all the necessary linear system operations. This feature constitutes an important factor in enhancing the efficiency of the new solver. Performance of different basis options is compared.

In Chapter 5, we consider an application of the two classes of problems (mixed complementarity problems and affine variational inequalities) in a slightly different context, namely the solution of parameterized linear complementarity problems. In particular, a sequence of related linear complementarity problems (and in some cases affine variational inequalities) needs to be solved. We present in detail an implementation which solves parameterized linear complementarity problems for all feasible values of the parameter automatically, together with some theoretical justification.

Chapter 2

Basis Packages

The main algorithms for processing both mixed complementarity problems and variational inequalities essentially employ a generalized Lemke's method, which involves a sequence of complementary pivotal steps. Therefore the key issue in both of the solvers lies in factoring and solving linear systems of equations and performing rank-one updates efficiently. We briefly illustrate these operations associated with the pivotal steps using a simplified version of the mixed linear complementarity systems (3.1.6) in the main PATH algorithm. The construction of this system will be described later in Chapter 3.

$$\begin{aligned} Mz(t) + q - w(t) + v(t) - (1-t)r &= 0 \\ 0 \le w \perp z &\ge l \\ 0 \le v \perp z &\le u \\ z \in [l,\, u], w \ge 0, v &\ge 0. \end{aligned} \tag{2.0.1}$$

This system has a complementary triple (z, w, v) and it is parameterized by $t \in [0\ 1]$. The algorithm constructs an initial guess for the basis using the active set corresponding to the starting point or the active set computed from the crash process (which will be discussed in Chapter 3). The variables (z, w, v) are partitioned into three sets according to the active set. Suppose the dimension of the problem is n,

1. $W = \{i \in \{1, \cdots, n\} \,|\, \tilde{z}_i = l_i \text{ and } \tilde{w}_i > 0\}$

2. $V = \{i \in \{1, \cdots, n\} \,|\, \tilde{z}_i = u_i \text{ and } \tilde{v}_i > 0\}$

3. $Z = \{1, \cdots, n\} \setminus W \cup V$.

Since $(\tilde{z}, \tilde{w}, \tilde{v})$ are complementary, $W \cap V \cap Z = \emptyset$ and $W \cup V \cup Z = \{1, \cdots, n\}$. Therefore we determine the initial basis by

- taking the Z column set from M, namely $M_{\cdot Z}$, associated with inactive z variables (that is z strictly in between bounds);
- taking the W column set from $-\mathbb{I}$, namely $-\mathbb{I}_{\cdot W}$, associated with strictly positive w variables;
- taking the V column set from $\mathbb{I}$, namely $\mathbb{I}_{\cdot V}$, associated with strictly negative v variables.

In the end the initial linear system of equations corresponding to the above basis selection is

$$[M_{\cdot Z} \quad -\mathbb{I}_{\cdot W} \quad \mathbb{I}_{\cdot V}] \begin{bmatrix} z_Z \\ w_W \\ v_V \end{bmatrix} = -q + (1-t)\, r.$$

With an advanced (regular) start, the algorithm chooses a vector r to make the system feasible with $t = 0$. We factor the initial basis in order to provide starting factors for subsequent updates and to report possible singularities. We postpone the discussion of the recovery of an invertible basis to Chapter 3. The algorithm then starts by letting t enter the basis with the goal of increasing it to 1, which will correspond to a complementary solution to the system (2.0.1). As t changes (increasing from 0 up to 1), at each subsequent pivot, a basic variable hits its bound and leaves the basis, and its complementary variable enters the basis. The above pivotal step is achieved by a column replacement in the basis which requires rank-one update and solve routines. Explicitly, let $\{\tilde{W}\ \tilde{V}\ \tilde{Z}\}$ denote the new basic column set after a single pivot, the new basis matrix can be expressed as follows:

$$[M_{\cdot\tilde{Z}} \quad -\mathbb{I}_{\cdot\tilde{W}} \quad \mathbb{I}_{\cdot\tilde{V}}] = [M_{\cdot Z} \quad -\mathbb{I}_{\cdot W} \quad \mathbb{I}_{\cdot V}] + ab^{\top}$$

where a is a column vector denoting the difference between the entering and leaving basic columns and b is a unit vector representing the location of this update in the basis. Similarly the main algorithm of the new AVI solver also requires factor, solve and update functionalities for performing the pivotal scheme on its complementarity system. These routines are contained in a basis package which is in part common to both solvers.

Section 2.1 describes the factor, solve and update operations in general and the functions contained in the basis object. A variety of linear algebra packages are considered as the providers. The details of their implementations in the basis package are discussed in each of the following sections respectively.

2.1 Basis Operations

In order to solve a basis system $Ax = b$, an LU triangular factorization of A is often performed, followed by forward and backward substitutions with triangular matrices in finding the solution x. In many applications, instead of a single system, a sequence of related systems needs to be solved, in which each A is subject to a rank-one modification from its predecessor. The most important type of modification is a column replacement in the basis. Besides its application in the complementarity pivots employed by the solvers for processing MCPs and AVIs, column replacement is critical to a number of other applications, including the simplex method for linear programming [14], reduced-gradient method for linearly constrained optimization problems [70] and fixed-point algorithms for solving nonlinear equations, etc. The central idea for solving a sequence of linear systems with rank-one updates efficiently is to modify the existing factors of the initial basis subsequently instead of refactoring the new basis after each update. A refactorization is only needed after a certain number of updates have been taken due to accuracy and/or storage considerations.

Different approaches in performing efficient factor, solve and update operations for linear systems have been studied. In the scope of this thesis we consider four basis package options for providing certain functionality expressed via the following functions:

Basis_Factor(): Factors the given basis matrix.

Basis_Solve(): Uses the factors to solve a linear system of equations.

Basis_Replace(): Replaces a column of the basis matrix.

Basis_NumSingular(): Indicates the singular row(s) and column(s) of the basis matrix when singularity is detected. The purpose of this operation is that when the initial basis is singular, we can substitute the singular column(s) by column(s) from an identity matrix to recover an invertible basis. This identity matrix corresponds to a set of artificial variables added to the (mixed) linear complementarity system that are fixed at zero. This addition will be illustrated later in Chapter 3 and Chapter 4.

2.2 LUSOL

The LUSOL routines [44] are based on a Markowitz factorization and a Bartels-Golub update [1, 64, 65]. They are used within optimization packages MINOS and SNOPT, and complementarity packages MILES and PATH to provide linear algebra functionalities associated with the basis systems. The major functions provided by the LUSOL routines are as follows:

Factor For a given sparse matrix $A_{m \times n}$, computes a factorization $A = LU$ by Gaussian elimination with a Markowitz pivotal strategy to choose permutations P and Q, so that PLP' is lower triangular and $P'UQ$ is upper triangular (when $m = n$) or upper trapezoidal (when A is rectangular).

Solve For a given vector b_m, uses the LU factors to find a vector x_n that solves the linear system $Ax = b$.

Update Modifies L and U to obtain a new factorization $A = LU$ when A is updated. The updates include addition, deletion, replacement of a column or row of matrix A and rank-one modification.

If matrix A is singular or ill-conditioned, the return status from the factorization routine will indicate the detection of singularity. Since the dimensions and condition of A are almost always reflected in U, the number of "apparent" singularities is taken to be the number of the "small" diagonals of the permuted U. This number, together with the positions of such elements, is also returned by the factorization routine. The *Basis_NumSingular*() routine is designed to use this information to determine the corresponding singular row(s) and column(s) of A.

2.3 UMFPACK

UMFPACK is a set of routines for solving unsymmetric sparse linear systems [17, 18, 19, 20]. It is based on the unsymmetric multifrontal method and direct sparse LU factorization. The primary UMFPACK routines required to factorize A and/or solve $Ax = b$ are as follows:

Factor For a given sparse matrix A, performs a column preordering to reduce fill-in and a symbolic factorization. Then performs a numerical factorization, $PAQ = LU$, $PRAQ = LU$ or $PR^{-1}Q = LU$, where R is a diagonal matrix of scale factors, P and Q are permutation matrices, L is lower triangular, and U is upper triangular or upper trapezoidal when A is rectangular, using the earlier symbolic ordering and analysis.

Solve Solves a square sparse linear system $Ax = b$, using the numeric factorization computed by factorization routines.

The UMFPACK package only provides the factor and solve functionalities required. In our implementation of the UMFPACK basis option, we exploit a stable and efficient *block-LU* updating method, proposed in [26] and [42]. This part of our C code is based on a Fortran code LUMOD, originally developed by Saunders [68]. The implementation of this updating routine can be combined with other factor and solve packages (besides UMFPACK) and be used more generally in other linear system operations where such functionalities are required.

2.3.1 Block-LU Update

A sequence of rank-one modifications in the form of column replacement occurs when we perform complementary pivots. For the sake of simplicity, let us express the linear system in a general form:

$$Hy = h, \quad y \in \mathbf{B} \tag{2.3.1}$$

with $\mathbf{B}$ a certain box set. Suppose H_0 is the basis matrix corresponding to a basic feasible solution of the above system (2.3.1), whose LU factorization is computed. At each subsequent pivot a rank-one modification is made to the basis matrix. After a certain number of updates, let V_k contain the columns from H that have newly become basic since the factorization of H_0; let U_k contain unit vectors representing the locations of the columns being updated, with k referring to the number of columns in U_k. (Note that k is not necessarily equal to the number of updates performed since the factorization of H_0.) Hence the above matrices have the following dimensions: H_0 is $n \times n$, V_k is $n \times k$, and U_k is $n \times k$. The new basis matrix H_a can be expressed as

$$H_a = H_0 + (V_k - H_0 U_k) U_k^\top,$$

and it is easy to see that the system $H_a\, y = h_a$ is equivalent to

$$\begin{bmatrix} H_0 & V_k \\ U_k^\top & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_a \\ 0 \end{bmatrix} \tag{2.3.2}$$

with the solution $y = y_1 + U_k\, y_2$.

The matrix in (2.3.2) has the following block-triangular factorization:

$$\begin{bmatrix} H_0 & V_k \\ U_k^\top & \end{bmatrix} = \begin{bmatrix} H_0 & \\ U_k^\top & -C_k \end{bmatrix} \begin{bmatrix} \mathbb{I} & Y_k \\ & \mathbb{I} \end{bmatrix},$$

where

$$H_0 Y_k = V_k, \quad C_k = U_k^\top Y_k.$$

The solution to (2.3.2), and hence $H_a\, y = h_a$ may be obtained by

$$H_0 w = h_a,$$

$$C_k y_2 = U_k^\top w,$$

$$y_1 = w - Y_k y_2.$$

All updating information is carried along via the Schur-complement $C_k(= U_k^\top H_0^{-1} V_k)$ and the matrix of transformed columns Y_k. Rather than modifying the factors of H_0, we can now carry out the updates on the factors of a much smaller matrix C_k, which has dimension k independent of the size of the original matrix H_0. The LU factors of H_0 can be used without modification for many iterations.

In our implementation, the maximum size of C_k is set to be 100. As long as the dimension of C_k doesn't exceed this number, no refactorization is performed. (The maximum size is set to be an option so that the user can modify this option to control the refactorization frequency.)

Depending on which columns enter and leave the basis at each pivot, four different types of updates to matrices C_k and Y_k are performed:

Case 1 Add a row ($r^\top$) to $U_k^\top$, and add a column (w) to Y_k, resulting in adding a row and a column to C_k:

$$U_{k+1}^\top = \begin{pmatrix} U_k^\top \\ r^\top \end{pmatrix} \quad \text{and} \quad Y_{k+1} = (Y_k \quad w)$$

$$C_{k+1} = U_{k+1}^\top {H_0}^{-1} V_{k+1} = \begin{pmatrix} C_k & U_k^\top w \\ r^\top Y_k & r^\top w \end{pmatrix}.$$

Case 2 Replace a column of Y_k by w, and replace a column of C_k by $U_k^\top w$. U_k is unchanged.

Case 3 Replace a row of $U_k^\top$ by $r^\top$, and replace a row of C_k by $r^\top Y_k$. Y_k is unchanged.

Case 4 Delete a row from $U_k^\top$, and delete a column from Y_k, resulting in deleting a row and column from C_k.

The updates of C_k are carried out on the LU factors of C_k, in a slightly different form: $L_C C_k = U_C$, where U_C is an upper triangular matrix and L_C is a square matrix. L_C and U_C are stored as dense matrices, since the size of C_k is relatively small. At each update, the method is able to maintain the dense factors of C_k efficiently using sweeps of stabilized elementary transformations [9, 43]. Note that U_k is maintained as a vector rather than in matrix from. In solving $C_k y_2 = U_k^\top w$, only a matrix-vector multiplication with L_C and $U_k^\top w$ and a backward triangular solve with U_C are required. Therefore the updates can be achieved at a much lower cost than performing a full factorization at every pivot. A detailed description of how to store Y_k efficiently with dense and sparse parts can be found in [26].

We denote the initial set of basic variables by $\mathcal{B}_0$ and its complement by $\mathcal{N}_0$, and give a description of how the above update cases to C_k and Y_k correspond to the types of variables entering and leaving the basis.

- If the entering column (c) is from $\mathcal{N}_0$ and the leaving column is from $\mathcal{B}_0$, perform **Case 1** with $r = e_p$ and $w = H_0^{-1}c$, where e_p is a unit vector corresponding to the position of the leaving column from H_0 and w is already available from the last solve in finding the new direction.

- If the entering column (c) is from $\mathcal{N}_0$, and the leaving column is from $\mathcal{N}_0$, perform **Case 2** with the already available $w(= H_0^{-1}c)$.

- If the entering column is from $\mathcal{B}_0$, and the leaving column is from $\mathcal{B}_0$, perform **Case 3** with $r = e_p$.

- If the entering column is from $\mathcal{B}_0$, and the leaving column is from $\mathcal{N}_0$, perform **Case 4** with the corresponding row and column.

After a number of iterations, the scheme performs a factorization of the current basis H_a and redefines it to be H_0.

When the matrix being factorized is singular or ill-conditioned, UMFPACK routines do not provide the same singularity information (such as the number of apparent singular elements and their locations in U) as the LUSOL routines. Therefore the upper triangular matrix U needs to be extracted from the object returned by UMFPACK together with the permutation matrices P and Q. Corresponding routines are supplied to determine the singular row(s) and column(s) of the original matrix. The threshold number used by LUSOL to determine "small" elements in U is adopted for the factors obtained from the UMFPACK routines.

2.3.2 Extensions of Block-LU

The rest of this section is devoted to a description of how to extend the block-LU method to perform extra types of updates, in particular deletion and addition.

As before, we start from an initial basis system H_0 whose LU factors are available. We consider the type of update where a column (q) and a row (p) need to be deleted from H_0 at the same time. Suppose the new reduced system after this deletion is

$$H_a x = h_a, \tag{2.3.3}$$

the computation can be equivalently carried out by solving an augmented system:

$$\begin{bmatrix} H_0 & e_p \\ e_q^\top & 0 \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} \tilde{h}_a \\ 0 \end{bmatrix}.$$

The bottom equation forces the q^{th} element of y to be 0, which corresponds to the removal of column q from H_0. Adding $e_p z$ to H_0 is equivalent to adding an unconstrained scalar z

to the p^{th} equation, which essentially frees up the p^{th} equation. Correspondingly, the p^{th} element in $\tilde{h}_a$ can take any scalar value and the rest of the elements in $\tilde{h}_a$ are the same as those in the original h_a. Therefore the remaining equations are equivalent to (2.3.3). We can carry out a block-triangular factorization of the above augmented system in a similar fashion to the block-LU method discussed before.

Solving a new system with a column (q) and a row (p) added to the previous system H_0 can be achieved by essentially attaching the row and column to the previous system and therefore forming an augmented system naturally. A more extensive explanation of the addition operation will be given in Chapter 4.

We postpone a detailed illustration of the corresponding updates to C_k and Y_k to Chapter 4, where it will be clear how these additional types of updates are induced by our special treatment to the basis systems in that context. We still maintain U_k in vector form, but in this case it contains both positive index number (which denotes the index of an actual row from the original system) and negative index number (which marks the position of a replaced or deleted column). Since the augmented rows may be actual rows from the original system, the lower right-hand-side of the augmented system (2.3.2) may contain nonzero element(s).

In summary, constructing special augmented matrices and extending the block-LU method enable us to perform operations such as deletion and addition of a row and a column at the same time.

2.4 COIN

The COIN-OR utilities [59] are a collection of open source utilities written in C++, which support COIN-LP and many other optimization projects in the COIN-OR repository.

Factor, solve, and update are contained in a set of "CoinFactorization" routines, based on a Markowitz factorization and a Forrest-Tomlin update [40].

Factor For a sparse matrix A given as triplets, computes a factorization by exploiting the "CoinFactorization::factorize" routines in the COIN-OR utilities. The factor process starts from a sparse factor routine. Conditioning on the number of elements in the selected pivotal row and column, the factor process continues in one of four different sparse subroutines. At every pivot iteration, a check is performed in order to determine whether to switch to a dense factor routine provided by either a Fortran code from LAPACK (preferred) or the COIN-OR utilities.

Solve For a given vector b, uses the existing LU factors of A to solve $Ax = b$ with the "CoinFactorization::updateColumnFT" routine of the COIN-OR utilities. The resulting solution vector is permuted by the permutation matrix from the above factor routines.

Update Exploits the "CoinFactorization::replaceColumn" routine in the COIN-OR utilities to obtain a new factorization when matrix A is updated.

The COIN-OR utilities alone are able to provide the factor, solve and update functionalities required by the basis package. To improve the accuracy and stability of COIN, we adapted the linear refinement and scaling used by COIN-LP routines. These extra procedures however may cost more solves and factorizations.

If matrix A is singular or ill-conditioned, the factorization routine will indicate the detection of singularity on exit. The columns that are left unpivoted will be marked by -1 in the permutation matrix (in vector form) returned. The $Basis\_NumSingular()$ and $Basis\_GetSingular()$ routines are modified to exploit this information and determine the corresponding singular row(s) and column(s) of the matrix.

2.5 Dense

Besides the three basis packages designed for sparse systems, another option is supported by a set of dense factorization utilities. The dense routines exploit the updating procedure proposed by Fletcher and Matthews [39] and provide the following major functionalities.

Factor For a given dense square matrix $A_{n\times n}$, computes a factorization $PAQ = LU$ by Gaussian elimination with a complete pivoting strategy. The permutation matrices P and Q are chosen so that L is unit lower triangular and U is upper triangular.

Solve For a given vector b_n, solves the two triangular systems with the L, U matrices using forward and backward substitutions to find a vector x_n that solves the linear system $Ax = b$.

Update Updates L and U in a stable manner to obtain a new factorization when a rank-one update is performed on A.

Despite the simplicity of the dense routines, we do not recommend it in our applications because we generally deal with large sparse systems, especially within the complementary pivoting process. In the dense case, only the singularity message is returned, but no actual $Basis\_NumSingular()$ routine is supplied.

2.6 Summary

This chapter discussed the linear algebra operations, in particular, factor, solve and rank-one update associated with the basis system. They are first coded in a basis package in the PATH solver [34] and the efficiency of the basis package has a significant impact on the efficiency of both of the solvers we consider in this thesis. This will be discussed further in the next chapter.

Descriptions of four interchangeable basis package options were given, amongst which LUSOL is the original option set as the default in PATH. The motivations for considering other new basis options, namely UMFPACK and COIN, and their performance comparisons will be given in the next chapter. More emphasis was given to the block-LU updates, since the sparse linear factorization package UMFPACK alone does not provide update routines and the extension of this method is essential to one of the basis options employed by the new affine variational inequality solver. The extension scheme is suggested by Saunders and also appears in the thesis of Huynh [50]. The use of this method associated with the special basis structure of the AVI system is new to this thesis.

Chapter 3

Linear Algebra Enhancements to PATH

The PATH algorithm processes mixed complementarity problems by solving their corresponding normal map equations using a nonsmooth Newton method [51, 52, 53]. It then performs Lemke's method [57, 58] to solve the resulting linear complementarity problem after each linearization. More details of the PATH algorithm are provided in Section 3.1. As discussed in Chapter 2, the pivotal scheme requires linear system routines (factor, solve and update), which are contained in a basis package. LUSOL basis package is the default option currently used by PATH for the factor, solve and update routines. This chapter describes our experiences using two other options: UMFPACK and COIN. Section 3.2 motivates exploring these new basis options by first providing some statistics demonstrating that most of the computational effort in PATH is spent in the basis package. We then show that UMFPACK package is considered as an alternative because it is more effective than LUSOL for factoring and solving certain linear large-scale systems. We also present some successes obtained by the COIN-LP solver to motivate our choice of the COIN-OR utilities as the other candidate for improving PATH. Computational results comparing the basis packages in PATH are given in Section 3.3. Finally, Section 3.4 summarizes our conclusions.

3.1 PATH Algorithm

PATH is a generalized Newton method for solving the normal map equation that is globalized via a nonmonotone search using a smooth merit function. Because the merit function is smooth, a steepest descent direction can be used when the search using the Newton point fails to yield a new iterate satisfying the nonmonotone search criteria. This algorithmic framework is well defined with global convergence and locally fast convergence rates [30, 63]. The implementation contains the following parts:

1. Preprocess the mixed complementarity problem to fix variables, improve bounds, and eliminate redundancy [35].
2. Identify an approximation to the active set at the solution using a crash technique.
3. Linearize the normal map at the current iterate. Solve the linearization by constructing a piecewise linear path between the current iterate and the solution. Then search using the generated path to determine a new iterate satisfying the nonmonotone search criteria.

Details for the crash method, forming and solving the linearizations, and the nonmonotone search criteria are given in the following sections.

3.1.1 Merit Function

When solving a nonlinear system of equations, a search along the Newton direction is performed to find a new iterate that sufficiently decreases the chosen merit function value. In the implementation of Newton's method in PATH, this search uses nonmonotone descent criteria [31, 47, 48] with a watchdog technique [6]. The current default merit function is based on the Fischer-Burmeister function [38], $\phi : \mathbb{R}^2 \to \mathbb{R}$, defined as

$$\phi(a, b) := \sqrt{a^2 + b^2} - a - b.$$

Any zero of the Fischer-Burmeister function is known to satisfy the complementarity conditions:

$$\phi(a, b) = 0 \Leftrightarrow a \geq 0, b \geq 0, ab = 0.$$

With this property, a general mixed complementarity problem in the form of (1.1.1) can be reformulated into a system of equations $\Phi(z) = 0$, with $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ defined componentwise as

$$\Phi_i(z) := \begin{cases} \phi(z_i - l_i, F_i(z)) & \text{if } -\infty < l_i < u_i = +\infty \\ -\phi(u_i - z_i, -F_i(z)) & \text{if } -\infty = l_i < u_i < +\infty \\ \phi(z_i - l_i, \phi(u_i - z_i, -F_i(z))) & \text{if } -\infty < l_i < u_i < +\infty \\ -F_i(z) & \text{if } -\infty = l_i < u_i = +\infty \\ 0 & \text{otherwise} \end{cases}$$

for $i \in \{1, \ldots, n\}$. The merit function is

$$\Psi(z) := \frac{1}{2}\|\Phi(z)\|^2. \tag{3.1.1}$$

The advantage of using Ψ over other classical merit functions such as the norm of the normal map residual is that Ψ is continuously differentiable with gradient $\nabla\Psi(z) = \partial\Phi(z)^T\Phi(z)$. Note that Φ is nondifferentiable in general; hence $\partial\Phi(z)$ is the generalized Jacobian of $\Phi(z)$. Detailed formulas for calculating the gradient of the merit function can be found in [30].

3.1.2 Crash Method

The crash method is a way to compute a good active set. It is based on a projected Newton method. First the active set at the initial point $z^0 = \pi_{\mathbf{B}}(x^0)$ is identified. The corresponding index set of variable z is denoted by $\mathcal{A}$, where

$$\mathcal{A} := \{i \in \{1, \ldots, n\} | \{z_i = l_i, F_i(z) \geq 0\} \text{ or } \{z_i = u_i, F_i(z) \leq 0\}\}.$$

The Newton direction for a reduced system is then computed:

$$(\nabla F_{\mathcal{I}\mathcal{I}}(z^k) + \epsilon \mathbb{I})\, d_{\mathcal{I}} = F_{\mathcal{I}}(z^k), \tag{3.1.2}$$

where $\mathcal{I} = \{1, \ldots, n\} \backslash \mathcal{A}$ and ϵ is a perturbation parameter (see below). We then set $d_{\mathcal{A}} = 0$ and compute an $\alpha \in (\bar{\alpha}, 1]$ such that the new iterate

$$z^k(\alpha) = (1-\alpha) z^k + \alpha \pi_{\mathbf{B}}(z^k - d)$$

with the default line search or

$$z^k(\alpha) = \pi_{\mathbf{B}}((1-\alpha) z^k + \alpha (z^k - d))$$

with an arc search decreases the merit function

$$\Psi(z(\alpha)) \leq \begin{cases} \Psi(z^k) - \sigma \nabla \Psi(z^k)^T (z^k - z^k(\alpha)), & \\ \qquad\qquad\qquad \text{if } \nabla \Psi(z^k)^T (z^k - z^k(\alpha)) < 0 & \\ (1 - \alpha\sigma)\Psi(z^k), \quad \text{otherwise} & \end{cases}$$

where $\bar{\alpha}$ is a constant minimum step size and $\Psi(z)$ defined in (3.1.1) is chosen as the merit function. In solving the reduced system in (3.1.2), the perturbation parameter ϵ is zero, unless the reduced matrix $\nabla F_{\mathcal{I}\mathcal{I}}(z^k)$ is rank deficient, in which case we choose a large enough ϵ such that $\nabla F_{\mathcal{I}\mathcal{I}}(z^k) + \epsilon \mathbb{I}$ is numerically nonsingular. Then ϵ is reduced at subsequent crash steps based on the residual of the merit function. Since the crash method does not guarantee convergence, we terminate the process if any of the following criteria is satisfied: the number of iterations exceeds a maximum value; the step length is too small; not enough changes of the active set have been made consecutively for several iterations; or the perturbation scheme is not successful. If the crash iterates manage to converge to a small enough merit function value, the original MCP is solved solely in the crash process. Some benefits of using the crash technique and its convergence properties can be found in [23].

