ALGORITHMS AND INTERFACES FOR STRUCTURED VARIATIONAL INEQUALITIES AND THEIR EXTENSIONS

by

Youngdae Kim

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2017

Date of final oral examination: 10/17/2017

The dissertation is approved by the following members of the Final Oral Committee:

Michael C. Ferris, Professor, Computer Sciences

Stephen M. Robinson, Professor, Industrial and Systems Engineering

Stephen J. Wright, Professor, Computer Sciences

Thomas F. Rutherford, Professor, Agricultural and Applied Economics

Alberto Del Pia, Assistant Professor, Industrial and Systems Engineering

Abstract

Variational inequalities (VIs) are a generalization of nonlinear system of equations, so-called generalized equations. In addition to the system of equations, they subsume geometric first-order optimality conditions, nonlinear (linear) complementarity problems, and mixed complementarity problems. Representative applications are equilibrium problems such as generalized Nash equilibrium problems (GNEPs) and multiple optimization problems with equilibrium constraints (MOPECs).

This thesis is concerned with algorithms and interfaces for structured variational inequalities and their extensions. Algorithms and interfaces are closely related to each other in a way that interfaces helping identify problem structures can lead to more robust and efficient algorithms, and structure-exploiting algorithms can guide us to design better structure-exposing interfaces.

Interfaces exposing problem structures are described based on an extended mathematical programming (EMP) framework, where the framework allows us to formulate equilibrium problems in a natural and intuitive way in modeling languages, for example AMPL, GAMS, or Julia, without requiring the modeler to supply derivatives. Extensions to support some complicated structures such as shared constraints, shared variables, and quasi-variational

inequalities (QVIs) are presented. Our interfaces generate a human-readable file from which we can easily identify high-level structure of the problem.

We present an extension to PATH and two general-purpose solvers, PATHAVI and SELKIE, each of which utilizes problem structures, such as implicitly defined variables, polyhedral constraints, and groups of interacting agents, respectively. These structures are identified through our interfaces. An extension to PATH exploits implicitly defined variables by restoring their feasibility via projection using the implicit function theorem. Projection is performed in both their primal and dual spaces. PATHAVI is a structure-preserving solver for affine variational inequalities such that it follows a piecewise-linear (PL) path on a PL-manifold constructed using given polyhedral constraints without applying any reduction. This is a key contrast to the existing solver PATH which is oblivious of those constraints except for preprocessing purposes and a QR decomposition-based method that performs a reduction, thus destroying the structure, if there is nontrivial lineality space. SELKIE is a solver for equilibrium problems which enables various decomposition schemes based on groups of agents information to be instantiated in a flexible and adaptable way. Parallelism can be achieved either whenever independent groups of agents are detected or per user's request. A sub-solver for each sub-model can be chosen so that a highly efficient solver can be employed tailored to a certain problem type. Examples illustrating the efficiency and effectiveness of our extension and solvers are given.

All our interfaces and solvers have been implemented and are available within GAMS/EMP.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Professor Michael Ferris. His class on nonlinear optimization in my first semester stimulated my interest in optimization, and many discussions that I had with him while taking his linear programming class gave me confidence in studying optimization for my Ph.D. Over the years, I have enjoyed a lot studying and discussing with him. Without his support, encouragement, enthusiasm for pursuing high standards, and valuable ideas, this thesis would not have been possible.

I would like to thank Stephen Robinson, Stephen Wright, Thomas Rutherford, and Alberto Del Pia for serving on my Ph.D. committee. I am especially grateful to Professor Robinson for his excellent classes on convex and variational analysis that significantly deepened my understanding, and to Professor Rutherford for his valuable comments and examples that greatly improved the thesis. I am also grateful to Professor Linderoth for his generous help and encouragement.

I would like to thank Steve Dirkse and Michael Bussieck for their time and effort to help me link my software into a GAMS system.

Many thanks to my colleagues at the Wisconsin Institute for Discovery, Yanchao Liu, Jesse Holzer, Lisa Tang, Taedong Kim, Cong Han Lim, Wonjun Chang, Carla Michini,

Jagdish Ramakrishnan, Olivier Huber, Adam Christensen, and Steven Wangen. I would like to give special thanks to Herman Stampfli and Angela Thorp for their help with administrative work.