3.1.3 Linearization

Newton's method for smooth functions linearizes the function at the current iterate and solves a linear system of equations to obtain a direction. The normal map $F_{\mathbf{B}}(x)$ defined in (1.3.1) is nonsmooth, however, because of the projection operator $\pi_{\mathbf{B}}(\cdot)$, and a linearization of $F_{\mathbf{B}}(x)$ is not available. Rather, the following piecewise affine map approximates the function around x^k:

$$L_k(x) := (\nabla F(\pi_{\mathbf{B}}(x^k)) + \epsilon \mathbb{I})(\pi_{\mathbf{B}}(x) - \pi_{\mathbf{B}}(x^k)) + F(\pi_{\mathbf{B}}(x^k)) + x - \pi_{\mathbf{B}}(x), \tag{3.1.3}$$

with a perturbation parameter ϵ similarly defined and updated as in the crash procedure. This approximation is solved by constructing a parametric piecewise linear path $p^k(t)$ for $t \in [0, T^k]$, with $T^k \in (0, 1]$ satisfying

$$p^k(0) = x^k, \tag{3.1.4}$$

$$L_k(p^k(t)) = (1-t)F_{\mathbf{B}}(x^k). \tag{3.1.5}$$

The Newton point x_N^k is defined as $p^k(T^k)$, which solves the linearization if $T^k = 1$. These conditions ensure that the path starts at the current point x^k and the norm of the approximation function decreases at least linearly in $1 - t$ on the path. Note that $p(t)$ may not be single valued because the path can make turns.

Instead of constructing the path directly using (3.1.4)–(3.1.5), a pivotal technique is used to solve an equivalent linear mixed complementarity problem as follows. Let $z(t) = \pi_{\mathbf{B}}(x(t))$, $v(t) = (x(t) - z(t))_+$ and $w(t) = (z(t) - x(t))_+$. Then, $p^k(t)$, can be expressed as

$$p^k(t) = z(t) - w(t) + v(t) \quad \forall\, t \in [0, T^k].$$

Using the transformation above, together with the definition of the piecewise affine map in (3.1.3), we can express (3.1.5) as

$$Mz(t) + q - w(t) + v(t) = (1-t)r,$$

where $M = \nabla F(\pi_{\mathbf{B}}(x^k)) + \epsilon\mathbb{I}$, $q = F(\pi_{\mathbf{B}}(x^k)) - \nabla F(\pi_{\mathbf{B}}(x^k))\pi_{\mathbf{B}}(x^k)$, and $r = F_{\mathbf{B}}(x^k)$. In the actual implementation, we scale the covering vector r in the above equation by a scalar s. We also augment the system by incorporating a vector of artificial variables (a) to help construct an invertible basis under possible rank deficiency. In particular, the linear complementarity problem becomes

$$\begin{aligned} Mz(t) + q - w(t) + v(t) + a - \tfrac{(1-t)(sr)}{s} &= 0 \\ 0 \le w \perp z &\ge l \\ 0 \le v \perp z &\le u \\ z \in \mathbf{B}, w \ge 0, v \ge 0, a \equiv 0, t &\in [0,1]. \end{aligned} \tag{3.1.6}$$

A guess of the initial basis of the above system follows the active set approximation resulting from the crash process. We rely on the factorization routines to detect possible rank deficiency and identify linearly dependent rows and columns in the initial basis. The basis is then defined appropriately based on the singularity information, so that the system (3.1.6) has an invertible basis at the starting point. A pivotal technique similar to Lemke's method with specific entering and leaving pivotal rules can then be used to construct the path. Each pivot leads to a new piece on the path. If, in the end, the pivots terminate with t leaving the basis at 1, the linear complementarity problem is solved successfully, and the Newton point $x_N^k = p^k(1)$ is generated from $(z(1), w(1), v(1))$. When ray termination or cycling occurs, the path generation will terminate at a point $p^k(T^k)$ with $T^k < 1$. More details on constructing an invertible basis and pivotal rules can be found in [22] and [34].

3.1.4 Nonmonotone Search

The nonmonotone descent scheme implemented in the PATH algorithm distinguishes among *m-steps*, *d-steps*, *watchdog steps* [6], and *projected gradient steps.*

The merit function value at the Newton point $z^k(T_k)$ is checked by a nonmonotone descent criteria during m-steps. In particular, given a *reference value* $\mathcal{R}$, the point is acceptable if

$$\Psi(z^k(T_k)) \leq \begin{cases} \mathcal{R} - \sigma \nabla \Psi(z^k(0))^T (z^k(0) - z^k(T_k)), & \\ \qquad\qquad\qquad \text{if } \nabla \Psi(z^k(0))^T (z^k(t) - z^k(T_k)) < 0 & \\ (1-\sigma)\mathcal{R}, \quad \text{otherwise} & \end{cases} \tag{3.1.7}$$

The reference value is decreased as the algorithm proceeds. If the Newton point satisfies this criteria, we save it as a *check point* for use with the watchdog strategy. Every time the check point is updated, the corresponding Newton point $x^k(T_k)$ comprising $(z^k(T_k), w^k(T_k), v^k(T_k))$ and the Newton point obtained at the next iteration ($z^{k+1}(T_{k+1})$ and $x^{k+1}(T_{k+1})$) are saved so that regeneration of the path will not be necessary if we have to go back to this check point.

A d-step is acceptable if the Newton point $z^k(T_k)$ is close enough to the current point $z^k(0)$. In particular, the point is accepted if

$$||z^k(T_k) - z^k(0)|| \leq \Delta$$

and the merit function does not become too large, where Δ is initialized to a preset value and is decreased as the algorithm progresses. If, at the same time, the nonmonotone descent criterion is satisfied, the current point is saved as a check point. If the d-step conditions are not satisfied, the nonmonotone criterion is checked. Moreover, after every $\bar{n}$ d-steps, the nonmonotone descent criterion is checked.

If the nonmonotone descent criterion is violated in an m-step or if the merit function value would increase too much over the current reference value when accepting a d-step, a *watchdog* step is taken. The watchdog step retrieves the most recent check point saved consisting of the four points $z^{\tilde{k}}(0)$, $x^{\tilde{k}}(0)$, $z^{\tilde{k}}(T_{\tilde{k}})$, and $x^{\tilde{k}}(T_{\tilde{k}})$ for some $\tilde{k}$. With these points, a search is performed to find a new point satisfying the nonmonotone descent criterion. The user can select to search along the line segment connecting the two projected points

$$z^{\tilde{k}}(\alpha) = (1-\alpha)z^{\tilde{k}}(0) + \alpha z^{\tilde{k}}(T_{\tilde{k}}) \tag{3.1.8}$$

or the projected arc connecting the two Newton points

$$z^{\tilde{k}}(\alpha) = \pi_{\mathbf{B}}((1-\alpha)x^{\tilde{k}}(0) + \alpha x^{\tilde{k}}(T_{\tilde{k}})). \tag{3.1.9}$$

In either case, we find a step length $\alpha \in (\bar{\alpha}, 1)$ iteratively such that

$$\Psi(z^{\tilde{k}}(\alpha)) \leq \begin{cases} \mathcal{R} - \sigma \nabla\Psi(z^{\tilde{k}}(0))^T(z^{\tilde{k}}(0) - z^{\tilde{k}}(\alpha)), \\ \qquad\qquad\qquad \text{if } \nabla\Psi(z^{\tilde{k}}(0))^T(z^{\tilde{k}}(0) - z^{\tilde{k}}(\alpha)) < 0 \\ (1-\alpha\sigma)\mathcal{R}, \quad \text{otherwise,} \end{cases} \tag{3.1.10}$$

where $\bar{\alpha}$ is a constant minimum step size and $\mathcal{R}$ is the current reference value. The default search type in the PATH solver is a line search.

If the step size in the search becomes too small without finding a point satisfying the nonmonotone descent criterion, a monotone projected gradient step with our smooth merit function Ψ is taken as follows. We first move back to the *best point*, which is the check point with the smallest merit function value, say $\bar{k}$. Then a step length $\alpha \in (\bar{\alpha}, 1)$ is determined such that

$$\Psi(z^{\bar{k}}(\alpha)) \leq \begin{cases} \Psi(z^{\bar{k}}(0)) - \sigma \nabla \Psi(z^{\bar{k}}(0))^T (z^{\bar{k}}(0) - z^{\bar{k}}(\alpha)) \\ \qquad\qquad\qquad\qquad \text{if } \nabla \Psi(z^{\bar{k}}(0))^T (z^{\bar{k}}(0) - z^{\bar{k}}(\alpha)) < 0 \\ (1 - \alpha\sigma)\Psi(z^{\bar{k}}(0)), \quad \text{otherwise,} \end{cases} \tag{3.1.11}$$

with

$$z^{\bar{k}}(\alpha) = (1-\alpha) z^{\bar{k}}(0) + \alpha \pi_{\mathbf{B}}(z^{\bar{k}}(0) - \nabla \Psi(z^{\bar{k}}))$$

when using a line search and

$$z^{\bar{k}}(\alpha) = \pi_{\mathbf{B}}((1-\alpha) z^{\bar{k}} + \alpha(z^{\bar{k}} - \nabla \Psi(z^{\bar{k}})))$$

when using an arc search. The default gradient search type in PATH is an arc search.

3.1.5 Summary

A summary of the main PATH algorithm is as follows:

PATH CODE

Step 1 Initialization: let $z^0, \bar{n} \geq 1, \triangle = \bar{\triangle} > 0, \beta \in (0,1)$ be given:

set $k = 0$, $\mathit{check\_point} = 0, \mathit{best\_point} = 0; j = 0, b = 0, \triangle_0 = \triangle, \mathcal{R}_0 = \Psi(z^0)$.

Step 2 If $\Psi(z^k) = 0$, stop.

Step 3 Update the perturbation parameter ϵ. Using the linearization approximation L_k, apply the transformation and generate a path from z^k to the solution of the linear complementarity subproblem: $[0, T_k] \mapsto \mathbf{B}, T_k \in (0,1]$ satisfying (3.1.6).

```
**Step 4** If ($k < check\_point + \bar{n}$) then
    d-step:
    if ($||z^k(T_k) - z^k(0)|| < \triangle$), the step is small enough; accept it:
      set $z^{k+1}(0) = z^k(T_k)$;
      set $\triangle = \triangle * \beta$;
      if the nonmonotone descent criterion in (3.1.7) is satisfied,
        update $check\_point$;
        increment k and go to Step 2.
      if $\Psi(z^k) > LargeConstant * \mathcal{R}_j$,
        perform a watchdog step;
    else, the step is too big; perform an m-step.
    else
    m-step:
    if the monotone descent criterion in (3.1.7) is satisfied with the reference value $\mathcal{R}_j$,
      accept the step:
      set $z^{k+1}(0) := z^k(T_k)$;
    else perform a watchdog step:
      set $k = check\_point, \triangle = \triangle_j$;
      if $\alpha \in (\bar{\alpha}, 1)$ can be found satisfying the condition in (3.1.10) with
      reference value $\mathcal{R}_j$, by conducting a line or arc search in (3.1.8)-(3.1.9),
        set $z^{k+1}(0) := z^k(\alpha)$;
      else perform a projected gradient step:
        set $k = best\_point, \triangle = \triangle_b$;
        find $\alpha \in (\bar{\alpha}, 1)$ satisfying (3.1.11);
    update check_point:
      increment $j$; update $\mathcal{R}_j$; set $\triangle_j = \triangle$;
      set check_point $= k + 1$;
      update best_point if $\Psi_{check\_point} < \Psi_{best\_point}$, by cloning the check_point
      info to the best_point;
```

Step 5 Increment k, and go to *Step 2*.

At the beginning of *Step 3*, with $z^k \in \mathbf{B}$ given, the initial values for w^k and v^k need to be supplied. In other words, we need to calculate a corresponding x^k, whose projection is z^k and which has the best normal map residual. We do so by solving the following optimization problem (omitting the superscripts):

$$\begin{aligned} & \min_{w,v} \|x - z + F(z)\| \\ \text{s.t.} \quad & x = z - w + v \\ & \pi_{\mathbf{B}}(x) = z \\ & w \geq 0, v \geq 0, w^T v = 0. \end{aligned}$$

In practice, this problem is solved simply as

$$\begin{cases} \text{if } z_i \leq l_i \text{ and } f_i > 0 & x_i = l_i - f_i, w_i = f_i, v_i = 0 \\ \text{else if } z_i \geq u_i \text{ and } f_i < 0 & x_i = u_i - f_i, w_i = 0, v_i = -f_i \\ \text{else} & x_i = z_i, w_i = v_i = 0 \end{cases}$$

for $i \in \{1, \ldots, n\}$.

Some "safeguard" steps are omitted from the algorithm summary. For example, to determine whether we should perform a watchdog step directly, we always check first if the normal map function $F_{\mathbf{B}}$ is defined at the newly generated point z^k.

3.2 UMFPACK and COIN-OR Utilities

The key to obtaining efficiency in the PATH algorithm depends on solving a series of subproblems in the form of linear mixed complementarity problems (3.1.6) using an adaptation of Lemke's method. We call each iteration of PATH a *major* iteration and each pivotal step in solving the linear MCP a *minor* iteration. At each major iteration, factorization of the current basis matrix, which corresponds to the active set at the current point, together with rank-one updates (corresponding to the pivotal steps) is required. In the crash procedure described before, a Newton system as in (3.1.2) needs to be solved at each iteration, which also requires routines capable of factoring and solving linear systems.

A test run on the MCPLIB problems with PATH/LUSOL suggests that on average, 74% of the total solving time is spent on factoring, solving, and updating linear systems. Moreover, as we can see below, using a dynamic game (*dyngame*), the proportion of time spent on these routines (contained in the basis package) increases significantly as the size of the system grows. The dynamic game also provides better means for comparing the performance of the LUSOL basis option with one of its alternatives – the UMFPACK option.

Dynamic games are mathematical models of the interaction between independent agents controlling a dynamical system. Such situations occur in military conflicts, economic competition, and parlor games such as chess or bridge. The actions of the agents (also called players) influence the evolution over time of the state of the system. The difficulty in deciding what should be the behavior of these agents stems from the fact that each action an agent takes at a given time will influence the reaction of the opponent(s) at later times. The specific model considered here is a game played on a grid based on the model of dynamic competition in an oligopolistic industry [27, 61]. This model has been used extensively in applications such as advertising, collusion, mergers, technology adoption, international trade, and finance and has become a central tool in analysis of strategic interactions among forward-looking players in dynamic environments.

Here we present the formulation of this particular dynamic game (dyngame) as a complementarity problem. Two firms are considered as players in a dynamic Cournot competition with learning and investment. During a period of time, each firm can be in one of S states, $s = 1, 2, \ldots, S$, therefore forming a grid. In each period, the two firms engage in quantity (output) competition. In particular, let q_i denote firm i's production quantity for $i = 1, 2$. The total output is

$$Q = q_1 + q_2.$$

The price for a given total quantity is defined by a linear inverse demand function

$$P(Q) = A - Q,$$

where A is a scalar. Firm i's production cost function is a quadratic function in this model:

$$CP_i(q) = a_1 q_1 + q_1^2.$$

During each period, the firms choose q to maximize their profits $f(q_1, q_2)$ defined as:

$$f_i(q_1, q_2) = q_i P(q_1 + q_2) - \theta_i(a_1 q_i + q_i^2), \tag{3.2.1}$$

where parameter θ_i denotes firm i's efficiency level which is computed based on the state the firm is in.

The state of the game is the firms' production costs. In the next period, each firm is able to transit from the current state s to state $s-1, s+1$ or remain in s. The dynamic setting obeys the "law of motion", that is the state follows a controlled discrete-time, finite-state, fist-order Markov process with transition probability $Pr(s'|u, s)$, where the i^{th} component of s' is $s_i - 1, s_i + 1$ or s_i for firm i in the next period and $u(= (u_1, u_2))$ denotes the actions of the firms. In our example, we consider a special case with independent transitions:

$$Pr(s'|u, s) = Pr_1(s'_1|u_1, s_1)Pr_2(s'_2|u_2, s_2),$$

with the action of firm i (u_i) being its investment effort. Moreover the transition probability is a combination of the individual probabilities associated with the following three forces:

Probability of Successful Learning : Current output may lead to lower production cost due to successful learning.

Probability of Successful Investment : Firms can make investment expenditures to reduce cost when successful.

Depreciation : Shock to efficiency may increase cost.

Suppose the investment expenditure is also quadratic:

$$CI_i(u_i) = b_i u_i + d_i u_i^2 \tag{3.2.2}$$

for $i = 1, 2$. Let $V_i(s_1, s_2)$ be the expected net present value (NPV) of firm i when the firms are in the current states (s_1, s_2) respectively. Due to Bellmann equation [27, 61], V_i can be written as:

$$V_i(s_1, s_2) = \max_{u_i} \left\{ f_i - CI_i + \beta E_{s_1', s_2'} \{ V_i(s_1', s_2') |\, u_1, u_2, s_1, s_2 \} \right\}.$$

The first two components to the right-hand-side of the above equation are the profit functions (3.2.1) and the investment costs (3.2.2) respectively. The last term is the expected future costs and revenues in the next period discounted at a common rate $\beta < 1$. The expectation can be computed by

$$E_{s_1', s_2'} \{ V_i(s_1', s_2') |\, u_1, u_2, s_1, s_2 \} = \Sigma_{s_1', s_2'} \, Pr_1(s_1' | u_1, s_1) \, Pr_2(s_2' | u_2, s_2) \, V_i(s_1', s_2'),$$

where s_i' can take values at $s_i - 1, s_i + 1$ and s_i.

The optimality conditions for this dynamic game can be written as

$$V_i(s_1, s_2) = f_i - CI_i + \beta E_{s_1', s_2'} \{ V_i(s_1', s_2') |\, u_1, u_2, s_1, s_2 \} \quad \perp \quad V_i(s_1, s_2) \text{ free} \tag{3.2.3}$$

$$0 \le -\nabla_{u_i} \left\{ -CI_i + \beta E_{s_1', s_2'} \{ V_i(s_1', s_2') |\, u_1, u_2, s_1, s_2 \} \right\} \quad \perp \quad u_i \ge 0 \tag{3.2.4}$$

$$0 \le -\nabla_{q_i} f_i \quad \perp \quad q_i \ge 0 \tag{3.2.5}$$

for $i = 1, 2$ and $s_i = 1, \ldots S$, that is a total of $6S^2$ complementarity equations. In (3.2.4), f is dropped when computing the derivative with respect to u, since it does not contain u variables. The complementarity system in (3.2.5) denotes the optimality conditions for maximizing the profits (3.2.1) over the output quantities.

Figure 1 is the nonzero structure of the initial basis matrix of the *dyngame* problem. The size of the matrix is 1600×1600, with $16,656$ nonzero elements. Hence the basis matrix is rather sparse, with density equal to 0.65%.

Figures 2 and 3 are the fill-in of the summations of the lower and upper triangular matrices obtained from the LU factorization computed by the LUSOL factor routines

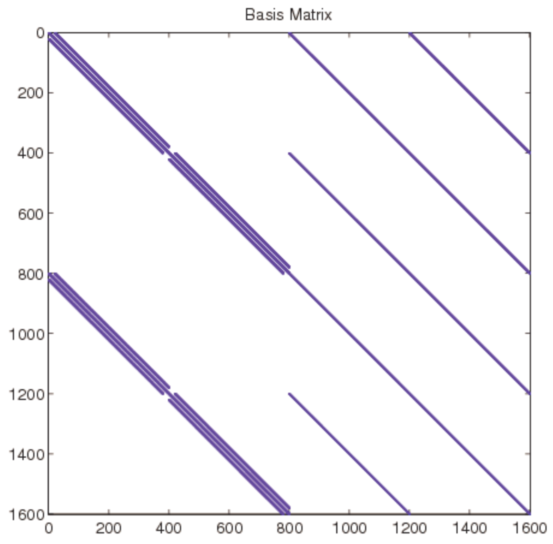
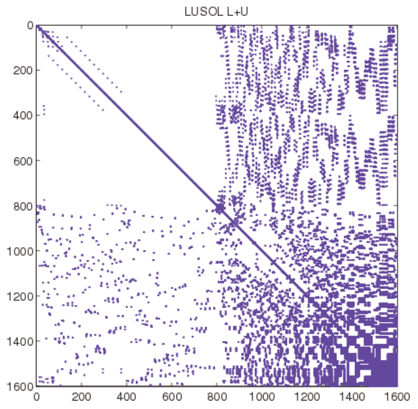


Figure 1: Initial basis matrix of *dyngame* for $\gamma = 1$ and $N = 20$



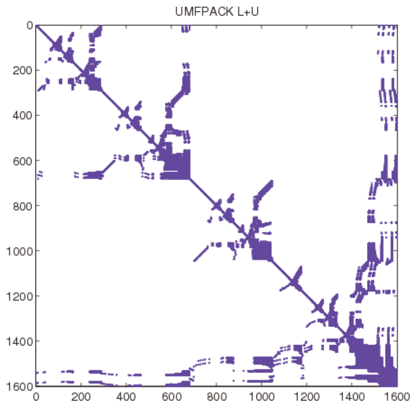


Figure 2: $L + U$ obtained from the LU factorization by the LUSOL routine

Figure 3: $L + U$ obtained from the LU factorization by the UMFPACK routine

and UMFPACK routines, respectively. LUSOL computes lower and upper triangular matrices L and U which satisfy $LU = PBQ$, where B is the original matrix and P, Q are the permutation matrices. The L and U matrices obtained from the UMFPACK routines satisfy $LU = P\tilde{B}Q$, where P, Q are the permutation matrices and $\tilde{B}$ is the original matrix after certain row scaling. The number of nonzero elements is $68,203$ in Figure 2 and $66,684$ in Figure 3. Hence the densities of the matrices are 2.66% and 2.6% in Figure 2 and Figure 3 respectively. The densities of the LU matrices resulting from the two factorization routines are also similar when the size of the problem increases, as seen in Table 1, where *Dim* is the dimension of the basis matrix and *Nnz* is the number of nonzero elements in $L + U$.

Table 1: Density of the $L + U$ Matrices from Factorization

Problem		LUSOL		UMFPACK	
N	Dim	Nnz	Density	Nnz	Density
20	1600×1600	68,171	2.66%	66,684	2.60%
50	$10,000 \times 10,000$	587,112	0.59%	658,755	0.66%
100	$40,000 \times 40,000$	2,773,928	0.17%	2,778,235	0.17%

Table 1 shows that the densities of the resulting $L + U$ matrices do not increase as the size of the problem grows. However, the time taken to perform the basis package functionalities when using LUSOL grows significantly with the increase of the problem size, as seen in Table 2 (*Basis* is the time spent in the basis package routines; *Total* is the total CPU time taken in the whole PATH code; *Pct* is the percentage of the total time spent in the basis package), whereas a rather moderate growth in the time spent in the basis routines is observed when using the UMFPACK package. Note that the dyngame example is solved in the crash procedure alone, therefore only factor and solve operations are involved. Table 2 again indicates that the major computational issue in

PATH lies in performing the basis routines efficiently, since most of the time is spent in the basis routines. One might postulate that the dramatic time increase in the LUSOL case is due to more irregular data access, but we have not demonstrated this rigorously at this stage. Clearly, however, if we can carry out the basis routines efficiently, we are potentially able to substantially improve the efficiency of the PATH algorithm.

Table 2: Time Spent in Basis Routines vs Total Time Taken in PATH

Problem	LUSOL			UMFPACK		
N	Basis	Total	Pct	Basis	Total	Pct
20	0.13	0.18	72.2%	0.09	0.14	64.3%
50	2.66	2.99	89.0%	0.86	1.18	72.9%
100	16.55	17.92	92.4%	4.41	5.78	76.3%

COIN-OR is a collection of open source routines for solving linear, mixed integer, nonlinear, and mixed integer nonlinear programming problems. It also contains basis factorization utilities designed to support many of the projects in the COIN-OR repository, including the COIN-LP solver. The results of solving several moderate-sized linear programs (LPs) using different solvers are presented in Table 3. The size of each model is given by its number of rows, columns, and number of nonzeros (*Row, Col* and *Nnz*). The CPU-seconds (*Time*) spent in solving these LPs using various LP solvers are presented under each solver name for comparison. The examples in Table 3 show that the COIN-LP solver is comparable to the CPLEX solver, which is widely considered to be the best LP solver. COIN-LP is also shown to be much more efficient than the MINOS and SNOPT solvers, whose basis routines are supported by the LUSOL package. Since COIN-LP depends on the COIN-OR utilities to process its linear systems, these results lead to the assumption that the COIN-OR utilities may provide a more efficient basis package than LUSOL. This assumption may fail, because the COIN-LP solver may rely on its pivotal scheme instead of linear system routines to achieve the efficiency exhibited here.

Table 3: LP Examples

Problem				Time			
Name	Row	Col	Nnz	CPLEX	COIN-LP	MINOS	SNOPT
storm	14388	34115	114974	0.61	1.25	15.19	16.45
pds-06	9882	28656	82270	0.24	0.34	18.68	19.29
pds-10	16559	48765	140064	0.56	0.71	155.94	126.14

3.3 Computational Results

The results obtained from using the LUSOL, UMFPACK, and COIN basis packages on the *dyngame* problem are presented in Table 4. The problem size containing the dimension of the basis matrix (*Dim*) and the number of nonzero elements (*nnz*) is listed in the first column of Table 4. We then report the time spent in the basis package (*Basis*) corresponding to different data parameter γ, and the final residuals (*Residual*) when using the LUSOL, UMFPACK and COIN options.

The CPU time spent in the basis package in Table 4 shows clearly that the UMFPACK basis package is significantly more efficient than LUSOL. The final residuals suggest that the UMFPACK basis package leads to increased accuracy. Arguably, the UMFPACK basis package also improves the reliability of the MCP solutions, not only because of the reduced residuals, but also because for $\gamma = 3$ with $N = 200$ and $N = 300$ the time spent on LUSOL is significantly longer than for $\gamma = 2$, while experiences from the rest of the smaller problems indicate that the case with $\gamma = 3$ should take a similar amount of time to solve as with $\gamma = 2$, which is precisely the situation with UMFPACK.

Table 4: Results of the *dyngame* Problem

Problem Size	γ	LUSOL		UMFPACK		COIN	
		Basis	Residual	Basis	Residual	Basis	Residual
N = 20	1	0.13	2.8×10^{-9}	0.09	2.0×10^{-9}	0.17	2.0×10^{-9}
Dim = 2,400	2	0.14	3.6×10^{-9}	0.09	1.6×10^{-9}	0.17	1.6×10^{-9}
Nnz = 19,856	3	0.07	3.3×10^{-9}	0.08	9.7×10^{-10}	0.08	9.8×10^{-10}
N = 50	1	2.66	2.0×10^{-8}	0.86	2.0×10^{-9}	4.80	2.0×10^{-9}
Dim = 15,000	2	0.93	1.3×10^{-8}	0.55	1.6×10^{-9}	1.32	2.5×10^{-9}
Nnz = 127,616	3	0.86	4.8×10^{-9}	0.49	1.0×10^{-9}	1.34	1.0×10^{-9}
N = 100	1	16.55	6.9×10^{-7}	4.42	2.1×10^{-9}	$*$	$*$
Dim = 60,000	2	6.07	5.4×10^{-7}	2.03	1.7×10^{-9}	$*$	$*$
Nnz = 5,152,216	3	9.28	6.0×10^{-9}	1.87	1.1×10^{-9}	18.61	1.1×10^{-9}
N = 200	1	93.52	3.7×10^{-7}	23.48	2.4×10^{-9}	-	-
Dim = 240,000	2	29.07	7.8×10^{-8}	8.72	1.9×10^{-9}	-	-
Nnz = 2,070,416	3	92.88	1.5×10^{-7}	8.45	1.4×10^{-9}	135.8	1.4×10^{-9}
N = 300	1	258.8	2.8×10^{-7}	63.01	2.9×10^{-9}	-	-
Dim = 540,000	2	69.41	3.0×10^{-7}	22.13	2.2×10^{-9}	-	-
Nnz = 4,665,616	3	329.8	5.5×10^{-7}	21.72	5.3×10^{-7}	352.7	5.3×10^{-7}

The COIN basis package, on the other hand, is successful only in solving a subset of the *d*yngame problems. For the instances marked with "$*$", COIN encounters accuracy issues when solving the linear systems. For example, in solving the problem with $\gamma = 1$ and $N = 100$, the inf-norm of the residual from computing the first crash iteration is checked. The residual obtained from using the COIN basis package is as big as 5.91×10^2, whereas LUSOL solves the system with a residual equal to 1.88×10^{-9}, and UMFPACK solves the system with a residual equal to 2.27×10^{-13}. We report this error but do not want to change COIN-OR source, so we can use the updated versions of COIN-OR utilities as they are available. The problems marked with "$-$" are too big for COIN to factor. On the successfully solved instances, COIN takes much more time than both LUSOL and UMFPACK, especially as the problem size grows. The number of major, minor, and crash iterations taken to solve each problem (successfully) is identical for all three basis packages.

Except the option setting for choosing among different basis packages, the default settings have been used for all the problems in Table 4 but one. For the last problem with $N = 300$ and $\gamma = 3$, the search type in the crash scheme is set to be *arc* search, since the default line search with both basis packages takes enormous amount of time to solve.

As expected, it is important to exploit sparse factoring and solving routines on the *dyngame* problem rather than the dense basis routines. This is clearly seen from the results in Table 5, obtained by solving *dyngame* with the dense basis option. The

Table 5: CPU Time Spent in the Basis Package for Solving *dyngame* Using Dense Option

Problem Size	γ	Basis Time	Residual
N = 20	1	61.37	2.0×10^{-9}
Col/rows = 2,400	2	60.51	1.6×10^{-9}
Nnz = 19,856	3	51.63	9.7×10^{-10}

time taken by the dense option is hundreds of times slower than the sparse options (UMFPACK and LUSOL). It takes more than 2400 CPU-seconds to solve the first crash step of *dyngame* with $N = 50$; hence the time for solving larger size problems is not shown.

The new basis packages are also tested on the MCPLIB [21] problems (1005 MCPs). The models which all basis packages succeed or fail to solve are not reported here. Table 6 only lists the models whose solution status is different for different basis packages. The total number of solves with different starting points for each problem is listed in the second column. Under each basis package name, the number of failures is shown. Comparatively, LUSOL has the most failures in solving these relatively difficult problems. UMFPACK does slightly better than LUSOL. COIN has the fewest failures among the three options. The summation of CPU times (basis and total), together with the total

Table 6: Comparative Results on MCPLIB Models

Model	No. of Solves	LUSOL	UMFPACK	COIN
cgereg	22	1	1	0
duopoly	1	0	1	1
fixedpt	2	0	2	0
fried8	5	2	0	0
jiangqi	3	1	0	0
kyh	2	0	0	1
kyh-scale	2	1	1	0
pgvon105	6	1	0	0
pgvon106	6	1	1	1
tiebout1	2	0	1	0
tinsmall	64	0	0	1
venables	2	2	0	0
fails count		9	7	4

of the major and minor iteration numbers for solving this MCP test set is presented in Table 7. In order to make the computation time comparable, only the problems that are solved successfully by all the options are considered (972 MCPs). We set the LUSOL statistics to be the base value (i.e., letting them be 100%) and compute the relative ratios of the other options' statistics to the base value. As we can see in Table 7, LUSOL outperforms both UMFPACK and COIN slightly in time. However, since the total time in solving these problems is under 2 minutes for all the basis options, both UMFPACK and COIN are still comparable in time to LUSOL in dealing with these relatively smaller-scale problems. Note that as we improve the numerical stability of COIN by incorporating linear refinement and scaling process, COIN does achieve more successes as shown in Table tab:diff but takes more solving time.

Table 8 contains all of the problems in MCPLIB whose sizes can be increased. The

Table 7: Sum of Iterations and Time Spent on MCPLIB Problems

Basis Package	Major (ratio%)	Minor (ratio%)	Basis (ratio%)	Total (ratio%)
LUSOL	10110 (100)	188720 (100)	93.64 (100)	119.59 (100)
UMFPACK	10448 (103)	224354 (119)	102.46 (109)	135.97 (114)
COIN	11434 (113)	180530 (96)	107.69 (115)	133.03 (111)

CPU-seconds spent in the basis package using UMFPACK, LUSOL, and COIN in solving each instance are listed in Table 8. For most of these problems (*bai_haung, bratu, dirkse2, dongbai, obstacle*, and *o*pt_cont), as the problem size increases, the advantage of UMFPACK in reducing the solving time over LUSOL becomes more significant. For the other problem (*bert_oc*), time reduction using the UMFPACK basis package is not observed. Nevertheless, the UMFPACK basis package can be used as an alternative to LUSOL without much loss of efficiency on this problem. The COIN option in general takes more time than LUSOL except for the *dirkse2* and *dongbai* problems. The COIN basis time marked by $*$ suggests that COIN may have accuracy issues because it takes different PATH steps from LUSOL and UMFPACK. COIN cannot process the problems marked with "$-$" because the problems sizes are too big. For the largest instance of the *dongbai* problem, UMFPACK essentially takes 19 Newton steps to solve (time marked with "$\diamond$"), whereas LUSOL and COIN both take 20, which may suggest that UMFPACK solves this system with better accuracy than LUSOL and COIN.

A closer examination of the distribution of the basis time taken by the UMFPACK and LUSOL options is given in Table 9, in which we randomly generate three sets of LCPs and total the basis time for each set. The two options take a similar number of iterations to solve each LCP in the test sets. (The COIN option is not compared here because, in general, it performs more solves and factors than the other options.) Problem Set I is comprised of 50 LCPs of relatively smaller-scale (average number of

Table 8: Comparative Results on Enlarged MCPLIB Problems

Test MCP			Basis Time		
Problem Name	Size	Density %	LUSOL	UMFPACK	COIN
bai_haung	4,900	0.10	0.10	0.04	0.18
	19,600	0.03	1.25	0.21	15.70
	78,400	0.01	9.16	1.06	552.07
	313,600	0.00	103.47	6.12	-
bert_oc	5,000	0.05	0.02	0.05	0.02
	50,000	0.01	0.29	0.56	0.44
	500,000	0.00	5.56	8.47	26.91
bratu	5,625	0.09	0.95	0.54	2.33
	22,500	0.02	16.60	5.29	97.54
	90,000	0.01	278.58	49.71	5918.56*
dirkse2	64,001	0.00	29.45	17.90	26.52
	216,001	0.00	395.49	261.64	363.42
	512,001	0.00	2287.89	1461.80	2181.53
dongbai	7500	0.08	4.64	0.60	2.21
	14,700	0.04	51.97	2.59	43.89*
	30,000	0.02	658.19	10.42$^\diamond$	255.01
opt_cont	288	5.59	0.002	0.007	0.005
	8,192	0.21	0.12	0.17	0.15
	32,032	0.05	0.57	0.76	0.82
	160,032	0.01	5.17	4.95	9.27
	320,032	0.01	17.32	12.52	30.66
	480,032	0.00	31.47	22.51	-
obstacle(1)	10,000	0.05	0.33	0.31	0.64
(2)			0.85	0.39	1.40
(3)			0.67	0.63	1.45
(4)			0.75	0.64	36.46*
(5)			0.87	0.35	1.69
(6)			1.25	0.65	154.19*
(7)			1.06	0.59	29.08*
(8)			1.17	0.43	3.05
obstacle(1)	40,000	0.01	5.61	3.16	22.98
(2)			8.74	3.54	48.76
(3)			11.28	5.65	3402.44*
(4)			12.15	5.65	1746.77*
(5)			8.39	2.62	52.92
(6)			31.46	9.00	262.90
(7)			18.34	5.74	2272.43*
(8)			10.69	2.44	120.46

cols/rows = 1168) and higher density (average density = 4.5%). While LUSOL spends slightly more time in the factor and update routines, its speed in the solve routines outweighs the other statistics and makes it more efficient than UMFPACK. When the problem size increases in Set II (20 LCPs with average number of cols/rows = 3795 and density = 4.4%), the advantage of UMFPACK over LUSOL in factor and update routines becomes more significant, and it outperforms LUSOL. The problems in Set III are generated with the same size as Set II problems but with reduced density (of 0.3% on average). For these systems, LUSOL works better than UMFPACK because the factor time taken by these two options is comparable, and even though LUSOL takes more time performing updates, it is much faster in its solve routines. Similar observations are obtained of the distribution of the basis time with the MCPLIB problems. In particular, LUSOL spends 82%, 13%, and 5% of basis time in factor, update, and solve routines, where UMFPACK spends 51%, 4%, and 45% of basis time in these routines, respectively.

Table 9: Distribution of Basis Time for Randomly Generated LCPs

	LUSOL		UMFPACK	
Set	Factor/Update/Solve	Basis Time	Factor/Update/Solve	Basis Time
I	22.89/7.00/2.26	32.15	17.24/1.67/20.66	39.57
II	821.16/642.69/47.75	1511.61	172.59/88.18/324.08	584.85
III	1.11/11.14/2.86	15.08	1.86/4.39/33.56	39.77

3.4 Summary

We have shown that incorporating the UMFPACK package as an alternative to the LUSOL basis package in PATH improves the solution of large-scale problems in that the time spent in the basis package (hence in the overall program) is reduced and the reliability or accuracy of the solution is increased. This advantage is most significant when the solution process is dominated by the crash procedure. However, on general sparse problems requiring large number of pivots in the complementarity subproblems (such as the LCP examples) or on small problems, the LUSOL basis package tends to be more reliable and more efficient than UMFPACK, in part due to the efficiency of the solve routines in LUSOL. Therefore, the LUSOL basis package remains the default basis option in the PATH code. Nevertheless UMFPACK is still comparable to LUSOL in these cases and its large-scale computing capability does not cost us much loss of efficiency.

The alternative of using the COIN basis package is motivated by COIN-LP's better performance than the LUSOL-based MINOS/SNOPT in solving LPs. In solving the MCPLIB problems, the COIN basis package is the most effective of the three options in the number of successes with the help of the linear refinements and scaling procedures. The trade-off, however, is an increase in the solution time. It is possible that utilizing these methods more generally within PATH would improve robustness with other basis routines. In solving large-scale systems, COIN is less efficient than LUSOL and UMFPACK; and for several large instances, we observe numerical instability with COIN. As an open source code, COIN-OR utilities have the advantage of being under constant modification and improvement. Hence we believe that the COIN has the potential to perform better in the future.

Our purpose in enhancing the PATH solver is to solve general MCPs with better efficiency and not for any specific structured MCPs. Therefore it is advantageous to have interchangeable basis packages in PATH, since their performances vary with different problem characteristics. The new options are based on open source linear system packages. The basis package structure in PATH allows easy updates to the newest version of these packages as they are available, without having to change either the PATH source or the linear system packages.

Chapter 4

A New AVI Solver

We have shown in Chapter 1 the relationship between AVI and complementarity problems: any AVI can be reformulated as a linear complementarity problem. However, this conversion may at times make the resulting LCP harder for currently available LCP or MCP solvers to solve. Therefore it is important to design a new solver (PathAVI) which is able to exploit the structure of the original AVIs. We first show several examples that motivate the development of the new solver in Section 4.1. The new solver is based on a scheme originally proposed by Cao and Ferris [5], which was further extended in [4]. Section 4.2 gives a brief description of Cao & Ferris algorithm, together with some key results of its termination properties. Unfortunately Cao & Ferris method becomes cumbersome as the problem size grows larger. Therefore it is important that the new solver is designed to be efficient at solving large-scale problems. Section 4.3 gives a detailed description of the implementation of the new solver and some theoretical justifications underlying this scheme. Since a key issue of our solver is to have an efficient linear system package to implement the pivotal scheme, we devote the rest of Section 4.4 to a description of a number of different options for performing necessary linear algebra functionalities. Computational results with comparisons between different basis options are presented in Section 4.5. Finally, we conclude this chapter with some discussion on an algorithm framework for general nonlinear variational inequalities as an extension of PathAVI.

Before proceeding, we give a few words about a *basis* of a vector space used in this chapter. It is defined as a linearly independent subset of the vector space that spans (or generates) the vector space.

4.1 Motivation

We know from the discussion in Chapter 1 that an affine variational inequality problem (AVI) can be expressed in terms of a generalized equation as follows:

$$0 \in Mz - q + N_{\mathcal{C}}(z), \tag{4.1.1}$$

with $M \in \mathbb{R}^{n\times n}$ and $\mathcal{C} = \{z \,|\, Bz \geq b\}$. This problem can be reformulated as a linear complementarity problem by introducing a variable u:

$$0 = Mz - q - B^\top u \quad \perp \quad z \text{ free} \tag{4.1.2}$$

$$0 \leq Bz - b \quad \perp \quad u \geq 0. \tag{4.1.3}$$

We also mentioned in Chapter 1 that this transformation is sometimes less desirable especially when the underlying polyhedral set of the original AVI is bounded. This situation happens for example when we express the first order optimality conditions for a quadratic program:

$$\begin{aligned} &\min_{z} \ \frac{1}{2} z^\top M z - z^\top q \\ &\text{s.t.} \ \ z \in \mathcal{C} := \{z | Bz \geq b\}, \ \text{bounded,} \end{aligned}$$

using an AVI:

$$0 \in Mz - q + N_{\mathcal{C}}(z).$$

The new AVI solver (PathAVI) is able to exploit the structure of $\mathcal{C}$ and only generates feasible points in the bounded set. Moreover, PathAVI is proven (in the following sections) to be able to process AVIs associated with copositive-plus matrices with respect to the recession cone of $\mathcal{C}$, and hence is guaranteed to solve AVIs over a bounded polyhedral set. The LCP formulation of the optimality condition on the other hand is

$$0 \in \begin{bmatrix} M & -B^\top \\ B & \end{bmatrix} \begin{bmatrix} z \\ u \end{bmatrix} - \begin{bmatrix} q \\ b \end{bmatrix} + N_{\mathbb{R}^n \times \mathbb{R}^m_+}\left(\begin{bmatrix} z \\ u \end{bmatrix} \right), \tag{4.1.4}$$

which is over an unbounded set $\mathbb{R}^n \times \mathbb{R}^m_+$ and is harder to solve (requiring more restrictive properties on M) for existing LCP solvers. We have discussed and shown by Nash equilibrium problems in Chapter 1 that AVI is preferably used to express many standard problems arising from applications where system equilibrium is the concern.

We now present a few concrete AVI examples to compare the PathAVI solver on the original AVIs and PATH on their LCP reformulations. We start with a simple AVI example with

$$M = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix}, q = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}.$$

Note that the polyhedral set is a unit simplex and hence bounded. PathAVI is able to solve this problem in 3 pivots. This AVI can be reformulated equivalently as a standard LCP:

$$\begin{aligned} 0 \le Mz - q - B_{3\cdot}^\top u \quad &\perp \quad z \ge 0 \\ 0 \le B_{3\cdot} z - b \quad &\perp \quad u \ge 0, \end{aligned}$$

with $B_{3\cdot}$ being the 3rd row of matrix B. When processing the above LCP using the traditional Lemke's Method, the scheme determines a ray termination and fails to find a solution. With the more effective PATH solver however, a solution to this LCP is obtained also in 3 pivots. The advantage of PATH over the traditional Lemke's method is in part due to the fact that PATH does an advanced (regular) start instead of a standard all slack ray start.

A few more slightly bigger $AVI(M, q, \mathcal{C})$ examples are generated to show that the LCP transformation of certain classes of AVIs makes the problem harder to solve. The AVIs are constructed with $M_{n\times n}$ symmetric and having approximately 20% negative eigenvalues. The underlying polyhedral sets are generated as bounded sets, where B has dimension $m \times n$. Table 10 lists the results of comparing the PathAVI solver on the original AVIs with the PATH solver on their equivalent LCP reformulations. *Status* indicates whether the solver succeeds (S) or fails (F) to solve the problem instance. *Iter* is the number of pivots taken. *Time* is the CPU-seconds spent on the problems. In general, PathAVI and PATH take similar amount of iterations. For the problems marked by $*$, PathAVI takes significantly fewer pivots than PATH on their LCP reformulations. The highlighted cases suggest that PATH fails to solve the LCP reformulations after a huge number of iterations (it terminated in both cases due to exceeding an iteration limit), while PathAVI solves these problems quite effectively.

Table 10: PathAVI on AVI vs PATH on Their LCP Reformulations

		PathAVI			*PATH*	
Size (m, n)	*Status*	*Iter*	*Time*	*Status*	*Iter*	*Time*
$(180, 60)$	S	48	0.005	S	56	0.013
	S	83	0.006	S	148	0.02
	S	103*	0.008	S	1816*	0.11
	S	**193**	**0.03**	**F**	**10176**	**0.88**
$(360, 120)$	S	183*	0.05	S	5986*	1.01
	S	263	0.06	S	86	0.05
	S	148	0.04	S	167	0.08
	S	**1516**	**0.36**	**F**	**10594**	**2.33**

It is clear from the above examples that it is necessary to develop a new solver designed to exploit the underlying structure of affine variational inequalities.

Furthermore, the above transformation between $AVI(M, q, \mathcal{C})$ and the linear complementarity problem (4.1.4) is a special case of a reduction method proposed by Robinson [67]. In general, the reduction method can be applied to variational inequalities having the following form:

$$0 \in \begin{bmatrix} 0 & A \\ -A^\top & 0 \end{bmatrix} \begin{bmatrix} p \\ y \end{bmatrix} + \begin{bmatrix} -d(p) \\ f \end{bmatrix} + N_{P\times Y}\left(\begin{bmatrix} p \\ y \end{bmatrix}\right), \tag{4.1.5}$$

where A is an $m \times n$ matrix, $f \in \mathbb{R}^n$, $d(p) \in \mathbb{R}^m$, and where P and Y are nonempty polyhedral convex subsets of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively.

Assuming that Y is a cone and Y° is the polar cone of Y:

$$Y^\circ = \{y^* \,|\, \langle y^*,\ y\rangle \le 0,\ \forall y \in Y\}.$$

Let $Z = \{p \,|\, A^\top p - f \in Y^\circ\}$. The solution to the above variational inequality (4.1.5) can be equivalently obtained by solving the following reduced generalized equation of p:

$$0 \in -d(p) + N_{P\cap Z}(p),$$

and recovering y from the corresponding normal cone expression.

The linear complementarity problem in (4.1.4) can be written as:

$$0 \in \begin{bmatrix} 0 & -B^\top \\ B & 0 \end{bmatrix} \begin{bmatrix} z \\ u \end{bmatrix} + \begin{bmatrix} Mz - q \\ -b \end{bmatrix} + N_{\mathbb{R}^n\times\mathbb{R}^m_+}\left(\begin{bmatrix} z \\ u \end{bmatrix}\right).$$

Therefore by letting $-d(p) = Mz - q$, $f = -d$, $P = \mathbb{R}^n$ and $Y = \mathbb{R}^m_+$, this problem can be reduced to the original AVI 4.1.1:

$$0 \in Mz - q + N_{\mathcal{C}}(z).$$

This reduction method can be applied to more general variational inequalities in the form of (4.1.5) with Y a cone, and is not limited to complementarity problems over orthant $\mathbb{R}^m_+$. It is able to achieve large reductions in numbers of variables, hence reducing the search space for solutions. (An even more general reduction scheme not restricted to Y being a cone is also proposed in [67].) Once again the reduction method suggests that it is worthwhile developing a new solver for solving variational inequalities over general polyhedral sets, since methods restricted to complementarity formulations are incapable of taking advantage of this reduction procedure.

4.2 Cao & Ferris Method

Cao & Ferris algorithm [5] is the basis for building the new solver. The central idea of the scheme contains the following stages:

Stage I Remove possible lines in the polyhedral set using a factorization procedure.

Stage II Construct an extreme point (x_e), and reduce the problem further by removing equality constraints.

Main Algorithm Assuming the resulting AVI from the first two stages can be expressed via a normal map reformulation as follows:

$$M_{\mathcal{C}}(x) = 0$$
$$\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b\},$$

where $M_{\mathcal{C}}(x) = M\pi_{\mathcal{C}}(x) - q + x - \pi_{\mathcal{C}}(x)$ is the normal map, choose a vector e from the interior (int) of $N_{\mathcal{C}}(x_e)$ and construct a piecewise-linear $(n+1)$-manifold $\mathcal{M}$ in $\mathbb{R}^{n+1}$ by forming the Cartesian product of each cell of the corresponding normal manifold $\mathcal{N}_{\mathcal{C}}$ with $\mathbb{R}_+$. Define a piecewise linear (PL) function $F : \mathcal{M} \to \mathbb{R}^n$ by

$$F(x, \mu) = M_{\mathcal{C}}(x) - \mu e. \tag{4.2.1}$$

We apply the algorithm of Eaves [25] to the PL equation $F(x, \mu) = 0$, using a ray start at $(w(\mu), \mu)$, with $w(\mu)$ defined as

$$w(\mu) = x_e + (q - Mx_e) + \mu e.$$

For large positive μ, $w(\mu)$ lies interior to the cell $\{x_e + N_{\mathcal{C}}(x_e)\}$ of $\mathcal{N}_{\mathcal{C}}$. The algorithm then proceeds in the direction $(-e, -1)$ until it either generates, in finitely many steps, a zero (x^*, μ^*) of the PL function $F(x, \mu)$ with $\mu^* = 0$, or determines infeasibility. In the first case it is clear that x^* satisfies $M_{\mathcal{C}}(x^*) = 0$, hence solves the original AVI.

More details of these parts are presented in the rest of this section, together with some discussion on the properties of the algorithm. We start by assuming a general formulation of $AVI(M, q, \mathcal{C})$ with

$$\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b, Hz = h\}.$$

4.2.1 Stage I

As explained in [66] (Proposition 4.1), we compute the basis for the lineality space of $\mathcal{C}$ defined as

$$lin\mathcal{C} = ker \begin{bmatrix} B \\ H \end{bmatrix},$$

by performing a QR factorization. Suppose $[W\ V]$ denotes its basis and its orthogonal complement, the reduced problem after removing lines is $AVI(\tilde{M}, \tilde{q}, \widetilde{\mathcal{C}})$ where

$$\tilde{M} = U^\top M U, \quad \tilde{q} = V^\top(\mathbb{I} - MZ)q,$$
$$\widetilde{\mathcal{C}} = \{\tilde{z} | \tilde{B}\tilde{z} \geq b, \tilde{H}z = h\}, \quad \tilde{B} = BV, \quad \tilde{H} = HV,$$

and

$$Z = W(W^\top M W)^{-1} W^\top, \quad U = (\mathbb{I} - ZM)V.$$

We consider the original solution pair expressed as (x, z), where x is a zero of the normal map, and $z = \pi(x)$ is the solution to the AVI. They can be recovered from the solution $(\tilde{x}, \tilde{z})$ to the reduced problem by

$$x_l = Z(q - MV\tilde{z})$$
$$x = x_l + V\tilde{x}$$
$$z = x_l + V\tilde{z}.$$

Note that in order for Stage I of Cao & Ferris method to work, a necessary condition is that $W^\top M W$ is invertible, i.e. the matrix M is invertible in the lineality space of $\mathcal{C}$. Otherwise, a different scheme needs to be exploited [4], which will be discussed further later.