Finally, I would like to thank my family for their love and devotion. They have been always there for me.

This work was supported in part by Air Force Grant FA9550-15-1-0212, as well as by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contact number DE-AC02-06CH11357 through the Project “Multifaceted Mathematics for Complex Energy Systems”.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Variational inequalities (VIs) are a generalization of nonlinear system of equations, so-called generalized equations. In addition to the system of equations, they subsume geometric first-order optimality conditions, nonlinear (linear) complementarity problems, and mixed complementarity problems. Representative applications are equilibrium problems such as generalized Nash equilibrium problems (GNEPs) and multiple optimization problems with equilibrium constraints (MOPECs). VIs can be also used to formulate some form of equilibrium problems with equilibrium constraints (EPECs) when the equilibrium constraints can be represented as a system of equations in this case.

The theme of this thesis is algorithms and interfaces for structured variational inequalities and their extensions. Algorithms and interfaces are closely related to each other in a way that interfaces helping identify problem structures can lead to more robust and efficient algorithms, and structure-exploiting algorithms can guide us to design better structure-exposing interfaces. We show how our interfaces allow users to expose inherent structures

of their problems and present the way our algorithms exploit those structures for improved performance.

On the interface side, we introduce an extended mathematical programming (EMP) framework in Chapter 2 which enables us to specify and solve equilibrium problems such as GNEPs [29, 44] and MOPECs [10, 65] in a natural and intuitive way in modeling languages, for example AMPL [39], GAMS [11], or Julia [6], without requiring the modeler to supply derivatives. In contrast to the traditional way of specifying a single optimization problem (a single agent model) in modeling languages, we need information about ownership of equations and variables of agents in equilibrium problems. As there are multiple agents in a single equilibrium model, we need to know which variables and equations are owned by which agents to identify each agent's problem. This is critical to construct correct first-order optimality conditions as there could be interactions between agents. Extensions to support some complicated structures such as shared constraints, shared variables, and quasi-variational inequalities on the EMP framework are presented. The framework generates a human-readable file using which we can easily identify a high-level structure of the problem.

On the algorithmic side, an extension to PATH [19, 37] and our two solvers, PATHAVI [50] and SELKIE [48], will be introduced where they exploit problem structures such as implicitly defined variables, polyhedral constraints, and groups of interacting agents, respectively. In Chapter 2, we introduce an extension to PATH, called spacer steps, which restores feasibility of the primal and dual implicitly defined variables at each major iteration of PATH via the implicit function theorem. We apply our spacer steps to some economic application formulated as EPEC, where economic state variables are implicitly defined by strategic

policy variables. We show that performance was improved significantly via our spacer steps.

In Chapter 3, we introduce PathAVI, a structure-preserving pivotal method for affine variational inequalities (AVIs). To compute a solution, it solves a normal map [67] by following a piecewise-linear (PL) path on a PL-manifold constructed using given polyhedral constraints. As it constructs the manifold, it does not apply a reduction although there is nontrivial lineality space thus we preserve structure of the original problem, especially its sparsity. These are a key contrast to the existing approach Path, which is oblivious of polyhedral constraints except for preprocessing purposes, and a QR decomposition-based method [14] that performs a reduction if there is nontrivial lineality space. The maximum number of pieces of the PL path of PathAVI could be significantly smaller than the ones of Path, and feasibility of constraints is guaranteed during path following. Linear algebra computations for pivoting could be much cheaper as sparsity is preserved. These make PathAVI more efficient and robust solver compared to the existing approaches. Examples comparing performance between PathAVI and Path over friction contact problems, AVIs defined on compact sets, and Nash equilibrium problems are given. As efficient and stable linear algebra computations play a significant role in performance of pivotal method, we implemented block LU update [26] and compare performance between LUSOL [75] and LUSOL and UMFPACK [18] with block LU update applied. Finally, we briefly describe our nonlinear extension of PathAVI for nonlinear VIs.

Chapter 4 presents Selkie, a solver for equilibrium problems which enables various decomposition schemes based on groups of agents information to be instantiated in a flexible and adaptable way. To achieve this, it transforms a given model into a set of structure-

exploiting sub-models via structure analysis and taking into account user's knowledge. A diagonalization method is then applied to those sub-models possibly with parallel computations for making full use of computational resources. Depending on the configurations of sub-model generations and diagonalization method to use, various decomposition schemes can be implemented. We can choose a sub-solver to use for each sub-model so that a highly efficient solver can be employed tailored to a certain problem type. For stronger convergence results and numerical stability, primal and dual proximal perturbations are implemented. Examples illustrating the flexibility and effectiveness of Selkie are given.