4.2.2 Stage II

In Stage II, the algorithm tries to find an extreme point $\tilde{z}_e$ in $\widetilde{\mathcal{C}}$, that is a basic feasible solution of the following system:

$$\begin{bmatrix} \tilde{B} & -\mathbb{I} \\ \tilde{H} & 0 \end{bmatrix} \begin{bmatrix} z \\ s \end{bmatrix} = \begin{bmatrix} b \\ h \end{bmatrix}, \ s \geq 0, \ z \text{ free.}$$

The reduction in Stage I guarantees full column rank of $\begin{bmatrix} \tilde{B} \\ \tilde{H} \end{bmatrix}$ and this matrix is usually nonsquare (with more rows than columns). In finding a basic feasible solution, a QR factorization is computed, followed by a sequence of updates to the resulting QR factors. In particular, the following auxiliary problem is constructed

$$\begin{array}{ll} \min\limits_{z, z_{aux}, s} & z_{aux} \\ \text{subject to} & \begin{bmatrix} \tilde{B} & -\mathbb{I} \\ \tilde{H} & 0 \end{bmatrix} \begin{bmatrix} z \\ s \end{bmatrix} + \left(\begin{bmatrix} b \\ h \end{bmatrix} - \begin{bmatrix} \tilde{B} \\ \tilde{H} \end{bmatrix} y_* \right) z_{aux} = \begin{bmatrix} b \\ h \end{bmatrix} \\ & s \geq 0,\ z \text{ free},\ z_{aux} \geq 0. \end{array}$$

Note that by letting $y_* = [1 \cdots 1]^\top$, $z = [1 \cdots 1]^\top, z_{aux} = 1, s = 0$ constitute an initial feasible point for this problem, with basic variables (z, z_{aux}). A QR factorization of the corresponding initial nonsquare basis matrix needs to be computed (instead of an LU factorization on a square system). The algorithm then proceeds by performing the revised simplex method on the above linear program. At each pivot, the dual variable and the incoming column are computed in the least square sense using the currently available QR factors; and the basis matrix is then updated either by a rank-one update to the QR factors or by adding a column to the factors. The details of this updating technique can be found in Golub and Van Loan [45].

After acquiring an extreme point $\tilde{z}_e$ (with $z_{aux} = 0$ at the solution of the above linear program), Cao & Ferris algorithm reduces the problem further by forcing the iterates to lie in the affine space generated by the equality constraints. In particular, it finds the basis Y for $ker(\widetilde{H})$ by a QR factorization, then reduces the problem to $AVI(\bar{M}, \bar{q}, \bar{\mathcal{C}})$ with

$$\bar{M} = Y^\top \tilde{M} Y, \quad \bar{q} = Y^\top (\tilde{q} - \tilde{M} \tilde{z}_e)$$

and

$$\bar{\mathcal{C}} = \{\bar{z} \,|\, \bar{B}\bar{z} \geq \bar{b}\}, \quad \bar{B} = \tilde{B}Y, \quad \bar{b} = b - \tilde{B}\tilde{z}_e.$$

This process retains the full column rank property of $\bar{B}$ after the transformation and an extreme point of the new set $\bar{\mathcal{C}}$ is $\bar{z}_e = 0$ ([5]). The solution pair $(\tilde{x}, \tilde{z})$ prior to Stage II can be recovered from the solution $(\bar{x}, \bar{z})$ to $AVI(\bar{M}, \bar{q}, \bar{\mathcal{C}})$ by

$$\tilde{x} = \tilde{z}_e + Y\bar{x}$$
$$\tilde{z} = \tilde{z}_e + Y\bar{z}.$$

4.2.3 Main Algorithm

Stage I and II ensure that at the beginning of the main algorithm, the reduced $AVI(\bar{M}, \bar{q}, \bar{\mathcal{C}})$ is over a polyhedral set containing no lines, having an extreme point and only inequality constraints. In this section, we focus our discussion on the implementation of the Main Algorithm outlined previously, where a path is generated using a pivotal scheme. From here on we drop the "bar" notation on variables x and z, but they are in fact contained in a reduced space rather than the original problem space.

Here we are concerned with solving the PL equation $F(x, \mu) = 0$ defined in (4.2.1), which can also be expressed as

$$x - z + \bar{M}z - \bar{q} - \mu e = 0. \tag{4.2.2}$$

We start the path from the cell $\sigma_1 = \mathcal{F}_{\mathcal{A}} + N_{\mathcal{F}_{\mathcal{A}}}$ of the normal manifold associated with $\bar{\mathcal{C}}$. It is defined by the face containing the extreme point $\bar{z}_e$ in its relative interior. In particular, the corresponding active set $\mathcal{A}$ is used to defined the face $\mathcal{F}_{\mathcal{A}}$ and normal cone $N_{\mathcal{F}_{\mathcal{A}}}$.

Based on Lemma 3.4 from [5], the interior of $N_{\bar{\mathcal{C}}}(\bar{z}_e)$ is not empty due to the fact that $\bar{B}$ has full column rank. Therefore a point e can be chosen from the interior of $N_{\bar{\mathcal{C}}}(\bar{z}_e)$. Since $N_{\bar{\mathcal{C}}}(\bar{z}_e)$ can be expressed as

$$\{-\bar{B}_{\mathcal{A}}^{\top} u \mid u \geq 0\},$$

for uniformity and concreteness, we choose

$$e = -\bar{B}_{\mathcal{A}}^{\top} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}.$$

A better choice of e may be a topic of interest in future research.

A piece of the path contained in $\sigma_{\mathcal{A}}$ can be expressed as $(x, z, u_{\mathcal{A}}, s_{\mathcal{I}}, \mu)$ satisfying

$$\begin{array}{rcl} \bar{B}_{\mathcal{A}} z & = & \bar{b}_{\mathcal{A}} \\ \bar{B}_{\mathcal{I}} z - s_{\mathcal{I}} & = & \bar{b}_{\mathcal{I}}, s_{\mathcal{I}} \geq 0 \\ x - z & = & -\bar{B}_{\mathcal{A}}^{\top} u_{\mathcal{A}}, u_{\mathcal{A}} \geq 0 \\ \mu & \geq & 0. \end{array} \tag{4.2.3}$$

By combining equations (4.2.2), (4.2.3) and removing the x variable, we have:

$$\begin{array}{rcl} \bar{M} z - \bar{q} - \mu e & = & \bar{B}_{\mathcal{A}}^{\top} u_{\mathcal{A}} + \bar{B}_{\mathcal{I}}^{\top} u_{\mathcal{I}} \\ \bar{B}_{\mathcal{A}} z - s_{\mathcal{A}} & = & \bar{b}_{\mathcal{A}} \\ \bar{B}_{\mathcal{I}} z - s_{\mathcal{I}} & = & \bar{b}_{\mathcal{I}} \\ \mu \geq 0,\ u_{\mathcal{A}} \geq 0,\ u_{\mathcal{I}} & = & 0,\ s_{\mathcal{I}} \geq 0,\ s_{\mathcal{A}} = 0. \end{array}$$

We add the variables that are set to zero for completeness. As we move along the direction of $(-e, -1)$, the path changes its direction whenever it enters a new cell, which corresponds to a complementary pivot on the (u, s) pair.

The system is further reduced by considering the first cell corresponding to the extreme point and substituting z by the following expression:

$$z = \bar{B}_{\mathcal{A}}^{-1}(s_{\mathcal{A}} + \bar{b}_{\mathcal{A}}).$$

In the end we are able to apply Lemke's method on the following complementary problem:

$$\begin{array}{rcl} \bar{M}\bar{B}_{\mathcal{A}}^{-1}(s_{\mathcal{A}} + \bar{b}_{\mathcal{A}}) - \bar{q} - \mu e & = & \bar{B}_{\mathcal{A}}^{\top} u_{\mathcal{A}} + \bar{B}_{\mathcal{I}}^{\top} u_{\mathcal{I}} \\ \bar{B}_{\mathcal{I}}\bar{B}_{\mathcal{A}}^{-1}(s_{\mathcal{A}} + \bar{b}_{\mathcal{A}}) - s_{\mathcal{I}} & = & \bar{b}_{\mathcal{I}} \\ \mu \geq 0,\ u_{\mathcal{A}} \geq 0,\ u_{\mathcal{I}} & = & 0,\ s_{\mathcal{I}} \geq 0,\ s_{\mathcal{A}} = 0 \\ 0 \leq u & \perp & s \geq 0. \end{array}$$

4.2.4 Property of the Algorithm

To summarize, we present some definitions and cite some theoretical results on the termination properties of this algorithm.

We first introduce some extensions of certain matrix classes by defining them with respect to a cone.

Let K be a given closed convex cone. A matrix A is said to be **copositive** with respect to K if

$$z^{\top} A z \geq 0,\ \forall z \in K.$$

A matrix A is said to be **copositive-plus** with respect to K if it is copositive with respect to K and

$$z^{\top} A z = 0,\ z \in K\ \Rightarrow (A + A^{\top}) z = 0.$$

In the case of $AVI(M, q, \mathcal{C})$, we are interested in the **recession cone** of $\mathcal{C}$: for a nonempty, closed convex set $\mathcal{C}$ in $\mathbb{R}^n$,

$$rec\,\mathcal{C} := \{d \in \mathbb{R}^n \,|\, x + \lambda d \in \mathcal{C},\ \forall x \in \mathcal{C},\ \forall \lambda \geq 0\}.$$

The **dual cone** associated with set $\mathcal{C}$ is defined as:

$$\mathcal{C}^D := \{z^* \,|\, y^\top z^* \geq 0,\ \forall y \in \mathcal{C}\}.$$

Then the key result from Cao and Ferris [5] (Corollary 4.5) states:

Theorem 4.2.1. *Cao & Ferris method is guaranteed to either solve the $AVI(M, q, \mathcal{C})$ or determine that the problem is infeasible in finitely many steps, if M is copositive-plus with respect to $rec\,(\mathcal{C})$ and invertible in the lineality space of $\mathcal{C}$. In the latter case, the problem being infeasible means the following system has no solution*

$$Mz - q \in (rec\,\mathcal{C})^D, \quad z \in \mathcal{C}.$$

Furthermore, Cao & Ferris method claims that in practice, nondegeneracy can be assumed under lexicographical ordering [5].

4.3 PathAVI

From the description of Cao & Ferris method in the last section, we see that a lot of computational effort is spent in Stage I and Stage II with QR factorizations and updates. Another drawback of the algorithm is that the sparsity structure of the original problem is typically lost after these transformations. Substituting out the z variable involves extra matrix and matrix multiplications ($\bar{M}\bar{B}_{\mathcal{A}}^{-1}$ and $\bar{B}_{\mathcal{I}}\bar{B}_{\mathcal{A}}^{-1}$), which is also computationally expensive. In this section, we present a new solver that is essentially a large-scale implementation of Cao & Ferris method. It avoids all the costly operations aforementioned and performs the pivotal scheme with efficient basis routines.

Same as before, we assume the general expression of $AVI(M, q, \mathcal{C})$ with $\mathcal{C} = \{z \in \mathbb{R}^n | Bz \geq b, Hz = h\}$. The new solver essentially contains two parts:

Phase I Find a basic feasible solution in $\mathcal{C}$ by a revised simplex method.

Main Algorithm Perform complementary pivots on the original system with no reductions.

Details of these steps and relevant proofs are provided in the rest of this section.

4.3.1 Phase I

The revised simplex method is generally used in linear programming. We employ this technique to obtain a basic feasible solution for the constraints in $\mathcal{C}$. More details for the revised simplex method can be found in [32].

We first find a basis for the column space of the constraint matrix $\begin{bmatrix} B_{p\times n} \\ H_{q\times n} \end{bmatrix}$ and a basis for its row space by an LU factorization, namely index sets $I \in \{1, \cdots, p\}, J \in \{1, \cdots, q\}$ and $K \in \{1, \cdots, n\}$ such that $\begin{bmatrix} B_{IK} \\ H_{JK} \end{bmatrix}$ is square and invertible and

$$rank\left(\begin{bmatrix} B \\ H \end{bmatrix}\right) = |K| = |I| + |J|. \tag{4.3.1}$$

The linearly independent free variables are z_K and the dependent free variables are $z_{\bar{K}}$, where $\bar{K} = \{1, \cdots, n\} \backslash K$. We then obtain the value for z_K by

$$z_K = \begin{bmatrix} B_{IK} \\ H_{JK} \end{bmatrix}^{-1} \begin{bmatrix} b_I \\ h_J \end{bmatrix},$$

and fix dependent variables $z_{\bar{K}}$ at zero.

To set up the Phase I problem, we may need to introduce artificial variables to make the constraints feasible with the current value of z. In particular, for equality constraints, we only need to introduce artificial variables for the indices $j \in \{1, \cdots, q\} \backslash J$:

$$H_{jK} z_K + d_j a_j = h_j,$$

where

$$d_j = \begin{cases} -1, & \text{if } H_{jK} z_K \geq h_j \\ 1, & \text{if } H_{jK} z_K < h_j \end{cases}, \quad a_j = |H_{jK} z_K - h_j|.$$

For inequality constraints, we have

$$\begin{aligned} B_{IK} z_k - \mathbb{I}_{IK} s_I &= b_I \\ B_{\bar{I}K} z_k - \mathbb{I}_{\bar{I}K} s_{\bar{I}} &= b_{\bar{I}} \end{aligned}$$

with $\bar{I}$ denoting the complement of set I. We add the slack variables that are required to be nonnegative for Phase I problem. It is clear that $s_I = 0$, but $s_{\bar{I}}$ may not be feasible. In that case, an extra artificial variable $\hat{a}$ needs to be added, as in a standard linear programming Phase I process.

The Phase I problem therefore is to minimize the nonnegative artificial variables:

$$\begin{array}{rl} \min\limits_{z,s,a,\hat{a}} & \sum\limits_{j \in \bar{J}} a_j + \hat{a} \\ \text{subject to} & B_{IK} z_K - \mathbb{I}_{IK} s_I = b_I \\ & H_{JK} z_K = h_J \\ & B_{\bar{I}K} z_k - \mathbb{I}_{\bar{I}K} s_{\bar{I}} + e\hat{a} = b_{\bar{I}} \\ & H_{\bar{J}K} z_K + Da = h_J \\ & z_K \text{ free}, \ s \geq 0, \ a \geq 0, \ \hat{a} \geq 0. \end{array}$$

Here e is a column vector with 0, 1 elements associated with the artificial $\hat{a}$; D is a diagonal matrix with elements d_j; $\bar{J}$ is the complement to J. The initial basic variables thus are $(z_K, s_{\bar{I}}, a)$. Note that the dependent $z_{\bar{K}}$ variables never enter the basis in the subsequent simplex pivots and the free variables z_K never leave.

At the end of Phase I, if the objective value is zero, the original AVI is feasible and we are ready to enter the main algorithm (extra pivots might be needed, which will be illustrated in the following text). If the objective value is positive, we declare infeasibility of the original AVI and quit.

We make a few observations and remarks on the final status of Phase I. During Phase I pivots, if an artificial variable leaves the basis, it is fixed at zero and never reenters the basis. If at the end of Phase I, there exist artificial variables in the basis, we take extra steps to pivot them out of the basis until all of them are nonbasic or we encounter zero pivots. We claim that if there exist artificial variables that cannot be pivoted out of the basis, they can only be artificial variables associated with equality constraints. This happens when the rows in the pivoting tableau corresponding to these variables contain only zeros. If the value of this artificial variable is zero, this equality constraint is linearly dependent upon the rest of the constraints and can be removed from the rest of the computation; if this artificial variable has nonzero value, the AVI is infeasible. We now justify the claim we made earlier.

Lemma 4.1. *Assuming there are no empty constraints, the artificial $\hat{a}$ introduced for inequality constraints can always be pivoted to the top of the tableau, i.e. leaving the basis.*

Proof. : For simplicity we consider the Phase I problem with only inequality constraints:

$$\begin{array}{rl} \min & \hat{a} \\ \text{subject to} & Bz - \mathbb{I}s + e\hat{a} = b \\ & z \text{ free},\ s \geq 0,\ \hat{a} \geq 0. \end{array}$$

Suppose at the end of Phase I, the basic variables are $(z, s_{\bar{I}}, \hat{a})$. (Note that $s_{\bar{I}}$ actually only contains $|\bar{I}| - 1$ elements.) Correspondingly, the basis of the system (denoted by $A_{\cdot B}$) at the end of the Phase I is

$$A_{\cdot B} = \begin{bmatrix} B_I & \\ B_{\bar{I}} & E \end{bmatrix}, \quad \text{with } E = \left[\begin{array}{ccc|c} -1 & & & 1 \\ & \ddots & & \vdots \\ & & -1 & 0 \\ \hline & & & 1 \end{array}\right],$$

after a rearrangement of rows and columns. The last column in E is associated with the artificial variable, containing 0, 1 elements. The values of these basic variables are

$$\begin{aligned} \begin{bmatrix} z \\ s_{\bar{I}} \\ \hat{a} \end{bmatrix} &= \begin{bmatrix} B_I & \\ B_{\bar{I}} & E \end{bmatrix}^{-1} \begin{bmatrix} \mathbb{I}_I \\ 0 \end{bmatrix} s_I + A_{\cdot B}^{-1} b \\ &= \begin{bmatrix} B_I^{-1} & \\ -E^{-1} B_{\bar{I}} B_I^{-1} & E^{-1} \end{bmatrix} \begin{bmatrix} \mathbb{I}_I \\ 0 \end{bmatrix} s_I + A_{\cdot B}^{-1} b \\ &= \begin{bmatrix} B_I^{-1} \\ -E^{-1} B_{\bar{I}} B_I^{-1} \end{bmatrix} s_I + A_{\cdot B}^{-1} b. \end{aligned} \tag{4.3.2}$$

We only need to prove that the row corresponding to $\hat{a}$ in the pivoting tableau contains nonzero element(s), therefore $\hat{a}$ can always be pivoted out of the basis. In particular, we examine the last row of the matrix in (4.3.2). Since

$$E^{-1} = \left[\begin{array}{ccc|c} -1 & & & -1 \\ & \ddots & & \vdots \\ & & -1 & 0 \\ \hline & & & 1 \end{array}\right],$$

the last row of $E^{-1}B_{\bar{I}}$ is in fact the last row of $B_{\bar{I}}$, which by our assumption is nonempty. The last row of $E^{-1}B_{\bar{I}}B_I^{-1}$ is thus also nonempty. Therefore there exists nonzero pivot(s) corresponding to $\hat{a}$. □

Note that in the above and the following proofs, for simplicity we assume that B has full column rank, so that all z variables are in the basis. Removal of this assumption does not affect the proofs. We sometimes denote a set of rows (I) of a matrix by B_I instead of $B_{I\cdot}$ when there is no confusion.

Another observation of the basis system at the end of Phase I is:

Lemma 4.2. *The final basis* $\begin{bmatrix} B_I^1 & & \\ H_J^1 & & \\ B_{\bar{I}}^1 & -\mathbb{I}_{\bar{I}}^1 & \\ H_{\bar{J}}^1 & & E^1 \end{bmatrix}$ *corresponding to the basic feasible solution (after a certain rearrangement of rows) contains an invertible matrix* $\begin{bmatrix} B_I^1 \\ H_J^1 \end{bmatrix}$.

Proof. Since we apply the simplex method to find a basic feasible solution of the Phase I problem, at the end of the pivots, the basis is invertible. It suffices to prove that, in the final basis $\begin{bmatrix} B_I^1 \\ H_J^1 \end{bmatrix}$ is square. We obtain this result by proving that after each pivot, the block matrix of interest is always square.

Suppose we have an invertible basis at the k^{th} pivot with square matrix $\begin{bmatrix} B_{I^k} \\ H_{J^k} \end{bmatrix}$. The artificial variables are fixed at zero once they leave the basis, so the next entering variable is an s variable, say $s_{I^k}(c)$. Two types of pivots may occur:

- if $s_{I^k}(c)$ interchanges with $s_{\bar{I}^k}(r)$, after this pivot, the new set I^{k+1} becomes

$$I^{k+1} = (I^k \backslash \{c\}) \cup \{r\}$$
$$\bar{I}^{k+1} = (\bar{I}^k \cup \{c\}) \backslash \{r\}$$

- if $s_{I^k}(c)$ interchanges with artificial(r), after this pivot,

$$I^{k+1} = (I^k \backslash \{c\})$$
$$J^{k+1} = J^k \cup \{r\}.$$

In both cases, the cardinality of $|I^{k+1}| + |J^{k+1}|$ remains the same as $|I^k| + |J^k|$, hence $\begin{bmatrix} B_{I^{k+1}} \\ H_{J^{k+1}} \end{bmatrix}$ remains square after a simplex pivot. $\square$

We conclude this section by presenting the final basis of the Phase I process and a discussion of its properties. The final basis of Phase I is:

$$\left[\begin{array}{c|c} B_{IK} & \\ H_{JK} & \\ \hline B_{\bar{I}K} & -\mathbb{I}_{\bar{I}} \end{array}\right],$$

corresponding to a basic feasible solution $(z_K, s_{\bar{I}}, z_{\bar{K}} = 0)$, which contains no artificial variables.

The upper left block of the basis is square and invertible (as proven in Lemma 4.2); and the index sets $I \cup J$ and K correspond to the bases of the row space and column space of $\begin{bmatrix} B \\ H \end{bmatrix}$ (4.3.1) respectively. Thus we have

$$rank \begin{bmatrix} B \\ H \end{bmatrix} = rank \begin{bmatrix} B_{IK} \\ H_{JK} \end{bmatrix} = rank \begin{bmatrix} B_{I\cdot} \\ H_{J\cdot} \end{bmatrix} = rank \begin{bmatrix} B_{\cdot K} \\ H_{\cdot K} \end{bmatrix}, \tag{4.3.3}$$

and $\begin{bmatrix} B_{I\cdot} \\ H_{J\cdot} \end{bmatrix}$ has full row rank and $\begin{bmatrix} B_{\cdot K} \\ H_{\cdot K} \end{bmatrix}$ has full column rank.

Note that the linear dependent equality constraints are removed from the final basis, so we will remove the subscript J of matrix H in the rest of the illustrations. In the special case when $\begin{bmatrix} B \\ H \end{bmatrix}$ has full column rank, $K = \{1, \cdots, n\}$.

Finally, we present a proposition which will be useful for the proofs in the next section.

Proposition 4.3. *Let $\mathcal{A}$ be a set of rows of B such that $B_{\mathcal{A}}$ has full row rank and $rank\,(B_{\mathcal{A}}) = rank\,(B)$, then the following holds:*

$$ker\,(B) = ker\,(B_{\mathcal{A}}).$$

Proof. Clearly $y \in ker\,(B) \Rightarrow y \in ker\,(B_{\mathcal{A}})$.

To prove the converse, let $\mathcal{I}$ be the complement of $\mathcal{A}$, we have

$$B_{\mathcal{I}} = XB_{\mathcal{A}} \text{ with nonzero matrix } X.$$

Then for any $y \in ker\,(B_{\mathcal{A}})$ and $y \neq 0$,

$$B_{\mathcal{I}}y = XB_{\mathcal{A}}y = 0.$$

Since $B_{\mathcal{A}}y = 0$, together with $B_{\mathcal{I}}y = 0$ in the above, we have $y \in ker\,(B)$. □

4.3.2 Main Algorithm

Based upon the equations (4.2.2) and (4.2.3) introduced in Section 4.2.3, we consider the following system in the Main Algorithm of PathAVI:

$$\begin{aligned} Mz - q - \mu e &= B_{\mathcal{A}}^\top u_{\mathcal{A}} + H^\top \lambda \\ B_{\mathcal{A}} z &= b_{\mathcal{A}} \\ Hz &= h \\ B_{\mathcal{I}} z - s_{\mathcal{I}} &= b_{\mathcal{I}} \\ z \text{ free},\ \mu \geq 0,\ u_{\mathcal{A}} &\geq 0, \lambda \text{ free},\ s_{\mathcal{I}} \geq 0. \end{aligned} \tag{4.3.4}$$

We compute the point $e = B_I^\top d_1 + H^\top d_2$ using the final basis of Phase I, which is also used to define the starting cell of the normal manifold in the Main Algorithm computation. (This choice of the starting point is justified later in this section.) This constrained system of equations (4.3.4) describes a piece of the path in a cell of the normal manifold associated with the original polyhedral set. It is different from the system in the Cao & Ferris Main Algorithm, in that it is formulated by the matrices in the original AVI problem without any reductions or transformations and it does not remove equality constraints or z variables. The new scheme still performs complementary pivots on the (u, s) pairs, and simply treats z and λ as free variables and thus they never leave the basis. In this section we discuss the equivalence of processing this system to the original Cao & Ferris scheme by first comparing it to the system after Stage I reduction then to the system after Stage II reduction. We then justify the choice of the starting basis, which is obtained from the Phase I process.

Consider the system after Stage I reduction discussed in Section 4.2.1:

$$\begin{array}{rcl} \tilde{M}\tilde{z} - \tilde{q} - \tilde{\mu}\tilde{e} &=& \tilde{B}_{\mathcal{A}}^{\top}\tilde{u}_{\mathcal{A}} + \tilde{H}^{\top}\tilde{\lambda} \\ \tilde{B}_{\mathcal{A}}\tilde{z} &=& b_{\mathcal{A}} \\ \tilde{H}\tilde{z} &=& h \\ \tilde{B}_{\mathcal{I}}\tilde{z} - \tilde{s}_{\mathcal{I}} &=& b_{\mathcal{I}} \\ \tilde{z} \text{ free}, \ \tilde{\mu} \geq 0, \ \tilde{u}_{\mathcal{A}} &\geq& 0, \tilde{\lambda} \text{ free}, \ \tilde{s}_{\mathcal{I}} \geq 0, \end{array} \tag{4.3.5}$$

with $\tilde{e} = \tilde{B}_{I}^{\top}d_1 + \tilde{H}^{\top}d_2$.

Lemma 4.4. *Processing the system (4.3.4) generates the same path as processing the system (4.3.5) after Stage I reduction.*

Proof. The piecewise linear path is determined by the directions of the pieces and the step lengths taken along the directions.

To compute a new direction of the path, the following systems of equations are solved, with the original system (4.3.4):

$$M\Delta z - \Delta\mu e = B_{\mathcal{A}}^{\top}\Delta u_{\mathcal{A}} + H^{\top}\Delta\lambda \tag{4.3.6}$$

$$B_{\mathcal{A}}\Delta z = 0 \tag{4.3.7}$$

$$H\Delta z = 0 \tag{4.3.8}$$

$$B_{\mathcal{I}}\Delta z - \Delta s_{\mathcal{I}} = 0, \tag{4.3.9}$$

and with the system (4.3.5) after the transformation:

$$U^{\top}MU\Delta\tilde{z} - \Delta\tilde{\mu}\tilde{e} = V^{\top}B_{\mathcal{A}}^{\top}\Delta\tilde{u}_{\mathcal{A}} + V^{\top}H^{\top}\Delta\tilde{\lambda} \tag{4.3.10}$$

$$B_{\mathcal{A}}V\Delta\tilde{z} = 0 \tag{4.3.11}$$

$$HV\Delta\tilde{z} = 0 \tag{4.3.12}$$

$$B_{\mathcal{I}}V\Delta\tilde{z} - \Delta\tilde{s}_{\mathcal{I}} = 0 \tag{4.3.13}$$

respectively, with $U = (\mathbb{I} - ZM)V, Z = W(W^\top MW)^{-1}W^\top$, W being the basis of $ker(\begin{bmatrix} B \\ H \end{bmatrix})$ and V being the complement of W.

Let $\Delta z = (\mathbb{I} - ZM)V\Delta\tilde{z}$, equation (4.3.7) and (4.3.8) are equivalent to the corresponding equations in the reduced system (4.3.11 and 4.3.12), due to the fact that W is in the kernel space. Equation (4.3.9) is equivalent to (4.3.13) with $\Delta s_\mathcal{I} = \Delta\tilde{s}_\mathcal{I}$.

Multiply $[(\mathbb{I} - ZM)V]^{-1}$ on both sides of equation (4.3.6) gives:

$$U^\top MU\Delta z - \Delta\mu(V^\top B_I^\top d_1 + V^\top H^\top d_2) = V^\top B_\mathcal{A}^\top \Delta u_\mathcal{A} + V^\top H^\top \Delta\lambda,$$

due to the fact that $Z^\top MZ = Z^\top$. The above equation is therefore equivalent to (4.3.10) in the reduced system with $\Delta u_\mathcal{A} = \Delta\tilde{u}_\mathcal{A}$ and $\Delta\mu_\mathcal{A} = \Delta\tilde{\mu}_\mathcal{A}$.

Since both directions are uniquely determined by these systems of equations, we have $\Delta u_\mathcal{A} = \Delta\tilde{u}_\mathcal{A}$, $\Delta s_\mathcal{I} = \Delta\tilde{s}_\mathcal{I}$ and $\Delta\mu_\mathcal{A} = \Delta\tilde{\mu}_\mathcal{A}$.

The ratio test for determining the largest step length before hitting a boundary of the current cell is performed by finding the largest θ satisfying the following inequalities:

$$\begin{aligned} u_\mathcal{A} + \theta\Delta u_\mathcal{A} &\geq 0 \\ s_\mathcal{I} + \theta\Delta s_\mathcal{I} &\geq 0 \\ \mu + \theta\Delta\mu &\geq 0. \end{aligned}$$

Hence both the original and reduced methods obtain the same step length.

Now consider the initial pivot. It is easy to prove that the original system (4.3.4) is equivalent to the reduced system (4.3.5) by letting

$$z = x_l + V\tilde{z},$$

with $x_l = Z(q - MV\tilde{z})$ and the rest of the variables $u_\mathcal{A} = \tilde{u}_\mathcal{A}, \lambda = \tilde{\lambda}, \mu = \tilde{\mu}, s_\mathcal{I} = \tilde{s}_\mathcal{I}$.

The initial direction is determined in part by:

$$\begin{bmatrix} \tilde{B}_I \\ \tilde{H} \end{bmatrix} \Delta\tilde{z} = 0 \Rightarrow \Delta\tilde{z} = 0,$$

with the reduced system (4.3.5), since Stage I reduction ensures invertibility of the above matrix. We then obtain $\Delta\tilde{s} = 0$ from equation (4.3.13). Equation (4.3.10) thus becomes

$$-\Delta\tilde{\mu}\tilde{e} = \tilde{B}_I^\top \Delta\tilde{u}_{\mathcal{A}} + \tilde{H}^\top \Delta\tilde{\lambda}.$$

We choose the first $\Delta\tilde{\mu} = -1$ in order to force the path to move into the starting cell. By plugging in the expression for $\tilde{e}$,

$$\begin{bmatrix} \tilde{B}_I \\ \tilde{H} \end{bmatrix}^\top \begin{bmatrix} \Delta\tilde{u}_{\mathcal{A}} \\ \Delta\tilde{\lambda} \end{bmatrix} = \begin{bmatrix} \tilde{B}_I \\ \tilde{H} \end{bmatrix}^\top \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \Rightarrow \begin{bmatrix} \Delta\tilde{u}_{\mathcal{A}} \\ \Delta\tilde{\lambda} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix},$$

due to invertibility of the matrix. The initial direction with the original system (4.3.4) is determined in part by:

$$\begin{bmatrix} B_I & \\ H & \\ B_{\bar{I}} & -\mathbb{I}_{\bar{I}} \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta s \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} \Delta z \\ \Delta s \end{bmatrix} = 0.$$

This is true since the above matrix is the basis at the end of the Phase I process. At the same time, Phase I guarantees that $\begin{bmatrix} B_I \\ H \end{bmatrix}$ has full column rank (4.3.3), so similarly in equation (4.3.6) of the original system, by choosing the initial $\Delta\mu = -1$, we have $\Delta u_{\mathcal{A}} = d_1$ and $\Delta\lambda = d_2$.

We have proven that the pivots are precisely the same for the original system and the reduced system after Stage I. $\square$

Now we consider the second part of our equivalence statement, that is the pivotal steps taken by solving the system after Stage I reduction are identical to those taken by solving the system after Stage II reduction, which is

$$\begin{aligned} \bar{M}\bar{z} - \bar{q} - \bar{\mu}\bar{e} &= \bar{B}_{\mathcal{A}}^{\top}\bar{u}_{\mathcal{A}} \\ \bar{B}_{\mathcal{A}}\bar{z} &= \bar{b}_{\mathcal{A}} \\ \bar{B}_{\mathcal{I}}\bar{z} - \bar{s}_{\mathcal{I}} &= \bar{b}_{\mathcal{I}} \\ \bar{z} \text{ free},\ \bar{\mu} \geq 0,\ \bar{u}_{\mathcal{A}} &\geq 0,\ \bar{s}_{\mathcal{I}} \geq 0 \end{aligned} \tag{4.3.14}$$

with $\bar{e} = \bar{B}_I^{\top} d_1$. We have the following Lemma:

Lemma 4.5. *Processing the system (4.3.5) after Stage I generates the same path as processing the system after Stage II reduction (4.3.14).*

Proof. Similar logic employed in the proof of Lemma 4.4 can be used to prove that both the general pivots taken during the process and the initial step are the same with both systems.

Once again we consider the system of equations for computing the direction of the path at each pivot, expressed as:

$$\tilde{M}\Delta\tilde{z} - \Delta\tilde{\mu}\tilde{e} = \tilde{B}_{\mathcal{A}}^{\top}\Delta\tilde{u}_{\mathcal{A}} + \tilde{H}^{\top}\Delta\tilde{\lambda} \tag{4.3.15}$$

$$\tilde{B}_{\mathcal{A}}\Delta\tilde{z} = 0 \tag{4.3.16}$$

$$\tilde{H}\Delta\tilde{z} = 0 \tag{4.3.17}$$

$$\tilde{B}_{\mathcal{I}}\Delta\tilde{z} - \Delta\tilde{s}_{\mathcal{I}} = 0 \tag{4.3.18}$$

with system (4.3.5) after Stage I reduction, and the system of equations after Stage II reduction:

$$\begin{aligned} Y^\top \tilde{M} Y \Delta \bar{z} - \Delta \bar{\mu} \bar{e} &= Y^\top \tilde{B}_{\mathcal{A}}^\top \Delta \bar{u}_{\mathcal{A}} && (4.3.19) \\ \tilde{B}_{\mathcal{A}} Y \Delta \bar{z} &= 0 && (4.3.20) \\ \tilde{B}_{\mathcal{I}} Y \Delta \bar{z} - \Delta \bar{s}_{\mathcal{I}} &= 0 && (4.3.21) \end{aligned}$$

with Y being the basis of $ker(H)$.

Let $\Delta \tilde{z} = Y \Delta \bar{z}$, equation (4.3.16) is equivalent to the corresponding equation (4.3.20) in the further reduced system. Equation (4.3.17) gets removed since Y is in the kernel space of H. Furthermore, equation (4.3.18) is equivalent to (4.3.21) with $\Delta \tilde{s}_{\mathcal{I}} = \Delta \bar{s}_{\mathcal{I}}$.

Multiply $Y^\top$ on both sides of equation (4.3.15) gives:

$$Y^\top \tilde{M} Y \Delta \tilde{z} - Y^\top (\tilde{B}_I^\top d_1 + \tilde{H}^\top d_2) \Delta \tilde{\mu} = Y^\top \tilde{B}_{\mathcal{A}}^\top \Delta \tilde{u}_{\mathcal{A}} + Y^\top \tilde{H}^\top \Delta \tilde{\lambda},$$

which is reduced to

$$Y^\top \tilde{M} Y \Delta \tilde{z} - Y^\top \tilde{B}_I^\top d_1 \Delta \tilde{\mu} = Y^\top \tilde{B}_{\mathcal{A}}^\top \Delta \tilde{u}_{\mathcal{A}},$$

due to the fact that Y is the basis of $ker(H)$. This equation is equivalent to equation (4.3.19) in the further reduced system with $\Delta \tilde{u}_{\mathcal{A}} = \Delta \bar{u}_{\mathcal{A}}$ and $\Delta \tilde{\mu} = \Delta \bar{\mu}$.

Since both directions are uniquely determined by these systems of equations, we have $\Delta \tilde{u}_{\mathcal{A}} = \Delta \bar{u}_{\mathcal{A}}$, $\Delta \tilde{s}_{\mathcal{I}} = \Delta \bar{s}_{\mathcal{I}}$ and $\Delta \tilde{\mu} = \Delta \bar{\mu}$.

In determining the largest step length before any of the above variables hits its boundary, ratio test is performed to find the largest θ satisfying the following inequalities:

$$\begin{aligned} u_{\mathcal{A}} + \theta \Delta u_{\mathcal{A}} &\geq 0 \\ s_{\mathcal{I}} + \theta \Delta s_{\mathcal{I}} &\geq 0 \\ \mu + \theta \Delta \mu &\geq 0. \end{aligned}$$

Therefore both the systems obtain the same step length.

Now consider the initial pivot. It is easy to prove (by similar strategy as the above) that the system (4.3.5) is equivalent to the further reduced system (4.3.14) by having

$$\tilde{z} = \tilde{z}_e + Y\bar{z},$$

and the rest of the variables $\tilde{u}_{\mathcal{A}} = \bar{u}_{\mathcal{A}}, \tilde{\lambda} = \bar{\lambda}, \tilde{\mu} = \bar{\mu}, \tilde{s}_{\mathcal{I}} = \bar{s}_{\mathcal{I}}$.

As seen in the proof of Lemma 4.4, the initial direction with the system after Stage I reduction is determined as

$$\Delta\tilde{z} = 0, \quad \Delta\tilde{s} = 0, \quad \Delta\tilde{\mu} = -1,$$

and with $\tilde{e} = \tilde{B}_I^\top d_1 + \tilde{H}^\top d_2$,

$$\Delta\tilde{u}_{\mathcal{A}} = d_1, \quad \Delta\tilde{\lambda} = d_2.$$

The initial direction with the system after Stage II reduction is determined by the following equations. First note that the initial active set is determined by the final basis of Phase I, namely $\mathcal{A} = I$. Since $\bar{B}_I$ has full column rank,

$$\bar{B}_I\Delta\bar{z} = 0 \Rightarrow \Delta\bar{z} = 0,$$

and thus

$$\bar{B}_{\mathcal{I}}\Delta\bar{z} - \Delta\bar{s}_{\mathcal{I}} = 0 \Rightarrow \Delta\bar{s}_{\mathcal{I}} = 0.$$

Equation (4.3.19) thus becomes

$$-\Delta\bar{\mu}\bar{e} = \bar{B}_I^\top\Delta\bar{u}_{\mathcal{A}}.$$

By choosing $\Delta\bar{\mu} = -1$, we have $\Delta\bar{u}_{\mathcal{A}} = d_1$. Therefore the initial directions determined by both systems are also equivalent independent of the choice for d_2.

We have proven that the system after Stage I reduction and the system after Stage II reduction generate the same path. $\square$

We now make a few remarks on the choice of the starting point.

As proven before, the starting point e in the original space is equivalent to $\tilde{e}$ in the reduced space after Stage I, and they both rely on the choice of $\begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$. After the transformation in Stage II, which is essentially a projection onto the lineality space of $\tilde{H}$, $\tilde{e}$ is equivalent to $\bar{e} = \bar{B}_I^\top d_1$ in the further reduced space.

As discussed in Section 4.2.3, d_1 is chosen to be a vector of minus ones. Therefore at the start of the Main Algorithm in the new solver, we choose the value for e by letting d_1 still be a vector of minus ones with proper dimension. Since d_2 will always be compressed during the removal of equality constraints, we simply choose d_2 to be a vector of zeros for our computation of a starting point.

We now focus our discussion on the initial basis of the Main Algorithm, which is based on the final basis of the Phase I process. We simplify the notation by omitting the H matrix since its rows are always contained in the active constraint set.

$$\left[\begin{array}{cc|c} -B_{\mathcal{A}}^\top & M & \\ & B_{\mathcal{A}} & \\ \hline & B_{\mathcal{I}} & -\mathbb{I}_{\mathcal{I}} \end{array}\right]$$

The invertibility of the above initial basis matrix depends on its upper left block matrix and we claim that

Lemma 4.6. $\begin{bmatrix} -B_{\mathcal{A}}^\top & M \\ & B_{\mathcal{A}} \end{bmatrix}$ *is invertible if and only if* $W^\top M W$ *is invertible.*

Here B *has dimension* $m \times n$*,* M *has dimension* $n \times n$*,* $W_{n\times k}$ *is the basis of* $ker(B)$ *and* $rank(B_{\mathcal{A}}) = rank(B)$*.*

Proof. Proof of "only if":

$$\begin{aligned}
\begin{bmatrix} -B_{\mathcal{A}}^{\top} & M \\ & B_{\mathcal{A}} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= 0 \\
&\iff -B_{\mathcal{A}}^{\top}x + My = 0, \quad y \in ker(B_{\mathcal{A}}) \\
&\iff -B_{\mathcal{A}}^{\top}x + MW\hat{y} = 0, \text{ assuming that } y = W\hat{y} \\
& \text{(by Proposition 4.3)} \\
&\iff -W^{\top}B_{\mathcal{A}}^{\top}x + W^{\top}MW\hat{y} = 0, \quad y = W\hat{y} \\
&\iff \hat{y} = 0 \Longrightarrow y = 0 \Longrightarrow x = 0.
\end{aligned}$$

Proof of "if":

$$\begin{aligned}
&\begin{bmatrix} -B_{\mathcal{A}}^{\top} & M \\ & B_{\mathcal{A}} \end{bmatrix} \text{ invertible} \\
\Longrightarrow\ & rank \begin{bmatrix} \mathbb{I}_{\mathcal{A}} & \\ & W \end{bmatrix} = rank \begin{bmatrix} -B_{\mathcal{A}}^{\top} & M \\ & B_{\mathcal{A}} \end{bmatrix} \begin{bmatrix} \mathbb{I}_{\mathcal{A}} & \\ & W \end{bmatrix} \\
& n = rank\,(B) + \ nullity\,(B) = rank\,(B_{\mathcal{A}}) + \ dim\,ker\,(B) = |\mathcal{A}| + k \\
\Longrightarrow\ & rank \begin{bmatrix} \mathbb{I}_{\mathcal{A}} & \\ & W \end{bmatrix} = n,
\end{aligned}$$

therefore

$$\begin{aligned}
& rank \begin{bmatrix} -B_{\mathcal{A}}^{\top} & MW \\ & B_{\mathcal{A}}W \end{bmatrix} = \ rank\,[-B_{\mathcal{A}}^{\top} \quad MW] = n \\
& k = rank\,W^{\top} = rank\,W^{\top}[-B_{\mathcal{A}}^{\top} \quad MW] = rank\,[W^{\top}MW] \\
\Longrightarrow \quad & W^{\top}MW \text{ invertible .}
\end{aligned}$$

$\square$

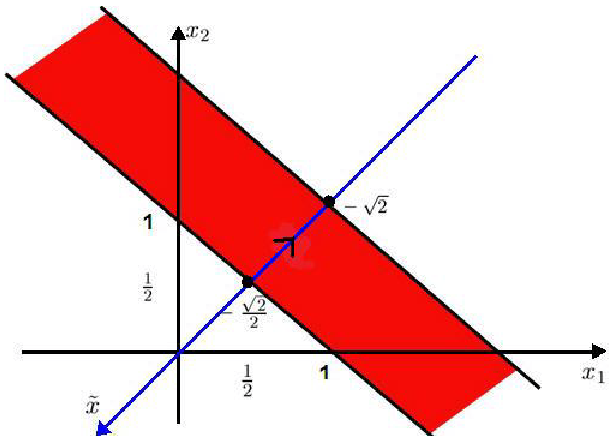


Figure 4: A simple example for comparing the path of Cao & Ferris and PathAVI

.

A special case is when B has full column rank in the original system. It is easy to see that the initial basis is invertible by a block partition.

Theorem 4.3.1. *The new PathAVI is equivalent to Cao & Ferris method.*

Proof. Obvious, from Lemma 4.4, Lemma 4.5 and Lemma 4.6. □

Therefore this new PathAVI solver is able to either solve the AVI or determine infeasibility. In practice, since it follows the same path as Cao & Ferris method, degeneracy issue can be resolved by always choosing the lexicographical minimum from the pivotal choices that achieve ties as mentioned in Section 4.2.4.

We hereby present a simple concrete example to illustrate the equivalence of the new method to the reduction method. Consider $AVI(M, q, \mathcal{C})$ with

$$B = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, q = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Set $\mathcal{C}$ is a stripe which contains lines as seen in Figure 4. Cao & Ferris method computes the basis of $ker\,(B)$ and its complement as:

$$[W\ V] = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}.$$

By Stage I, the reduced problem is $AVI(\tilde{M}, \tilde{q}, \widetilde{\mathcal{C}})$ with

$$\tilde{M} = [1], \quad \tilde{q} = -\sqrt{2},$$
$$\widetilde{\mathcal{C}} = \{\tilde{z} \in \mathbb{R} | -\sqrt{2} \leq \tilde{z} \leq -\frac{\sqrt{2}}{2}\}.$$

After the reduction the problem space is one dimensional, lying on the new axis marked by $\tilde{x}$ in Figure 4. It then finds an extreme point $\tilde{z}^0 = -\frac{\sqrt{2}}{2}$ in Stage II. In the Main Algorithm of Cao & Ferris method, the path goes from $\tilde{z}^0$ to $\tilde{z}^1$ (at $-\sqrt{2}$) in the reduced $1-D$ space.

Using the PathAVI solver, the Phase I stage ends with the final basis $I = \{1\}$, based on which the Main Algorithm starts with an invertible initial basis and generates a path from $z^0 = [0.5, 0.5]' \rightarrow z^1 = [1, 1]'$ in the original $2-D$ space. This path coincides with the one generated by the reduction scheme.

4.3.3 Extension of PathAVI

In the previous section, we considered AVIs that are solvable by the Cao & Ferris method illustrated in Section 4.2 and proved its equivalence to the new PathAVI solver. However the above schemes fail to proceed when $W^\top MW$ is not invertible. A different reduction method is proposed by Cao and Ferris [4] to remove this assumption. It transforms the original problem into one that can be processed by the aforementioned Cao & Ferris method, thus two sets of reduction schemes have to be used consecutively in this case.

Instead of incorporating this additional reduction scheme into our new solver, which is inefficient for large-scale problems, we simply extend the current PathAVI method to take this instance into consideration. In particular, the complementary system in the Main Algorithm system of PathAVI is augmented by introducing artificial variables:

$$\begin{aligned} -B^\top u - H^\top \lambda + Mz + \mathbb{I}a - \mu e - q &= 0 \\ Bz - s &= b \\ Hz &= h \\ z \text{ free},\ \lambda \text{ free},\ \mu \in [0,1],\ 0 \le u \perp s \ge 0,\ a &\equiv 0. \end{aligned} \tag{4.3.22}$$

The artificial variables (a) are used to construct an initial invertible basis consistent with the final basis obtained from Phase I. Once again, denote $\begin{bmatrix} B_I \\ H \end{bmatrix}$ in the final basis of Phase I simply by $B_{\mathcal{A}}$, the initial basis in the Main Algorithm is then:

$$\begin{bmatrix} -B_{\mathcal{A}\bar{K}}^\top & M_{\bar{K}K} & M_{\bar{K}\bar{K}} & \\ -B_{\mathcal{A}K}^\top & M_{KK} & M_{K\bar{K}} & \\ & B_{\mathcal{A}K} & B_{\mathcal{A}\bar{K}} & \\ & B_{\mathcal{I}K} & B_{\mathcal{I}\bar{K}} & -\mathbb{I} \end{bmatrix} \tag{4.3.23}$$

corresponding to variables $(u_{\mathcal{A}}(\lambda), z_K, z_{\bar{K}}, s_{\mathcal{I}})$.

When M is not invertible in the lineality space of $\mathcal{C}$, by Theorem 4.2.1, matrix (4.3.23) has no inverse. The basis at the end of Phase I suggests that its submatrix

$$\begin{bmatrix} -B_{\mathcal{A}K}^\top & M_{KK} \\ & B_{\mathcal{A}K} \end{bmatrix}$$

is invertible. By substituting artificial variables in place of $z_{\bar{K}}$, a full rank initial basis can be recovered:

$$\begin{bmatrix} -B_{\mathcal{A}\bar{K}}^{\top} & M_{\bar{K}K} & \mathbb{I}_{\bar{K}} & \\ -B_{\mathcal{A}K}^{\top} & M_{KK} & & \\ & B_{\mathcal{A}K} & & \\ & B_{\mathcal{I}K} & & -\mathbb{I} \end{bmatrix}. \tag{4.3.24}$$

Complementary pivots are performed subsequently on the (u, s) pairs as well as the (a, z) pairs. In particular, when a_i leaves the basis at a certain pivot, the corresponding entering variable is z_i, and once z variables enter the basis, they never leave.

The starting point (e) is constructed differently from before, in that we find $e = [e^1\ e^2]^{\top}$ such that

$$-B_{\mathcal{A}\bar{K}}^{\top} u_{\mathcal{A}} + M_{\bar{K}K} z_K^0 + \mu e^1 = q_{\bar{K}} \tag{4.3.25}$$

$$-B_{\mathcal{A}K}^{\top} u_{\mathcal{A}} + M_{KK} z_K^0 + \mu e^2 = q_K \tag{4.3.26}$$

$$u_{\mathcal{A}} \geq 0.$$

Here z_K^0 is obtained from the Phase I process.

We first construct e^2 from (4.3.26). $B_{\mathcal{A}K}^{\top}$ is known to be invertible from Phase I, therefore we compute $u_{\mathcal{A}}$ by

$$u_{\mathcal{A}}^0 = -B_{\mathcal{A}K}^{-\top}(-M_{KK} z_K^0 + q_K).$$

If $u_{\mathcal{A}}^0$ is feasible, choose e^2 to be a vector of zeros and the initial value of $u_{\mathcal{A}} = u_{\mathcal{A}}^0$. Otherwise, we can obtain a feasible point by setting $u_{\max} = \max\{-u_{\mathcal{A}}^0\}$ (> 0) and taking the following value for $u_{\mathcal{A}}$:

$$u_{\mathcal{A}} = u_{\mathcal{A}}^0 + u_{\max} d,$$

with the elements of d defined as

$$d_j = \begin{cases} 1, & \text{if } u^0_{\mathcal{A}_j} < 0 \\ 0, & \text{if } u^0_{\mathcal{A}_j} \geq 0. \end{cases}$$

Hence $e^2 = -B^\top_{\mathcal{A}K} u_{\max} d$ and

$$e^1 = B^\top_{\mathcal{A}\bar{K}} u_{\mathcal{A}} - M_{\bar{K}K} z^0_K + q_{\bar{K}}. \tag{4.3.27}$$

The initial value of μ^0 is chosen to be 1.

Theorem 4.3.2. *The extended PathAVI scheme is able to process (solve or determine infeasibility) $AVI(M, q, \mathcal{C})$, if M is copositive-plus with respect to the recession cone of $\mathcal{C}$.*

Proof. Since a generalized Lemke's method is exploited, and we have proven before that PathAVI is able to process AVI with an invertible basis at the beginning of the Main Algorithm, it suffices to prove that the extended PathAVI Main Algorithm with an augmented system starts from a ray. That is, the system (4.3.24) is feasible for all $\mu \geq \mu^0 (= 1)$, and we only need to prove this for the equations in (4.3.25) and (4.3.26).