For the rest of this chapter, we introduce basic backgrounds: i) mathematical definition of complementarity problems and (quasi-) variational inequalities in Section 1.1; ii) GNEPs and MOPECs described by a set of agents in Section 1.2; iii) the equivalence between equilibrium problems and their variational forms in Section 1.3. Other backgrounds will be provided as needed as we proceed.

1.1 Complementarity problems and variational inequalities

We introduce QVIs, VIs, and MCPs in a finite-dimensional space. For a given continuous function $F : \mathbb{R}^n \to \mathbb{R}^n$ and a point-to-set mapping $K(x) : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ where $K(x)$ is a closed convex set for each $x \in \mathbb{R}^n$, $x^* \in K(x^*)$ is a solution to the QVI(K, F) if

$$\langle F(x^*), x - x^* \rangle \geq 0, \quad \forall x \in K(x^*), \tag{QVI}$$

where $\langle \cdot, \cdot \rangle$ is the Euclidean inner product.

If we restrict the point-to-set mapping $K(\cdot)$ to be a fixed closed convex set $K \subset \mathbb{R}^n$, then $x^* \in K$ is a solution to the VI(K, F) if

$$\langle F(x^*), x - x^* \rangle \geq 0, \quad \forall x \in K. \tag{VI}$$

If we further specialize to the case where the feasible region is a box $B = \{x \in \mathbb{R}^n \mid l_i \leq x_i \leq u_i, \text{ for } i = 1, \ldots, n\}$ with $l_i \leq u_i$ and $l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{\infty\}$, the VI($B, F$) is typically termed a mixed complementary problem, and it is easy to see that $x^* \in B$ is a solution to the MCP(B, F) if one of the following conditions holds for each $i = 1, \ldots, n$:

$$\begin{aligned} x_i^* = l_i, \quad & F_i(x^*) \geq 0, \\ l_i \leq x_i^* \leq u_i, \quad & F_i(x^*) = 0, \\ x_i^* = u_i, \quad & F_i(x^*) \leq 0. \end{aligned} \tag{MCP}$$

In shorthand notation, the above condition is written as $l \leq x^* \leq u \perp F(x^*)$. We sometimes use MCP($x, F$) when the feasible region of x is clear from the context.

1.2 Equilibrium problems: GNEPs and MOPECs

A GNEP or MOPEC is defined by a set of agents. Each agent represents either an optimization problem or equilibrium conditions, such as market clearing conditions, formulated as variational inequalities (VI). Mathematically, a typical equilibrium problem is defined as

follows:

$$\begin{aligned} &\text{find} && (x_1^*, x_2^*, \ldots, x_N^*, x_{N+1}^*) \quad \text{satisfying,} \\ &x_i^* \in && \underset{x_i \in K_i(x_{-i}^*)}{\arg\min} \quad f_i(x_i, x_{-i}^*), \quad \text{for } i = 1, \ldots, N, \\ &x_{N+1}^* \in && \text{SOL}\left(K_{N+1}\left(x_{-(N+1)}^*\right), F\left(x_{N+1}, x_{-(N+1)}^*\right)\right), \end{aligned} \tag{1.1}$$

where x_i and x_{-i} represent vectors of decision variables of agent i and all the other agents except for agent i, respectively, for $i = 1, \ldots, N+1$. The function $f_i(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is the objective function of agent i for $i = 1, \ldots, N$, and a set-valued mapping $K_i(\cdot) : \mathbb{R}^{n-n_i} \rightrightarrows \mathbb{R}^{n_i}$ defines agent i's feasible region for $i = 1, \ldots, N+1$, where n_i is the dimension of its decision variables with $\sum_{i=1}^{N+1} n_i = n$. The notation $\text{SOL}(K, F)$ represents a set of solutions to the variational inequality $\text{VI}(K, F)$. Throughout this thesis, we assume that f_i's are twice continuously differentiable and F is continuously differentiable.