For $\mu \geq \mu^0 > 0$,

$$u_{\mathcal{A}} = u^0_{\mathcal{A}} + \mu u_{\max} d > u^0_{\mathcal{A}} + \mu^0 u_{\max} d \geq 0.$$

By adding the artificial variables to equation (4.3.25), we have:

$$-B^\top_{\mathcal{A}\bar{K}} u_{\mathcal{A}} + M_{\bar{K}K} z^0_K + \mathbb{I}a + \mu e^1 = q_{\bar{K}}.$$

Plugging in $u_{\mathcal{A}} = u^0_{\mathcal{A}} + \mu u_{\max} d$ and rearranging the terms give us:

$$\mathbb{I}a + \mu(e^1 + e^2) = t, \tag{4.3.28}$$

with $t = B_{\mathcal{A}\bar{K}}^{\top} u_{\mathcal{A}}^0 - M_{\bar{K}K} z_K^0 + q_{\bar{K}}$. When $\mu = \mu^0$, the choice of e^1 in (4.3.27) makes the above equation feasible. For $\mu \geq \mu^0 > 0$, let $a = -(e^1 + e^2)(\mu - \mu^0)$ and $a \in [l, u]$, such that

$$\begin{cases} \text{if } (e^1 + e^2)_i = 0, & l_i = u_i = 0 \\ \text{if } (e^1 + e^2)_i > 0, & l_i = -\infty,\ u_i = 0 \\ \text{if } (e^1 + e^2)_i < 0, & l_i = 0,\ u_i = \infty \end{cases}$$

When μ decreases to μ^0 and the actual pivoting starts and proceeds, $a \equiv 0$. $\square$

4.4 Large-Scale Implementation

In our implementation of the Main Algorithm described above, we employ the basis package described in Chapter 2 for the factor, solve and update routines required by the pivotal technique. This enables the user to choose among a variety of sparse linear system packages and it is done at link time without having to change other parts of the code. We describe the use of three basis package options in this section. The first two options directly adopt two of the basis packages used in the PATH solver. The third basis option is new to the PathAVI solver. It essentially solves a reduced linear system, which is subject to new types of updates besides column replacement at each pivot. The last basis option is the main focus of this section.

As described in the previous section, the initial basis matrix of the main complementarity system

$$\begin{bmatrix} -B_{\mathcal{A}}^{\top} & M & \\ & B_{\mathcal{A}} & \\ & B_{\mathcal{I}} & -\mathbb{I}_{\mathcal{I}} \end{bmatrix} \begin{bmatrix} u_{\mathcal{A}} \\ z \\ s_{\mathcal{I}} \end{bmatrix} = \begin{bmatrix} q \\ b_{\mathcal{A}} \\ b_{\mathcal{I}} \end{bmatrix}$$

is associated with basic variables $(u_{\mathcal{A}}(\lambda), z, s_{\mathcal{I}})$. Similar to the PATH algorithm, each subsequent complementary pivot corresponds to a solve and a rank-one update in terms of a column replacement.

The first basis option in PathAVI exploits the LUSOL basis option, where the LUSOL linear package alone is able to provide all the linear functionalities (factor and solve and update) required by the pivotal routines.

The second basis option is the UMFPACK basis option, which once again relies on the UMFPACK package for factor and solve operations and on the block-LU method for rank-one updates.

The third basis package takes the idea of the aforementioned block-LU updating scheme coupled with the UMFPACK package, and extends the method to maintain the factors of a reduced matrix. In particular, a partition of the full basis matrix (H) can be formed as:

$$H = \begin{bmatrix} H_r & \\ D & -\mathbb{I} \end{bmatrix},$$

with $H_r = \begin{bmatrix} -B_{\mathcal{A}}^{\top} & M \\ & B_{\mathcal{A}} \end{bmatrix}$ and $D = \begin{bmatrix} 0 & B_{\mathcal{I}} \end{bmatrix}$. In solving the system $Hd = h$ in our case, h corresponds to the entering column at each pivot, which always has the form $\begin{bmatrix} h_r \\ 0 \end{bmatrix}$. The solution of this system may be obtained by

$$H_r d_1 = h_r, \quad d_2 = D d_1.$$

Therefore by partitioning, only a reduced system needs to be solved at each pivot instead of the full size basis system.

The reduced system with $H_r = \begin{bmatrix} -B_{\mathcal{A}}^\top & M \\ & B_{\mathcal{A}} \end{bmatrix}$ only contains basic columns corresponding to the basic variables $(u_{\mathcal{A}}(\lambda,\mu), z)$. Therefore besides replacing a column, a number of other types of updates need to be considered depending upon the types of variables leaving and entering the basis. In particular,

1. if u_i enters the basis and u_j leaves, replace a column in H_r.

2. if s_i enters the basis and u_j leaves, delete a column and row from H_r.

3. if u_i enters the basis and s_j leaves, add a column and row to H_r.

4. if s_i enters the basis and s_j leaves, replace a row in H_r.

Note that since addition(deletion) of a row and a column always occurs at the same time, H_r remains square when it expands or shrinks in dimension.

As illustrated in Section 2.3.2, by constructing an augmented matrix and applying the block-LU method to perform the deletion, addition and replacement we are able to perform all types of updates on this reduced system, which has the potential of speeding up our updates and improving the efficiency in large-scale problems. However, a drawback of the scheme is that unlike updating the full scale basis matrix where the column used in updating Y_k is always readily available from the previous solve, we often need an extra solve when updating the reduced system.

Let us recall the types of updates to matrices C_k and Y_k performed in the block-LU technique:

Case 1 Add a row $(r^\top)$ to $U_k^\top$, and add a column (w) to Y_k, resulting in adding a row and a column to C_k:

$$U_{k+1}^\top = \begin{pmatrix} U_k^\top \\ r^\top \end{pmatrix} \quad \text{and} \quad Y_{k+1} = (Y_k \quad w)$$

$$C_{k+1} = U_{k+1}^\top H_0{}^{-1} V_{k+1} = \begin{pmatrix} C_k & U_k^\top w \\ r^\top Y_k & r^\top w \end{pmatrix}.$$

Case 2 Replace a column of Y_k by w, and replace a column of C_k by $U_k^\top w$. U_k is unchanged.

Case 3 Replace a row of $U_k^\top$ by $r^\top$, and replace a row of C_k by $r^\top Y_k$. Y_k is unchanged.

Case 4 Delete a row from $U_k^\top$, and delete a column from Y_k, resulting in deleting a row and column from C_k.

Modifications to C_k and Y_k corresponding to the above four types of basis updates are illustrated one by one in the following subsections. In some cases, we will present concrete examples based on a small AVI problem, namely $AVI(M, q, \mathcal{C})$ defined over a polyhedral set $\mathcal{C} = \{z \in \mathbb{R}^2 \,|\, Bz \geq b\}$ with

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & -1 \end{bmatrix}.$$

Suppose the initial active set is $\mathcal{A} = \{1, 2\}$. The initial reduced system is then:

$$H_0 = \begin{bmatrix} M & -B_{\mathcal{A}}^\top \\ B_{\mathcal{A}} & \end{bmatrix} = \left[\begin{array}{cc|cc} m_{11} & m_{12} & -1 & 0 \\ m_{21} & m_{22} & 0 & -1 \\ \hline 1 & 0 & & \\ 0 & 1 & & \end{array}\right],$$

corresponding to basic variables (z_1, z_2, u_1, u_2). The set of basic variables associated with the full basis system is $(z_1, z_2, u_1, u_2, s_3)$. Note that we rearranged the columns in the order of $(z, u_{\mathcal{A}})$ in the above matrix expression and for simplicity we assume that the matrix is invertible. In the following illustration, we denote the active set associated with the initial reduced system (H_0) by $\mathcal{B}_0$ and its complement by $\mathcal{N}_0$. The initial active set $\mathcal{B}_0$ corresponds to both the initial basic columns $\begin{bmatrix} -B_{\mathcal{A}}^\top \\ 0 \end{bmatrix}$ associated with basic variables $u_{\mathcal{A}}$ and the basic rows $[B_{\mathcal{A}} \quad 0]$ associated with the active constraints. In this particular example, $\mathcal{B}_0 = \{1, 2\}$ and $\mathcal{N}_0 = \{3\}$.

4.4.1 Replace a Column in H_r

Column replacement is the standard type of updates that is explained in detail in Section 2.3.1. We hereby give an example where the entering variable is μ (from $\mathcal{N}_0$) associated with the ray column and the leaving variable is a u variable (from $\mathcal{B}_0$). This corresponds to the first type of column replacement.

In our AVI example, the ray (or the starting point in the interior of the initial normal cone) can be computed as

$$e = -B_{\mathcal{A}}^\top \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

Suppose u_1 is the basic variable that is being replaced by μ. This pivot corresponds to replacing the 3^{rd} column in H_0 by $c = [1 \quad 1 \quad 0 \quad 0]^\top$, since we have $-e\mu$ in the main PathAVI system (4.3.4). This update is accomplished by augmenting H_0 by $e_3^\top$ and column c and we obtain the following matrix:

$$H_1 = \left[\begin{array}{cc|cc|c} m_{11} & m_{12} & -1 & 0 & \mathbf{1} \\ m_{21} & m_{22} & 0 & -1 & \mathbf{1} \\ \hline 1 & 0 & & & \\ 0 & 1 & & & \\ \hline & & \mathbf{1} & & \end{array}\right] \Leftrightarrow \left[\begin{array}{cc|cc} m_{11} & m_{12} & 1 & 0 \\ m_{21} & m_{22} & 1 & -1 \\ \hline 1 & 0 & & \\ 0 & 1 & & \end{array}\right],$$

where the highlighted row/column indicates the modification made upon the previous basis. The matrix on the right-hand-side of "$\Leftrightarrow$" in the above expression is the actual system that needs to be solved, whose third column is explicitly replaced by a new column (c). If we denote this matrix by H_a, as explained in Section 2.3.1, the solution to this modified system $H_a y = h_a$ can be recovered from the solution of the following system

$$\begin{bmatrix} H_0 & c \\ e_3^\top & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_a \\ 0 \end{bmatrix} \tag{4.4.1}$$

by $y = y_1 + e_3\, y_2$. We shall use this notation ($\Leftrightarrow$) in the following examples to denote this equivalence relationship between an augmented system and the actual modified system. The above update corresponds to a special instance of **Case 1** in updating Y_k and C_k, that is

$$U_1 = e_3,\ Y_1 = H_0^{-1}\, c,\ C_1 = U_1^\top Y_1.$$

As mentioned before, in a real implementation, U_k only records the indices of the positions of the entering columns or the entering rows instead of the actual vectors. The basic variables associated with the full basis system becomes $(z_1, z_2, \mu, u_2, s_3)$.

4.4.2 Delete Row p and Column q from H_r

With different types of columns and rows being deleted from H_r, the corresponding updates to C_k and Y_k belong to one of the cases presented before. In particular:

Type I If the column (q) being deleted is from $\mathcal{B}_0$ and the row (p) being deleted is from $\mathcal{B}_0$, perform **Case 1** with $r = e_q$ and $w = H_0^{-1}e_p$. This is precisely the instance we have illustrated in Section 2.3.2.

Continuing from the previous example with H_1, when u_1 leaves the basis, its complementary variable s_1 enters the basis at the second pivot. Let us assume that it drives u_2 out of the basis. This basis change corresponds to deleting a column (4^{th}) and a row (3^{rd}) from H_1. To achieve this update, H_1 is augmented by a unit column and a unit row vector as follows:

$$H_2 = \left[\begin{array}{cc|cc|c|c} m_{11} & m_{12} & -1 & 0 & 1 & \\ m_{21} & m_{22} & 0 & -1 & 1 & \\ \hline 1 & 0 & & & & \mathbf{1} \\ 0 & 1 & & & & \\ \hline & & 1 & & & \\ \hline & & & \mathbf{1} & & \end{array}\right] \Leftrightarrow \left[\begin{array}{cc|c} m_{11} & m_{12} & 1 \\ m_{21} & m_{22} & 1 \\ \hline 0 & 1 & \end{array}\right].$$

The basic variables becomes $(z_1, z_2, \mu, s_1, s_3)$ for the full basis system.

From here on we no longer assume that the following updates take place consecutively from previous pivots and we make up the pivots with the sole purpose of illustrating more possible instances of updates.

Type II If the column being deleted is from $\mathcal{N}_0$, and the row (p) being deleted is from $\mathcal{B}_0$, perform **Case 2** with $w = H_0^{-1}e_p$.

We assume first that H_2 is modified by adding the u_1 column $[-1 \quad 0 \quad 0 \quad 0]^\top$ back together with a new row B_3. (corresponding to s_3). The augmented matrix becomes

$$H_3 = \left[\begin{array}{cc|cc|c|c} m_{11} & m_{12} & -1 & 0 & 1 & \\ m_{21} & m_{22} & 0 & -1 & 1 & \\ \hline 1 & 0 & & & & 1 \\ 0 & 1 & & & & \\ \hline \mathbf{-1} & \mathbf{-1} & & & & \\ \hline & & & 1 & & \end{array}\right]$$

We will explain this addition update in the next section.

We now start from H_3 and assume that another update to the current set of basic variables $(z_1, z_2, u_1, \mu, s_1)$ occurs with s_2 entering the basis and replacing variable μ. This pivot corresponds to deleting a column from $\mathcal{N}_0$ and a row from $\mathcal{B}_0$. The augmented matrix H_3 is modified by replacing the newly incorporated column (corresponding to μ) with a unit column vector marking the row that needs to be removed, in particular,

$$H_3 \rightarrow H_4 = \left[\begin{array}{cc|cc|c|c} m_{11} & m_{12} & -1 & 0 & & \\ m_{21} & m_{22} & 0 & -1 & & \\ \hline 1 & 0 & & & & 1 \\ 0 & 1 & & & \mathbf{1} & \\ \hline -1 & -1 & & & & \\ \hline & & & 1 & & \end{array}\right] \Leftrightarrow \left[\begin{array}{cc|c} m_{11} & m_{12} & -1 \\ m_{21} & m_{22} & 0 \\ \hline -1 & -1 & \end{array}\right].$$

Hence the size of the augmented matrix does not always increase at every update and it stays the same in this particular example.

To achieve this, **Case 2** update is invoked to modify Y_k and C_k. Specifically, U_3 is unchanged, Y_3 is modified by replacing its first column by $w = H_0^{-1}e_4$ and consequently the first column of C_3 is replaced by $U_3^\top w$.

Type III If the column (q) being deleted is from $\mathcal{B}_0$, and the row being deleted is from $\mathcal{N}_0$, perform **Case 3** with $r = e_q$.

Type IV If the column being deleted is from $\mathcal{N}_0$, and the row being deleted is from $\mathcal{N}_0$, perform **Case 4** with the corresponding row and column.

We add another example by assuming a slightly different pivot from the previous case. Instead of s_2 entering the basis and replacing μ, we have s_2 replacing s_3, which corresponds to a newly added row B_3. (contained in $\mathcal{N}_0$). An alternative modification to H_3 is carried out by removing the actual row and column.

$$H_3 \rightarrow \tilde{H}_4 = \left[\begin{array}{cc|cc|c} m_{11} & m_{12} & -1 & 0 & \\ m_{21} & m_{22} & 0 & -1 & \\ \hline 1 & 0 & & & 1 \\ 0 & 1 & & & \\ \hline & & & 1 & \end{array}\right] \Leftrightarrow \left[\begin{array}{cc|c} m_{11} & m_{12} & -1 \\ m_{21} & m_{22} & 0 \\ \hline 0 & 1 & \end{array}\right]$$

The update to Y_k and C_k then follows the **Case 4** situation. A row is removed from $U_3^\top$, and a column is removed from Y_3, thus a row and a column are removed from C_3.

4.4.3 Add a Column and Row to H_r

The type of updates to C_k and Y_k depends on the type of row and column being added:

Type I If the column (q) being added is from $\mathcal{N}_0$ and the row (p) being added is from $\mathcal{N}_0$, perform **Case 1** with $r = e_q$ and $w = H_0^{-1} e_p$. This is precisely the instance we have illustrated above.

Type II If the column (c) being added is from $\mathcal{N}_0$, and the row (p) being added is from $\mathcal{B}_0$, perform **Case 2** with already available $w (= H_0^{-1} c)$.

Type III If the column being added is from $\mathcal{B}_0$, and the row (r) being added is from $\mathcal{N}_0$, perform **Case 3** with new row r.

We now go back to the above example and describe the update from H_2 to H_3 at the third pivot. With basic variables $(z_1, z_2, \mu, s_1, s_3)$ for the full basis matrix at the end of the second pivot, or basic variables (z_1, z_2, μ) associated with the reduced system, we assume next that u_1 enters the basis and replaces s_3, which corresponds to column $c (= -B_{1\cdot}^\top)$ reentering basis and row $r (= B_{3\cdot})$ entering the reduced system. Since u_1 is in the initial basis $\mathcal{B}_0$ and s_3 is from $\mathcal{N}_0$, this is a Type III addition update. It is achieved by replacing the unit row (marking the originally replaced column c) by the actual entering row, specifically:

$$H_2 \to H_3 = \left[\begin{array}{cc|cc|c|c} m_{11} & m_{12} & -1 & 0 & 1 & \\ m_{21} & m_{22} & 0 & -1 & 1 & \\ \hline 1 & 0 & & & & 1 \\ 0 & 1 & & & & \\ \hline \mathbf{-1} & \mathbf{-1} & & & & \\ \hline & & & 1 & & \end{array}\right] \Leftrightarrow \left[\begin{array}{cc|cc} m_{11} & m_{12} & -1 & 1 \\ m_{21} & m_{22} & 0 & 1 \\ \hline -1 & -1 & & \\ 0 & 1 & & \end{array}\right].$$

The updates to Y_k and C_k belong to **Case 3** with a row replacement in $U_2^\top$, hence a row replacement in C_2. Y_2 is unchanged.

Type IV If the column being added is from $\mathcal{B}_0$, and the row being added is from $\mathcal{B}_0$, perform **Case 4** with the corresponding (unit) row and column.

4.4.4 Replace a Row in H_r

Replacing a row of H_0 is achieved by modifying C_k and Y_k as follows:

Type I If the entering row (r) is from $\mathcal{N}_0$ and the leaving row (p) is from $\mathcal{B}_0$, perform **Case 1** with new row r and $w = H_0^{-1}e_p$.

Type II If the entering row (r) is from $\mathcal{N}_0$, and the leaving row is from $\mathcal{N}_0$, perform **Case 3** with the new row r.

Type III If the entering row is from $\mathcal{B}_0$, say the p^{th} row, and the leaving row is from $\mathcal{B}_0$, perform **Case 2** with already available $w(= H_0^{-1}e_p)$.

Type IV If the entering row is from $\mathcal{B}_0$ (p^{th}), and the leaving row is from $\mathcal{N}_0$, perform **Case 4** with the corresponding row and the p^{th} column.

To summarize, we have presented a few examples to illustrate some typical types of updates especially associated with addition and deletion. The rest types of updates can be derived similarly. At each pivot, the augmented matrix is modified. The size of the augmented matrix can increase, decrease or remain the same due to different types of entering and leaving row/column. With each modification however, the block-LU structure is preserved, therefore the block-LU scheme can be extended to all of the above types of updates.

4.5 Computational Results

Exploiting sparse linear system packages and updating schemes enables PathAVI to process large-scale problems. The user can choose one of the basis options according to the characteristics of the AVI. The following AVIs are generated randomly (based on projection problems) and serve as a guideline as to which basis package might outperform the others on different types of systems.

Table 11: Smaller-scale $M_{n\times n}$ positive definite, $B_{m\times n}$ randomly generated with density 10%. Average size $n = 400$, $m = 768$

Basis Option	*Factor Time*	*Update Time*	*Solve Time*	*Total Time*
LUSOL	5.12	12.91	3.73	22.04
UMFPACK	5.19	9.11	34.97	49.58
REDUCED	5.06	34.61	28.34	68.84

Table 11 contains a total of 50 problems, which suggest that in solving relatively dense systems with medium size, LUSOL is most efficient among the three. UMFPACK performs factors and updates faster but is slow in the solve process. Pivots with a reduced system (with average size at most $2n$) use less solution time compared to UMFPACK on a full scale basis matrix (with average size $= m + n$). However as we mentioned earlier in this section, each update of the reduced version might require an additional solve, hence the total update time is much longer than the other two options.

Now we consider the total time of 20 larger-scale problems in Table 12 generated by the (otherwise) same method as the above smaller instances. We see that when the size of the problem increases, the UMFPACK basis package is faster than LUSOL with both full and reduced basis matrix options, due to the fact that in larger-scale cases UMFPACK performs factorizations much faster than LUSOL and with these relatively dense instances, block-LU updates with the full scale basis is much more efficient than the LUSOL updates. More motivation and comparison results on the performance of the LUSOL and UMFPACK basis packages can be found in Chapter 3.

Table 12: Larger-scale $M_{n\times n}$ p.d., $B_{m\times n}$ randomly generated with density 10%. Average size of $m = 2495$, $n = 768$

Basis Option	*Factor Time*	*Update Time*	*Solve Time*	*Total Time*
LUSOL	180.72	1159.1	69.38	1410.5
UMFPACK	47.27	301.4	494.46	844.1
REDUCED	46.59	673.30	411.47	1138.8

The next two sets of problems are generated with sparser systems and the results are summarized in Table 13. We generate the problem with m much larger than n, so that UMFPACK on a full scale basis matrix would work with a matrix of size $m + n$ which is much larger than the reduced basis matrix with size at most $2n$. (Average size of small-scale cases is $m = 768$, $n = 200$; average size of large-scale cases is $m = 2495$, $n = 650$.) In this case, the reduced basis package becomes competitive. However LUSOL still outperforms the UMFPACK based options.

Table 13: Larger-scale $M_{n\times n}$ p.d., $B_{m\times n}$ randomly generated with density 1%

	Basis Option	*Factor*	*Update*	*Solve*	*Total Time*
Total of 50	LUSOL	0.65	1.64	0.75	3.24
Small-scale	UMFPACK	0.99	1.39	11.18	13.81
Instances	REDUCED	0.91	5.90	5.09	12.56
Total of 20	LUSOL	9.81	40.68	12.34	63.66
Large-scale	UMFPACK	6.78	32.77	150.22	190.54
Instances	REDUCED	6.29	94.78	71.17	178.18

The computational results presented here suggest that the CPU time in PathAVI is dominated by the linear algebra associated with the pivotal scheme. Incorporating the sparse linear system packages enables PathAVI to process large-scale instances efficiently. Overall, the LUSOL option is always much more efficient in performing the solve functionality than the UMFPACK option at a cost during the factor and update routines. For smaller-scale systems and large-scale but sparse systems, LUSOL is overall the most efficient, since the speed of the LUSOL solve routines outweighs the other statistics. For large-scale problems when the system is relatively dense, the advantage of the factor and update routines in the UMFPACK options (of both the full scale basis and reduced basis) becomes more significant and they outperform the LUSOL option. When the reduced basis is much smaller than the full size basis, UMFPACK with the reduced options becomes competitive.

4.6 Discussion of a Nonlinear Extension

We have presented a new solver for solving affine variational inequalities together with some theoretical justification and numerical results. In particular, it is able to process a wide class of AVIs with the same convergence properties as the original Cao & Ferris method; at the same time, the implementation is effective in dealing with large-scale problems by avoiding costly algebraic operations and allowing easy inclusion of efficient linear system packages.

Furthermore, it provides the foundation for future development of a nonlinear variational inequality solver. To conclude this chapter, we propose a scheme for processing nonlinear VIs as an extension of PathAVI and discuss some issues and difficulties the nonlinear solver may encounter.

For a nonlinear $VI(f, \mathcal{C})$ with $f := \mathbb{R}^n \mapsto \mathbb{R}^n$, its corresponding normal map equation

$$f(\pi_{\mathcal{C}}(x)) + (x - \pi_{\mathcal{C}}(x)) = 0 \tag{4.6.1}$$

can be solved by a generalized Newton Method similar to the PATH algorithm in Section 3.1.3. Here we only consider nonlinear VI over polyhedral set $\mathcal{C}$. Any nonlinear constraints contained in the set can be treated by a complementarity reformulation with multipliers.

At the k^{th} iteration, $AVI(M^k, q^k, \mathcal{C})$ is constructed based on a piecewise affine approximation of the normal map at x^k with:

$$M^k = \nabla f(\pi_{\mathcal{C}}(x^k)), \quad q^k = f(\pi_{\mathcal{C}}(x^k)) - \nabla f(\pi_{\mathcal{C}}(x^k))\pi_{\mathcal{C}}(x^k).$$

Now the procedure employed by the PathAVI solver can be used to solve this affine variational inequality. As before, it constructs a parametric piecewise linear function

$$F(x(\mu), \mu) = M^k \pi_{\mathcal{C}}(x) + q^k + x - \pi_{\mathcal{C}}(x) - e\mu.$$

A parametric piecewise linear path is thus generated, which leads to the Newton point (x_N^k, z_N^k) at $\mu = 0$.

A nonmonotone search similar to the PATH algorithm can be adopted, namely the *m-steps*, together with *d-steps* and *watchdog steps*. Details of these steps are explained in Section 3.1.4. We note a few differences in implementing these schemes in the case of a nonlinear VI solver. They are in part due to the fact that the projection of a point from $\mathbb{R}^n$ onto a polyhedral set is not as easy to compute as the projection onto the positive orthant in the case of mixed complementarity problems.

First of all, the choice of merit functions might be limited. The Fischer-Burmeister function is associated with the complementarity relationship and cannot be directly adopted in the case of VI. A natural candidate for the merit function is the norm of the normal map residual

$$\Psi(x) = \frac{1}{2}||f_{\mathcal{C}}(x)||^2.$$

A Newton point is acceptable when

$$\Psi(x_N^k) \le (1-\sigma)\mathcal{R},$$

where σ is a small constant and $\mathcal{R}$ is a given reference value determined by former merit function values. This merit function is no longer differentiable. Therefore, we cannot perform the projected gradient scheme when the scheme fails to find a point satisfying the nonmonotone descent criterion.

A nonlinear VI defined with $f := \mathbb{R}^n \mapsto \mathbb{R}^n$ and $\mathcal{C} = \{z \,|\, Bz \ge b,\ Hz = h\}$ can be transformed into a complementarity problem as follows:

$$0 = f(z) - B^\top u - H^\top \lambda \quad \perp \quad z \text{ free} \tag{4.6.2}$$

$$0 \le Bz - b \quad \perp \quad u \ge 0 \tag{4.6.3}$$

$$0 = Hz - h \quad \perp \quad \lambda \text{ free .} \tag{4.6.4}$$

Therefore a conjecture for another choice of the merit function is the residual of the Fischer-Burmeister reformulation of this equivalent complementarity problem. Similar to the discussion in Section 3.1.1, we define

$$\Phi_i(z) := \begin{cases} f_i(z) - (B^\top)_{i\cdot}\, u - (H^\top)_{i\cdot}\, \lambda, & \text{for } i \in (4.6.2) \\ \phi(u_i, B_{i\cdot}\, z - b_i), & \text{for } i \in (4.6.3) \\ H_{i\cdot}\, z - h_i, & \text{for } i \in (4.6.4) \end{cases}$$

with

$$\phi(a,b) := \sqrt{a^2+b^2} - a - b.$$

The merit function is thus $\Psi(z) := \frac{1}{2}\|\Phi(z)\|^2$, which has the appeal of being continuously differentiable.

Another issue occurs with the watchdog steps, where we search for a point satisfying the nonmonotone descent criterion in between an earlier saved initial point and Newton point. Originally, we have the choice of line search and arc search. In this case, the arc search technique cannot be performed, since the projection of a point from $\mathbb{R}^n$ to the polyhedral set $\mathcal{C}$ cannot be computed easily. Hence line search may be our only choice.

Despite the above limitations, an algorithm with a generalized Newton method and a nonmonotone search scheme, combined with the new AVI solver can be adopted for solving nonlinear variational inequalities. Further investigations on the choices of merit functions and search schemes, and the global convergence property of the algorithm associated with these choices are interests of future research.

For certain classes of nonlinear variational inequalities, solving them through a sequence of affine variational inequalities can be more advantageous than solving their equivalent complementarity reformulation. Similar to what we have discussed before, the nonlinear variational inequality solver would always generates points that are feasible, while the complementarity solver searches for solutions in an augmented space with both primal and dual variables. This property of the variational inequality solver becomes even more important in nonlinear instances, since the complementarity solver may start from a point that is not well-defined for the original problem. For example, solving $VI(f, \mathcal{C})$ with :

$$f(z) := \begin{bmatrix} log(2+\delta-z_1) \\ log(2+\delta-z_2) \end{bmatrix} \quad \text{and} \quad \mathcal{C} = \{z \in \mathbb{R}^n \mid 1 \le z_1 + z_2 \le 2\}$$

is equivalent to solving the following nonlinear complementarity problem:

$$
\begin{aligned}
0 = \begin{bmatrix} log(2+\delta - z_1) \\ log(2+\delta - z_2) \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_1 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} u_2 \quad &\perp \quad \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \text{ free} \\
1 \le z_1 + z_2 \quad &\perp \quad u_1 \ge 0 \\
-2 \le -z_1 - z_2 \quad &\perp \quad u_2 \ge 0.
\end{aligned}
$$

In order to find a zero of their associated normal maps, both solvers will start by linearizing the normal map at an initial point x^0:

$$f_{\mathcal{C}}(x) = f(\pi_{\mathcal{C}}(x^0)) + \nabla f(\pi_{\mathcal{C}}(x^0))(\pi_{\mathcal{C}}(x) - \pi_{\mathcal{C}}(x^0)) + x - \pi_{\mathcal{C}}(x) = 0.$$

For any $x^0 \in \mathbb{R}^2$, the VI solver will only evaluate the function at $\pi_{\mathcal{C}}(x^0)$, which is always a well-defined point. In the case of the complementarity reformulation, the feasible set is $\mathbb{R}^2 \times \mathbb{R}^2_+$ (for z and u variables). For an initial choice of $\tilde{x}^0 \in \mathbb{R}^4$, the function will be evaluated at its projection point, which may not be well-defined. For example, suppose the initial point is $x^0 = (3,\ 3)$. In the VI problem over polyhedral set $\mathcal{C}$, this point will be projected onto $z^0 = (1,\ 1)$, which is well-defined for the *log* function. Correspondingly, an initial choice of $\tilde{x}^0 = (3,\ 3,\ 0,\ 0)$ for the complementarity problem will be projected onto itself, but is not well-defined for the *log* function anymore.

Chapter 5

A Complementarity Pivoting Method for Solving Parametric LCPs

Parametric Linear Complementarity Problems (PLCPs) are a special set of linear complementarity problems (LCPs) where the vector component generally denoted by q changes with a parameter, thus resulting in different solutions as the parameter value varies. Applications of PLCPs arise from diverse areas. Practical applications of PLCPs in portfolio selection, structural engineering and actuarial graduation problems are discussed in [62]. Theoretically, PLCP can also be used to analyze some well-known traffic paradoxes [12], arising from traffic equilibrium problem. Special-purpose algorithms have been designed for solving special classes of PLCPs [62] based on a generalization of existing algorithms such as Lemke's method ([57], [58]), Cottle and Dantzig [11] algorithm and Graves' [46] principal pivoting algorithm. A more general review of these PLCP schemes can also be found in [12]. In this chapter, we present an implementation of an extended Lemke's method which solves a PLCP for all feasible values of the parameter automatically (which is new to this thesis), together with some theoretical justification. Furthermore, with each fixed parameter, the corresponding LCP is equivalent to an affine variational inequality problem. For certain classes of PLCPs, it may be advantageous to consider solving the PLCP as a sequence of affine variational inequality problems with the PathAVI solver proposed earlier.

We first introduce the definition of parametric linear complementarity problem. Let M be a given square matrix of order n and let q, d be given column vectors in $\mathbb{R}^n$. Let Λ be a closed interval in $\mathbb{R}$. We consider the following parametric LCP: find a vector z satisfying

$$\begin{aligned} \mathbf{LCP}(M, q+\lambda d) : w &= Mz + q + \lambda d \\ z \geq 0,\ w \geq 0,\ z^\top w &= 0 \end{aligned} \tag{5.0.1}$$

as a function of λ, for $\lambda \in \Lambda$.

The scheme we are going to propose is based on Lemke's method, which is effective for solving linear complementarity problems, hence in Section 5.1, a brief review of Lemke's method for solving LCPs is presented. In Section 5.2, we will make some theoretical justification on the existence of the solution, using the method we propose. In Section 5.3, the outline of the scheme for solving parametric LCPs will be given, together with some theoretical support on the applicability of Lemke's method. A simple PLCP example will be given in Section 5.4. Several applications of PLCP on parametric Quadratic Programs (QP) and others are the subject of Section 5.5.

5.1 An Overview of Lemke's Method

The first algorithm proposed to solve linear complementarity problems was the famous pivotal algorithm of Lemke.

For positive semidefinite (PSD) M, Lemke's method generates a finite sequence of feasible, almost complementary pairs that terminates at a complementary pair or when there is an unbounded ray, the original LCP is determined infeasible. This result is stated in [11]:

Theorem 5.1.1. *If $M \in \mathbb{R}^{n\times n}$ is positive semidefinite, then for each $q \in \mathbb{R}^n$, Lemke's algorithm terminates at a solution of LCP(M,q) or at an unbounded ray. In the latter case, the set $\{z|Mz + q \geq 0, z \geq 0\}$ is empty, that is, there is no feasible pair.*

Proof. (Cottle & Dantzig 1968). □

The outline of Lemke's algorithm is as follows [37].

Algorithm 5.1 (Phase I: Generating a feasible almost-complementary tableau).

1. If $q \geq 0$, STOP: $z = 0$ is a solution of $LCP(M,q)$, that is, $(z,w) = (0,q)$ is a feasible complementary pair.

2. Otherwise, add the artificial variables z^0 and w^0 that are constrained to satisfy the following relationships:

$$w = Mz + ez_0 + q, \quad w_0 = z_0,$$

   where e is a vector of ones in $\mathbb{R}^n$. Create the initial tableau.

	z	z_0	1
w	M	e	q
w_0	0	1	0

3. Pivot row selection: Make this tableau feasible by carrying out a Jordan exchange on the z_0 column and the row corresponding to the most negative q_i. Without removing the artificial variables from the tableau, proceed to Phase II.

Algorithm 5.2 (Phase II: Generating a feasible complementary or unbounded tableau). Start with a feasible almost complementary pair (z, w) and the corresponding tableau in Jordan exchange form

$$\begin{array}{c|cc|c}
\hline
 & w_{I_1} & z_{J_2} & 1 \\
\hline
z_{J_1} & H_{I_1 J_1} & H_{I_1 J_2} & h_{I_1} \\
w_{I_2} & H_{I_2 J_1} & H_{I_2 J_2} & h_{I_2} \\
\hline
\end{array}$$

Take note of the variable that became nonbasic (i.e., became a column label) at the previous iteration. (At the first step, this is simply the component of w that was exchanged with z_0 during Phase I.)

1. Pivot column selection: Choose the column s corresponding to the complement of the variable that became nonbasic at the previous pivot.

2. Pivot row selection: Choose the row r such that

$$-h_r/H_{rs} = \min_i \{-h_i/H_{is} | H_{is} < 0\}.$$

   If all $H_{is} \geq 0$, STOP: An unbounded ray has been found.

3. Carry out a Jordan exchange on element H_{rs}. If (z, w) is complementary, STOP: (z, w) is a solution. Otherwise, go to Step 1.

5.2 Theoretical Justification

For a specific value of λ there may or may not exist a solution to the $LCP(M, q + \lambda d)$. We are going to restrict our attention to the values of λ, for which there exist solutions to the corresponding LCP.

A solution to the parametric $LCP(M, q + \lambda d)$ must satisfy the linear feasibility relations

$$q + \lambda d + Mz \geq 0,\ z \geq 0,\ \lambda \in \Lambda. \tag{5.2.1}$$

We define the set of z which satisfies the above conditions as $FES(M, q + \lambda d)$. It is possible to find the largest and smallest value of λ for which (5.2.1) has a solution by solving linear programs, and they are

$$\lambda_* = \inf\{\lambda \in \Lambda : FEA(M, q + \lambda d) \neq \emptyset\}, \tag{5.2.2}$$

$$\lambda^* = \sup\{\lambda \in \Lambda : FEA(M, q + \lambda d) \neq \emptyset\}, \tag{5.2.3}$$

and they can be used to refine the interval Λ. In general it is possible that either $\lambda_* = -\infty$ or $\lambda^* = \infty$, or both.

There is no point seeking solutions for λ values such that $FES(M, q + \lambda d) = \emptyset$. The following theorem indicates the existence of solutions to the parametric $LCP(M, q + \lambda d)$ for all finite number $\lambda \in [\lambda_*, \lambda^*]$.

Theorem 5.2.1. *Let* λ_* *and* λ^* *be defined as in* (5.2.2) *and* (5.2.3). *The parametric* $LCP(M, q + \lambda d)$ *has a solution for each finite number* $\lambda \in [\lambda_*, \lambda^*]$.

Proof. : We are given that for both λ_* and λ^*, $FES(M, q + \lambda d) \neq \emptyset$. Let z_*, z^* be the vectors in the feasible set corresponding to parameters λ_* and λ^*. We have

$$Mz_* + q + \lambda_* d \geq 0, z_* \geq 0$$

$$Mz^* + q + \lambda^* d \geq 0, z^* \geq 0$$

Now take any $\overline{\lambda} \in [\lambda_*, \lambda^*], \overline{\lambda}$ can be expressed as a linear combination of λ_* and λ^*,

$$\overline{\lambda} = a\lambda_* + (1 - a)\lambda^*, a \in [0, 1].$$

We first prove $FES(M, q + \overline{\lambda}d) \neq \emptyset$.

Let $\overline{z} = az_* + (1-a)z^*$. Since $z_*, z^* \geq 0, a \in [0,1]$, we have $\overline{z} \geq 0$.

$$\begin{aligned} M\overline{z} + q + \overline{\lambda}d &= aMz_* + (1-a)Mz^* + aq + (1-a)q + a\lambda_* d + (1-a)\lambda^* d \\ &= a(Mz_* + q + \lambda_* d) + (1-a)(Mz^* + q + \lambda^* d) \geq 0 \end{aligned}$$

Therefore $\overline{z} \in FES(M, q + \overline{\lambda}d)$. We have proven for any $\lambda \in [\lambda_*, \lambda^*]$, there exists a vector z which satisfies the feasibility condition. We also know that if M is positive semidefinite and $FES(M, q + \overline{\lambda}d) \neq \emptyset$, there exists a solution to $LCP(M, q + \lambda d)$. Hence the parametric $LCP(M, q + \lambda d)$ has a solution for each finite number $\lambda \in [\lambda_*, \lambda^*]$. □

5.3 Algorithm

5.3.1 Outline of the Algorithm

Step 1 Compute the lower and upper bounds for parameter λ that will generate feasible solutions by solving linear programs,

$$\begin{array}{ll} \min\limits_{\lambda,z}(\max\limits_{\lambda,z}) & \lambda \\ \text{subject to} & Mz + q + \lambda d \geq 0 \\ & z \geq 0, \lambda \in \Lambda. \end{array}$$

Let $\lambda = \lambda_*$.

Step 2 Phase I, this is very similar to Alg 5.1 (Phase I) of Lemke's method .

i. If $q + \lambda d \geq 0$, STOP: $z = 0$ is a solution of $LCP(M, q + \lambda d)$, that is, $(z, w) = (0, q + \lambda d)$ is a feasible complementary pair.

ii. Otherwise, add the artificial variables z^0 and w^0 that are constrained to satisfy the following relationships:

$$w = Mz + ez_0 + q + \lambda d, w_0 = z_0,$$

where e is the vector of ones in $\mathbb{R}^n$ and create the initial tableau

	z	z_0	1	λ
w	M	e	q	d
w_0	0	1	0	0

iii. Pivot row selection: Carry out a Jordan exchange on the z_0 column and row corresponding to the most negative $q_i + \lambda d_i$.

Step 3 Same as Alg. 5.2 (Phase II) of Lemke's method .

Step 4 Determine the range $[\lambda_L, \lambda_U]$ on which the current solution is feasible and complementary.

Step 5 If $\lambda_U \neq \lambda^*$, determine which component of $q + \lambda d$ becomes negative. Eliminate column z_0, and go to step 2. Otherwise, STOP, we have found all the feasible complementary pairs for $\lambda \in [\lambda_*, \lambda^*]$.

In the special case where $\lambda_* = -\infty$ and $\lambda^* < +\infty$, the algorithm starts from $\lambda = \lambda^*$ and subsequently decreases the value of λ until $-\infty$. If both λ_* and λ^* are infinite, the algorithm starts from the "middle" (at $\lambda = 0$) and first decreases λ until $-\infty$ then returns to the middle point and increases λ until $+\infty$.

The MATLAB code can be found in the Appendix.

5.3.2 Some Computation Considerations

In our algorithm, we choose the starting value of the parameter λ at the smallest feasible value λ_* and solve $LCP(M, q+\lambda_* d)$ by essentially applying a series of Jordan exchanges and column/row rearrangements on the original matrix M. The process terminates with a set of basic feasible complementary variables and correspondingly a new matrix M'. We then increase the value of λ until the present complementary solution no longer remains feasible. We use the new value λ', at which the solution just becomes infeasible, together with the current set of basic variables, to restart Lemke's method and solve a new $LCP(M', q + \lambda' d)$. We shall increase the value of λ and perform the above steps recursively until the value of λ reaches its upper bound λ^*. Therefore, each time the process restarts, it will start from the current cell defined by the active set associated with the final basis of the previous solve.

We know that Lemke's method works for positive semidefinite matrix M. In order to ensure that Lemke's method is applicable to our scheme, we need to prove that at the end of each solve (i.e. the start of the next LCP), the new transformed matrix M' remains positive semidefinite.

Theorem 5.3.1. *Let M be the matrix corresponding to the complementary basic variables denoted by $(w_1, w_2, ..., w_n)$, and z_j be the complement of w_j, for $j = 1, \ldots, n$. After a series of Jordan exchanges and column/row rearrangements, we obtain a new complementary set of basic variables $(y_1, y_2, ..., y_n)$, where $y_j \in \{w_j, z_j\}$ for $j = 1, \ldots, n$. Let M' be the matrix corresponding to the vector y. If M is positive semidefinite, M' is also positive semidefinite.*

Proof. Let $u = (u_1, ..., u_n)^T \in \mathbb{R}^n$. Define $v = (v_1, ..., v_n)^T$ by

$$v = Mu. \tag{5.3.1}$$

After a series of Jordan exchanges and rearrangements, we obtain a new vector ξ, with $\xi_j \in \{u_j, v_j\}$ for $j = 1...n$, and its corresponding complementary vector η, where $\eta_j = u_j$ if $\xi_j = v_j$ and $\eta_j = v_j$ if $\xi_j = u_j$. Since $v_j = M_{j.}u$, as u varies over $\mathbb{R}^n$, ξ also varies over $\mathbb{R}^n$. M', ξ and η are obtained from M, u and v by a series of Jordan exchanges and column, row rearrangements, for any u and v defined by (5.3.1),

$$\eta = M'\xi$$

also holds. Now,

$$u^T M u = u^T v = \xi^T \eta = \xi^T M' \xi.$$

These imply that $\xi^T M' \xi \geq 0$ for all $\xi \in \mathbb{R}^n$ iff $u^T M u \geq 0$ for all $u \in \mathbb{R}^n$. Hence M' is PSD iff M is PSD. $\square$

Theorem 5.3.1 justifies the fact that we are able to use Lemke's method to find a solution every time we restart the process with a new value of λ.

5.4 A Simple Example

We will use a simple $2-D$ example to illustrate the algorithm.

Consider $M = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$, $q = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$, $d = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

It's easy to check that M is PSD.

1. Compute the lower and upper bounds for λ. We can obtain these bounds using simply linear programming techniques and obtain $\lambda_* = -0.5$ and $\lambda^* = \infty$.

Starting from $\lambda = -0.5$, we shall find all the solutions to the parametric LCP corresponding to the values of $\lambda \in [-0.5, \infty]$.

2. Apply Phase I of Lemke's method. The initial tableau is the following.

	z_1	z_2	z_0	1	λ
w_1	1	-1	1	-1	1
w_2	-1	1	1	2	1

In practice we don't have to put row w_0 in the tableau. At $\lambda = -0.5$, we first pivot on the z_0 column, and the first row.

	z_1	z_2	w_1	1	λ
z_0	-1	1	1	1	-1
w_2	-2	2	1	3	0

3. Apply Phase II of Lemke's method and obtain the first complementarity solution $z = [1.5 \quad 0]^\top$, with basic variables $[z_1 \quad w_2]^\top = [1 - \lambda \quad 1 + 2\lambda]^\top$.

	w_1	z_2	1	λ
z_1	1	1	1	-1
w_2	-1	0	1	2

The solution is feasible and complementary for $\lambda \in [-0.5, 1]$. When λ exceeds 1, the complementary pair are no longer feasible.

Note that the new square matrix remains PSD. Repeat step 2 and 3 with $\lambda \geq 1$, we obtain a new complementary solution, that is $z = [0\ 0]^\top$, with basic variables $[w_1 \quad w_2]^\top = [-1 + \lambda \ \ 2 + \lambda]^\top$.

	z_1	z_2	1	λ
w_1	1	-1	-1	1
w_2	-1	1	2	1

This solution is feasible for all $\lambda \geq 1$. We have thus found all the complementary solutions for all the feasible values of λ.

5.5 Application to Parametric Convex Quadratic Programming

An example of PLCP arises from solving the parametric convex quadratic programs (QP). Parametric QP is defined as:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^\top Dx + (c^1 + \lambda c^2)^\top x \\ s.t. \quad & Ax \geq b^1 + \lambda b^2 \\ & x \geq 0 \end{aligned} \tag{5.5.1}$$

where D is a symmetric positive semidefinite matrix of order n and λ belongs to a closed interval in $\mathbb{R}$ similar to the case of PLCP. If either c^2 or b^2 is equal to 0, we have a special case where the parameter only appears in the right hand side constant vector, or the linear term of the objective function respectively.

Since D is positive semidefinite, the KKT point of (5.5.1) corresponds to its optimal solution. The KKT conditions can be written as a parametric LCP$(M, q + \lambda d)$ with:

$$M = \begin{bmatrix} D & -A^\top \\ A & 0 \end{bmatrix}, \quad q = \begin{bmatrix} c^1 \\ -b^1 \end{bmatrix}, \quad d = \begin{bmatrix} c^2 \\ -b^2 \end{bmatrix}.$$

Note that the matrix M constructed in this way is also a positive semidefinite matrix. Therefore the PLCP can be solved by the scheme proposed before.

A well-known application of parametric QP arises from portfolio selection problem [62]: let $x = (x_1, \cdots, x_n)^\top$ be the proportions of the investor's wealth invested in n

securities, which sum to 1. Let the security returns be represented by a vector of random variables $R = (R_1, \cdots, R_n)^\top$ with mean $\mu = (\mu_1, \cdots, \mu_n)^\top$ and covariance matrix $V = (\sigma_{ij})_{i,j=1}^n$ where $\sigma_{ij} = Cov(R_i, R_j)$. The portfolio selection problem is to find the value of x that will

$$\begin{aligned} \min \quad & \frac{1}{2} x^\top V x - \theta \mu^\top x \\ s.t. \quad & 0 \le x \le a \\ & e^\top x = 1, \end{aligned}$$

where e is a vector of all ones, a is a given positive vector denoting the upper bounds on the proportions to be invested in the securities, and θ is the coefficient of risk aversion. By varying θ from 0 to ∞, all efficient portfolios can be determined. The upper bounding constraints exist for legal, personal or institutional reasons. Some component of the vector a can be infinite. The covariance matrix V is symmetric positive semidefinite under certain assumptions. The KKT conditions lead to the following PLCP:

$$\begin{aligned} 0 \le Vx - \theta\mu - ue + v \quad & \perp \quad x \ge 0 \\ e^\top x = 1 \quad & \perp \quad u \textit{ free} \\ x \le a \quad & \perp \quad v \ge 0. \end{aligned} \tag{5.5.2}$$

The matrix in LCP$(M, q + \lambda d)$ thus has

$$M = \left[\begin{array}{c|cc} V & -e & \mathbb{I} \\ \hline e^\top & & \\ -\mathbb{I} & & \end{array} \right].$$

In the above case, for each fixed parameter value, the PLCP (5.5.2) is equivalent to an affine variational inequality problem, which can be written as:

$$0 \in Vx - \theta\mu + N_{\mathcal{C}}(x)$$

with $\mathcal{C} = \{x \,|\, 0 \leq x \leq a,\ e^\top x = 1\}$. To extend this result, a parametric QP in the following form:

$$\begin{aligned} \min_x \quad & \frac{1}{2}x^\top Dx + (c^1 + \lambda c^2)^\top x \\ s.t. \quad & x \in \text{ polyhedral set } \mathcal{C}, \end{aligned}$$

can be solved by a sequence of affine variational inequalities in a similar fashion as the PLCP case. In particular, for a given feasible range of $\lambda \in [\lambda_*, \lambda^*]$, start by solving an $AVI(D, -(c^1 + \lambda c^2), \mathcal{C})$ with $\lambda = \lambda_*$. At the solution, compute the range for λ which preserves the feasibility of the current AVI system. As λ exceeds this range, restart the Main Algorithm of AVI in the current cell and find a new AVI solution corresponding to a new range of λ and repeat, until λ^* is reached. Similar to our previous discussion in Chapter 4, solving a parametric QP over a bounded polyhedral constraint set (as in the case of the portfolio selection problem), by parameterized AVI is potentially more effective than by PLCP.

Another application of PLCP is solving multiple criteria problems with one quadratic objective, several linear objectives, and a set of linear constraints. A reference direction approach is employed for such problems and a PLCP formulation of the problem is thus developed [55]. Extended portfolio selection problems and noise reduction design problems are typical examples for such problems in practice. The problem is defined as:

$$\begin{aligned} \min \quad & V(x) = \frac{1}{2}x^\top Dx \\ \max \quad & l(x) = Cx \\ \text{Subject to:} \quad & Ax \leq b \\ & x \geq 0, \end{aligned} \tag{5.5.3}$$

where D is a symmetric positive semidefinite matrix, C is a $k \times n$ matrix containing the coefficients of the linear objective functions. All linear objective functions need to be maximized simultaneously.

By ignoring the quadratic objective, the rest multiple objective linear program can be approached by investigating the following problem:

$$\begin{aligned} \min \quad & \epsilon \\ \text{Subject to:} \quad & Cx + \epsilon w \geq g \qquad (5.5.4) \\ & x \in X := \{x \,|\, Ax \leq b,\ x \geq 0\}, \qquad (5.5.5) \end{aligned}$$

with an aspiration level vector g and a weighting vector w. These parameters are provided initially and will be updated iteratively.

Combining the quadratic objective with the above using a weighted sum, the problem in (5.5.3) becomes:

$$\begin{aligned} \min \quad & \mu \frac{1}{2} x^\top D x + (1-\mu)\epsilon \\ \text{Subject to:} \quad & Cx + \epsilon w \geq g \qquad (5.5.6) \\ & x \in X, \mu \in (0,1). \end{aligned}$$

Furthermore, dividing the objective function of problem (5.5.6) by μ and letting $t = \frac{1}{\mu} - 1$ ($t > 0$) transform the above problem (5.5.6) into the following:

$$\begin{aligned} \min \quad & \frac{1}{2} x^\top D x + t\epsilon \\ \text{Subject to:} \quad & Cx + \epsilon w \geq g \qquad (5.5.7) \\ & x \in X. \end{aligned}$$

Reference directions Δt and Δg ($t + \lambda \Delta t > 0$) are used to parameterize the vectors in system (5.5.7), in order to search for new solutions, namely,

$$\begin{aligned}
\min \quad & \frac{1}{2}x^\top Dx + (t+\lambda\Delta t)\epsilon \\
\text{Subject to:} \quad & Cx + \epsilon w \geq g + \lambda\Delta g \qquad (5.5.8) \\
& Ax \leq b, x \geq 0,
\end{aligned}$$

and then they are used to update t and g respectively.

Now the system becomes a parameterized QP and can be solved via a standard PLCP, that is

$$\text{LCP}\left(\begin{bmatrix} D & & -C^\top & A^\top \\ & & -w^\top & \\ C & w & & \\ -A & & & \end{bmatrix}, \begin{bmatrix} 0 \\ t \\ -g \\ b \end{bmatrix} + \lambda \begin{bmatrix} 0 \\ \Delta t \\ -\Delta g \\ 0 \end{bmatrix}\right).$$

5.6 Summary

We have implemented a MATLAB code based on Lemke's Method to compute PLCPs automatically for all parameter values of interest. In order to complete the code, effective techniques for testing whether the matrix in the PLCP is PSD and for breaking possible ties after updating the parameter value need to be considered. In real applications, we may not always have PSD matrices in PLCPs. For solving such problems, we hope to enhance our scheme by incorporating some alternative LCP algorithms which deal with a wider variety of matrix classes, for example the PATH solver. Another alternative is to solve a PLCP as a sequence of equivalent AVIs. Similar to what we already discussed before in Chapter 4, for a certain class of PLCPs, this alternative may be more efficient. Furthermore, the AVI algorithm proposed in Chapter 4 can process PLCPs with less restrictive matrices, for example copositive-plus matrices.

Appendix A

Code for Solving Parametric LCPs

A.1 Main Code

```
% syntax: [s,err] = plemke(M,q,d,k_lo,k_up)
% PLEMKE - Solves parametric linear complementarity problems (PLCPs).
% A PLCP solves
%    Mz + q + kd >= 0, z >= 0, z'(Mz + q + kd) = 0.
%    k is the parameter varying from k_lo to k_up
% if k_lo or k_up or both are omitted, the default range is (-inf,inf).
%
% ERR returns an error condition:
%   0: Solution found
%   1: Maximum iterations exceeded
%   2: Unbounded ray termination
% If NARGOUT==1, a warning message is displayed instead.
%
% The algorithm determines the feasible range of parameter k
% then finds LCP solutions for all feasible values of k.
%

function [s,err] = plemke(M,q,d,k_lo,k_up)

n = length(q);
```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%
% Construct LP to decide feasible range for the parameter%
% Determine   k \in [lk, uk]                             %
%        s.t. Mz + q + kd >= 0, z >= 0                   %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%
A = [M d];
p = [zeros(n,1);1];
ub = inf*ones(n+1,1);
lb = [zeros(n,1);-inf];
ge = find(d<inf);

% Compute the lower bound on parameter
%          min  k
%          s.t. Mz + q + kd >= 0, z >= 0

[obj,x,lambda,status] = cplexlp(p,A,-q,lb,ub,[],ge);
if status == 1
    if nargin < 4
        lk = x(n+1);
    else
        lk = max(x(n+1),k_lo);
    end
elseif status == 2 % problem is unbounded
    if nargin < 4
        lk = -inf;
    else
        lk = max(-inf,k_lo);
    end
elseif status == 3
```

```
    error('problem is infeasible');
elseif status == 4   % problem infeasible or unbounded
    % recompute lp with zero objective
    [obj,x,lambda,status] = cplexlp(zeros(n+1,1),A,-q,lb,ub,[],ge);
    if status == 1 % problem is feasible therefore is unbounded
        if nargin < 4
            lk = -inf;
        else
            lk = max(-inf,k_lo);
        end
    else           % problem is infeasible
        error('problem is infeasible');
    end
else
    error('problem is non-optimal');
end

% Compute the upper bound on parameter
%          max  k
%          s.t. Mz + q + kd >= 0, z >= 0

[obj,x,lambda,status] = cplexlp(-p,A,-q,lb,ub,[],ge);
if status == 1
    if nargin < 5
        uk = x(n+1);
    else
        uk = min(x(n+1),k_up);
    end
elseif status == 2 % problem is unbounded
```

```
    if nargin < 5
        uk = inf;
    else
        uk = min(inf,k_up);
    end
elseif status == 3
    error('problem is infeasible');
elseif status == 4   % problem infeasible or unbounded
    % recompute lp with zero objective
    [obj,x,lambda,status] = cplexlp(zeros(n+1,1),A,-q,lb,ub,[],ge);
    if status == 1 % problem is feasible therefore is unbounded
        if nargin < 5
            uk = inf;
        else
            uk = min(inf,k_up);
        end
    else           % problem is infeasible
        error('problem is infeasible');
    end
else
    error('problem is non-optimal');
end

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
% Decide the starting value of parameter k                %
% Start from a finite bound or zero                       %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %
flag = 1;
sign = 0;
```

```
k = 0;

lk = max(lk,-inf);
uk = min(uk,inf);

if lk > -inf
  k = lk;
  sign = 1;
elseif uk < inf
    k = uk;
    sign = -1;
end

z = zeros(2*n,2);

% Determine initial basis
bas1=(n+1:2*n)';
B1 = -speye(n);
x1 = [q d];

% Check if initial basis provides solution
if all(x1(:,1) + k*x1(:,2)>0)
  z(bas1,:)=x1;
else
  % Find a solution with the first value of the parameter
  [x1,B1,bas1,err] = mainlemke(x1,k,M,B1,bas1);
  if nargout<2 && err(1)~=0
      ss='Warning: solution not found';
      fprintf(' for parameter k at %6.2f  -  ',k);
      if err(1)==2
          disp([ss 'Unbounded ray']);
      elseif err(1)==1
          disp([ss 'Iterations exceeded limit']);
      end
      return;
  end
  z(bas1,:) = x1;
end

numsol = 1;
% Check if given upper and lower bound are feasible
if (sign > 0)     % check lower bound
    if (nargin >= 4 && lk > k_lo) % given lower range is not feasible
        s(numsol) = struct('solution',[],'range',[k_lo,lk]);
        numsol = numsol + 1;
    end
elseif (sign < 0) % check upper bound
    if (nargin >= 5 && uk < k_up) % given lower range is not feasible
        s(numsol) = struct('solution',[],'range',[uk,k_up]);
        numsol = numsol + 1;
    end
end
s(numsol) = struct('solution',z(1:n,:),'range',[]);

% Find solutions to other values of the parameter

if sign ~= 0        % k starts from a finite bound
  x = x1; B = B1; bas = bas1;
```

```
while flag
    z = zeros(2*n,2);
    critical = -x(:,1)./x(:,2); % Find the next critical value of k
    cand = find(sign*critical > sign*k && critical >= lk
                                          && critical <= uk);
    if ~isempty(cand)
        k1 = sign*min(sign*critical(cand));
        s(numsol).range = [min(k,k1),max(k,k1)];
        % Call mainlemke to found the solution at k1
        [x,B,bas,err] = mainlemke(x,k1,M,B,bas);
        if nargout<2 && err(1)~=0
            ss='Warning: solution not found';
            fprintf(' for parameter k at %6.2f  -  ',k1);
            if err(1)==2
                disp([ss 'Unbounded ray']);
            elseif err(1)==1
                disp([ss 'Iterations exceeded limit']);
            end
            return;
        end
        z(bas,:) = x;
        numsol = numsol + 1;
        s(numsol) = struct('solution',z(1:n,:),'range',[]);
        k = k1;
    else          % no more critical k in [lk, uk]
        k1 = (sign+1)/2 * uk + (sign-1)/2 * lk;
        s(numsol).range = [min(k,k1),max(k,k1)];
        if (sign < 0)     % check lower bound
            if (nargin >= 4 && lk > k_lo)
```

```
                % given lower range is not feasible
                numsol = numsol + 1;
                s(numsol) = struct('solution',[],'range',[k_lo,lk]);
            end
        elseif (sign > 0) % check upper bound
            if (nargin >= 5 && uk < k_up)
                % given lower range is not feasible
                numsol = numsol + 1;
                s(numsol) = struct('solution',[],'range',[uk,k_up]);
            end
        end
        disp('solution for each feasible range of the parameter');
        for i = 1:numsol
            fprintf('\n range of parameter k :
                [ %d, %d ]\n solution : \n',s(i).range);
            fprintf(' [%6.2f] + k *[%6.2f]\n',
                s(i).solution(:,1), s(i).solution(:,2));
        end
        break;
    end
   end
else           % k in (-inf,inf)
  sign = -1;   % k starts from zero and decrease first
  x = x1; B = B1; bas = bas1;
  while flag
    z = zeros(2*n,2);
    critical = -x(:,1)./x(:,2); % Find the next critical value of k
    cand = find(sign*critical > sign*k && critical >= lk
                                          && critical <= uk);
    if ~isempty(cand)
        k1 = sign*min(sign*critical(cand));
        s(numsol).range = [k1,k];
        % Call mainlemke to found the solution.
        [x,B,bas,err] = mainlemke(x,k1,M,B,bas);
        if nargout<2 && err(1)~=0
            ss='Warning: solution not found';
            fprintf(' for parameter k at %6.2f  -  ',k1);
            if err(1)==2
                disp([ss 'Unbounded ray']);
            elseif err(1)==1
                disp([ss 'Iterations exceeded limit']);
            end
            return;
        end
        z(bas,:) = x;
        numsol = numsol + 1;
        s(numsol) = struct('solution',z(1:n,:),'range',[]);
        k = k1;
    else
        s(numsol).range = [lk,k];
        if (nargin >= 4 && lk > k_lo)
            % given lower range is not feasible
            numsol = numsol + 1;
            s(numsol) = struct('solution',[],'range',[k_lo,lk]);
        end
        break;
    end
end
```

```
sign = 1;  % k starts from zero and increase
k = 0;
x = x1; B = B1; bas = bas1;
while flag
    z = zeros(2*n,2);
    critical = -x(:,1)./x(:,2); % Find the next critical value of k
    cand = find(sign*critical > sign*k && critical >= lk
                                          && critical <= uk);
    if ~isempty(cand)
        k1 = sign*min(sign*critical(cand));
        if k == 0
            k = s(1).range;
            s(1).range = [k(1),k1];
        else
            s(numsol).range = [k,k1];
        end
        % Call mainlemke to found the solution.
        [x,B,bas,err] = mainlemke(x,k1,M,B,bas);
        if nargout<2 && err(1)~=0
            ss='Warning: solution not found';
            fprintf(' for parameter k at %6.2f  -  ',k1);
            if err(1)==2
                disp([ss 'Unbounded ray']);
            elseif err(1)==1
                disp([ss 'Iterations exceeded limit']);
            end
            return;
        end
        z(bas,:) = x;
```

```
            numsol = numsol + 1;
            s(numsol) = struct('solution',z(1:n,:),'range',[]);
            k = k1;
        else
            s(numsol).range = [k,uk];
            if (nargin >= 5 && uk < k_up)
                % given lower range is not feasible
                numsol = numsol + 1;
                s(numsol) = struct('solution',[],'range',[uk,k_up]);
            end
            disp('solution for each feasible range of the parameter');
            for i = 1:numsol
                fprintf('\n range of parameter k :
                    [ %d, %d ]\n solution : \n',s(i).range);
                fprintf(' [%6.2f] + k *[%6.2f]\n',
                    s(i).solution(:,1), s(i).solution(:,2));
            end
            break;
        end
    end
end
```

A.2 Subroutine for Performing a Modified Lemke's Pivotal Scheme

For fixed value of λ (or k as in the the code), compute the solution for the linear complementarity problem.

```
% Subroutine for finding a solution to the LCP
% corresponding to a fixed value of parameter k.
% Similar to the Lemke's method, except pivoting on
% q and d vector in Tableau separately

function [x,B,bas,err] = mainlemke(x,k,M,B,bas)

n = length(x(:,1));
zer_tol = 1e-5;
piv_tol = 1e-8;
maxiter = min([1000 25*n]);

t = 2*n+1;      % Artificial variable
entering=t;     % is the first entering variable

% Determine initial leaving variable
[tval,lvindex]=max(-(x(:,1) + k*x(:,2)));
y = -x(lvindex,:);
leaving=bas(lvindex);
err=0;

bas(lvindex)=t;       % pivot in the artificial variable
x(:,1)=x(:,1)-x(lvindex,1);
x(:,2)=x(:,2)-x(lvindex,2);
x(lvindex,:)=y;
B(:,lvindex)=-B*ones(n,1);

% Main iterations begin here
for iter=1:maxiter
```

```
% Check if done; if not, get new entering variable
if (leaving == t) break
elseif (leaving <= n)
  entering = n+leaving;
  Be = sparse(leaving,1,-1.0,n,1);
else
  entering = leaving-n;
  Be = M(:,entering);
end
d = B\Be;

% Find new leaving variable
j=find(d>piv_tol);          % indices of d>0
if isempty(j)               % no new pivots - ray termination
  err=2;
  break
end
theta=min((x(j,1)+k*x(j,2)+zer_tol)./d(j)); % minimal ratios, d>0
% indices of minimal ratios, d>0
j=j(find(((x(j,1)+k*x(j,2))./d(j))<=theta));
lvindex=find(bas(j)==t);    % check if artificial among these
if ~isempty(lvindex)        % Always use artifical if possible
  lvindex=j(lvindex);
else                        % otherwise pick among set of max d
  theta=max(d(j));
  lvindex=find(d(j)==theta);
  % if multiple choose randomly
  lvindex=j(ceil(length(lvindex)*rand));
end
```

```
  leaving=bas(lvindex);

  % Perform pivot
  ratio=x(lvindex,:)./d(lvindex);
  x(:,1) = x(:,1) - ratio(1,1)*d;
  x(:,2) = x(:,2) - ratio(1,2)*d;
  x(lvindex,:) = ratio;
  B(:,lvindex) = Be;
  bas(lvindex) = entering;
end                                    % end of iterations
if iter>=maxiter & leaving~=t err=1; end

% z(bas,:) = x;
% z = z(1:n,:);

% Display warning messages if no error code is returned
if nargout<4 & err(1)~=0
  s='Warning: solution not found - ';
  if err(1)==2
    disp([s 'Unbounded ray']);
  elseif err(1)==1
    disp([f 'Iterations exceeded limit']);
  end
end

return;
```

Bibliography

[1] R. H. Bartels and G. H. Golub, *The simplex method of linear programming using the lu decomposition*, Communications of the ACM **12** (1969), 266–268.

[2] S. C. Billups, S. P. Dirkse, and M. C. Ferris, *A comparison of large scale mixed complementarity problem solvers*, Computational Optimization and Applications **7** (1997), 3–25.

[3] A. Brooke, D. Kendrick, and A. Meerhaus, *Gams: A user's guide*, The Scientific Press, South San Francisco, CA, 1988.

[4] M. Cao and M. C. Ferris, *Lineality removal for copositive-plus normal maps*, Communications on Applied Nonlinear Analysis **2** (1995), 1–10.

[5] ———, *A pivotal method for affine variational inequalities*, Mathematics of Operations Research **21** (1996), 44–64.

[6] R. M. Chamberlain, M. J. D. Powell, C. Lemarechal, and H. C. Pedersen, *The watchdog technique for forcing convergence in algorithms for constrained optimization*, Math Program. **16** (1982), 1–17.

[7] X. Chen and X. Deng, *3-nash is ppad-complete*, vol. 134, Electronic Colloquium on Computational Complexity, 2005.

[8] ———, *Settling the complexity of two-player nash equilibrium*, 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 261–272.

[9] A. K. Cline, *Two subroutine packages for the efficient updating of matrix factorizations*, Tech. Report Report TR-68, Department of Computer Sciences, The University of Texas, Austin, TX, 1977.

[10] R. W. Cottle, *Complementarity and variational problems*, Symposia Mathematica **19** (1976), 177–208.

[11] R. W. Cottle and G. B. Dantzig, *Complementary pivot theory of mathematical programming*, Linear Algebra and its Applications **1** (1968), 103–125.

[12] R. W. Cottle, J. S. Pang, and R. E. Stone, *The linear complementarity problem*, Academic Press, Boston, 1992.

[13] J. Czyzyk, M. P. Mesnier, and J. J. More, *The network-enabled optimization server*, Preprint MCS-P615-0996, Argonne National Laboratory, Argonne, Illinois, 1996.

[14] G. B. Dantzig, *Linear programming and extensions*, Princeton Universtiy Press, Princeton, NJ, 1963.

[15] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a nash equilibrium*, 38th ACM Symposium on Theory of Computing, 2005, pp. 71–78.

[16] C. Daskalakis and C. H. Papadimitriou, *Three-player games are hard*, vol. 139, Electronic Colloquium on Computational Complexity, 2005.

[17] T. A. Davis, *Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software **30** (2004), no. 2, 196–199.

[18] ———, *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software **30** (2004), no. 2, 165–195.

[19] T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse lu factorization*, SIAM Journal on Matrix Analysis and Applications **18** (1997), no. 1, 140–158.

[20] ———, *A combined unifrontal/multifrontal method for unsymmetric sparse matrices*, ACM Transactions on Mathematical Software **25** (1999), no. 1, 1–19.

[21] S. P. Dirkse and M. C. Ferris, *MCPLIB: A collection of nonlinear mixed complementarity problems*, Optimization Methods and Software **5** (1995), 319–345.

[22] ———, *The path solver: A non-monotone stabilization scheme for mixed complementarity problems*, Optimization Methods and Software **5** (1995), 123–156.

[23] ———, *Crash techniques for large-scale complementarity problems*, Complementarity and Variational Problems: State of the Art (M. C. Ferris and J. S. Pang, eds.), SIAM, Philadelphia, Pennsylvania, 1997, pp. 40–61.

[24] B. C. Eaves, *On the basic theorem of complementarity*, Mathematical Programming **1** (1971), 68–87.

[25] ———, *A short course in solving equations with pl homotopies*, Nonlinear Programming (Providence, RI) (R. W. Cottle and C. E. Lemke, eds.), American Mathematical Society, SIAM-AMS Proceedings, 1976, pp. 73–143.

[26] S. K. Eldersveld and M. A. Saunders, *A block-lu update for large-scale linear programming*, SIAM J. Matrix Anal. Appl. **13** (1992), no. 1, 191–201.

[27] R. E. Ericson and A. Pakes, *Markov-perfect industry dynamics: A framework for empirical work*, Rev. Econ. Stud. **62** (1995), 53–82.

[28] F. Facchinei and J. S. Pang, *Finite dimensional variational inequalities and complementarity problems*, Springer Series in Operations Research, vol. 1, Berlin-Heidelberg-New York: Springer-Verlag, 2003.

[29] ———, *Finite dimensional variational inequalities and complementarity problems*, Springer Series in Operations Research, vol. 2, Finite Dimensional Variational Inequalities and Complementarity Problems, 2003.

[30] M. C. Ferris, C. Kanzow, and T. S. Munson, *Feasible descent algorithms for mixed complementarity problems*, Mathematical Programming **86** (1999), 475–497.

[31] M. C. Ferris and S. Lucidi, *Nonmonotone stabilization methods for nonlinear equations*, J. Optimiz. Theory .App. **81** (1994), 53–71.

[32] M. C. Ferris, O. L. Mangasarian, and S. J. Wright, *Linear programming with matlab*, MPS-SIAM series on optimization, Society for Industrial Mathematics, 2008.

[33] M. C. Ferris, M. P. Mesnier, and J. More, *Neos and condor: Solving nonlinear optimization problems over the internet*, ACM Transactions on Mathematical Software **26** (2000), 1–18.

[34] M. C. Ferris and T. S. Munson, *Interfaces to path 3.0: Design, implementation and usage*, Computational Optimization and Applications **12** (1999), 207–227.

[35] ______, *Preprocessing complementarity problems*, Complementarity: Applications, Algorithms and Extensions, Volume 50 of Applied Optimization (M. C. Ferris, O. L. Mangasarian, and J. S. Pang, eds.), Kluwer Academic Publishers, 2001, pp. 143–164.

[36] M. C. Ferris and J. S. Pang, *Engineering and economic applications of complementarity problems*, SIAM Review **39** (1997), 669–713.

[37] Michael C. Ferris, O. L. Mangasarian, and Stephen J. Wright, *Linear programming via matlab*, Handout for Linear Programming Course 525 provided in CS WU-Madison.

[38] A. Fischer, *A special newton-type optimization method*, Optim. **24** (1992), 269–284.

[39] R. Fletcher and S. P. J. Matthews, *Stable modifications of explicit lu factors for simplex updates*, Mathematical Programming **30** (1984), 267–284.

[40] J. J. H. Forrest and J. A. Tomlin, *Updated triangular factors of the basis to maintain sparsity in the product form simplex method*, Math Program. **2** (1972), 263–278.

[41] R. Fourer, D. M. Gay, and B. W. Kernighan, *A modeling language for mathematical programming*, Duxbury Press, 2002.

[42] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Sparse matrix methods in optimization*, SIAM J. Sci. and Statist. Comput. **5** (1984), 562–589.

[43] P. E. Gill, W. Murray, and M. J. Wright, *Numerical linear algebra and optimization*, vol. 1, A. M. Wylde, Redwood City, CA, 1991.

[44] P. R. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *Maintaining lu factors of a general sparse matrix*, Linear Algebra and its Applications **88/89** (1987), 239–270.

[45] G. H. Golub and C. F. Van Loan, *Matrix computations*, The John Hopkins University Press, Baltimore, MD, 1996.

[46] R. L. Graves, *A principal pivoting simplex algorithm for linear and quadratic programming*, Operations Research **15** (1967), 482–494.

[47] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for newton's method*, SIAM J. Numer. Anal. **23** (1986), 707–716.

[48] ______, *A class of nonmonotone stabilization methods in unconstrained optimization*, Numer. Math **59** (1991), 779–805.

[49] P. T. Harker and J. S. Pang, *Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications*, Mathematical Programming, Series B **48** (1990), 161–220.

[50] H. M. Huynh, *A large-scale quadratic programming solver based on block-lu updates of the kkt system*, Ph.D. thesis, Stanford University, 2008.

[51] N. H. Josephy, *A newton method for the pies energy model*, Technical Summary Report 1971, Mathematics Research Center, University of Wisconsin-Madison, Madison, Wisconsin, 1979.

[52] ______, *Newton's method for generalized equations*, Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin-Madison, Madison, Wisconsin, 1979.

[53] ______, *Newton's method for generalized equations and the pies energy model*, Ph.D. thesis, Department of Industrial Engineering, University of Wisconsin-Madison, 1979.

[54] W. Karush, *Minima of functions of several variables with inequalities as side constraints*, Master's thesis, Univ. of Chicago, 1939.

[55] P. Korhonen and G. Y. Yu, *A reference direction approach to multiple objective quadratic-linear programming*, European Journal of Operational Research **102** (1997), 601–610.

[56] H. W. Kuhn and A. W. Tucker, *Nonlinear programming*, Proceedings of 2nd Berkeley Symposium on Mathematical Statistics and Probability, Berkeley: University of California Press, 1951, pp. 481–492.

[57] C. Lemke and J. Howson, *Equilibrium points of bimatrix games*, Journal of the Society for Industrial and Applied Mathematics **12** (1964), 413–423.

[58] C. E. Lemke, *Bimatrix equilibrium points and mathematical programming*, Management Science **11** (1965), 681–689.

[59] R. Lougee-Heimer, *The common optimization interface for operations research*, IBM J. Res. Dev. **47** (2003), no. 1, 57–66.

[60] MATLAB, *User's guide*, The MathWorks, Inc., 1992.

[61] A. Pakes and P. McGuire, *Computing markov-perfect nash equilibria: Numerical implications of a dynamic differentiated product model*, Rand J. Econ. **25** (1994), 555–589.

[62] J. S. Pang, I. Kaneko, and W. P. Hallman, *On the solution of some (parametric) linear complementarity problems with applications to portfolio analysis, structural engineering and graduation*, Mathematical Programming **16** (1979), no. 1, 325–347.

[63] D. Ralph, *Global convergence of damped newton's method for nonsmooth equations via the path search*, Math Oper. Res. **19** (1994), 352–389.

[64] J. K. Reid, *Fortran subroutines for handling sparse linear programming bases*, Tech. Report AERE R8269, Atomic Energy Research Establishment, Harwell, England, 1976.

[65] ______, *A sparsity-exploiting variant of the bartels-golub decomposition for linear programming bases*, Mathematical Programming **24** (1982), 55–69.

[66] S. M. Robinson, *Normal maps induced by linear transformations*, Mathematics of Operations Research **17** (1992), no. 3, 691–714.

[67] ______, *A reduction method for variational inequalities*, Mathematical Programming **80** (1998), 161–169.

[68] M. A. Saunders, *LUMOD*, Fortran software for updating dense LU factors, http://www.stanford.edu/group/SOL/software/lumod.html, 2000.

[69] F. Tin-Loi and M. C. Ferris, *Holonomic analysis of quasibrittle fracture with non-linear softening*, Advances in Fracture Research (M. I. Ripley B. L. Karihaloo, Y. W. Mai and R. O. Ritchie, eds.), vol. 2, Pergamon Press, Oxford, 1997, pp. 2183–2190.

[70] P. Wolfe, *The reduced-gradient method*, unpublished manuscript, 1962, RAND Corporation.