There could be interactions between agents in (1.1) such that decisions made by agents could affect some other agent's decision in the form of its feasibility, objective function value or clearing conditions. For example, x_{-i} could appear in the functional part of agent i's problem or its feasible region or both. These interactions make it hard to apply existing theories and algorithms. Refer to [29, 44] for more details.

Note that if there are no VI agents in (1.1), then the problem is called a GNEP. Otherwise, it is a MOPEC. For a GNEP, if each optimization agent's feasible region is a fixed set, that is $K_i(\cdot) \equiv K_i$ for $i = 1, \ldots, N$, then it is called a Nash equilibrium problem (NEP).

Throughout this thesis, we assume that equilibrium problems are of the form (1.1), call it MOPEC, and there are $(N+1)$ number of agents where the first N agents are optimization agents, and the $(N+1)$th agent is an equilibrium agent. When there is no equilibrium agent, then the problem becomes a GNEP. If there are no optimization agents but a single

equilibrium agent, then the problem is a VI.

1.3 Equivalence between equilibrium problems and their variational forms

The results described below are simple extensions of existing results found in [44]. We first show the equivalence between the equilibrium problems and their associated QVIs.

Proposition 1.1. *If $f_i(\cdot,\cdot)$ is continuously differentiable, $f_i(\cdot, x_{-i})$ is a convex function, and $K_i(x_{-i})$ is a closed convex set for each given x_{-i}, then x^* is a solution to the equilibrium problem MOPEC if and only if it is a solution to the QVI(K, F) where*

$$K(x) = \prod_{i=1}^{N+1} K_i(x_{-i}),$$

$$F(x) = (\nabla_{x_1} f_i(x_1, x_{-1})^{\mathrm{T}}, \ldots, \nabla_{x_N} f_N(x_N, x_{-N})^{\mathrm{T}}, G(x_{N+1}, x_{-(N+1)})^{\mathrm{T}})^{\mathrm{T}}.$$

Proof. ($\Rightarrow$) Let x^* be a solution to the MOPEC. For optimization agents, the first-order optimality conditions are necessary and sufficient by the given assumption. Therefore we have

$$\langle \nabla_{x_i} f_i(x_i^*, x_{-i}^*), x_i - x_i^* \rangle \geq 0, \quad \forall x_i \in K_i(x_{-i}^*), \text{ for } i = 1, \ldots, N.$$

Also we have

$$\langle G(x_{N+1}^*, x_{-(N+1)}^*), x_{N+1} - x_{N+1}^* \rangle \geq 0, \quad \forall x_{N+1} \in K_{N+1}(x_{-(N+1)}^*).$$

The result follows.

($\Leftarrow$) Let x^* be a solution to the QVI(K, F). The result immediately follows from the fact that $K(x)$ is a product space of $K_i(x_{-i})$'s for $i = 1, \ldots, N+1$. □

If each of the agents i has knowledge of a closed convex set X and use this to define their feasible region $K_i(x_{-i})$ using a shared constraint $K_i(x_{-i}) := \{x_i \in \mathbb{R}^{n_i} \mid (x_i, x_{-i}) \in X\}$, then the QVI$(K, F)$ can be solved using a simpler VI(X, F).

Proposition 1.2. *Suppose that* $K_i(x_{-i}) = \{x_i \in \mathbb{R}^{n_i} \mid (x_i, x_{-i}) \in X\}$ *for* $i = 1, \ldots, N+1$ *with* X *being a closed convex set. If* x^* *is a solution to the* VI(X, F) *with* F *defined in Proposition 1.1, then it is a solution to the QVI*(K, F)*, thus it is a solution to the MOPEC with the same assumptions on* $f_i(\cdot)$ *given in Proposition 1.1. The converse may not hold.*

Proof. ($\Rightarrow$) Let x^* be a solution to the VI(X, F). Clearly, $x^* \in K(x^*)$. Suppose there exists $x \in K(x^*)$ such that $\langle F(x^*), x - x^* \rangle < 0$. We have $x_i \in K_i(x^*_{-i})$ so that $(x_i, x^*_{-i}) \in X$ for $i = 1, \ldots, N+1$. There must exist $i \in \{1, \ldots, N+1\}$ satisfying $\langle F_i(x^*), x_i - x^*_i \rangle < 0$. Set $\tilde{x} = (x_i, x^*_{-i})$. Then $\tilde{x} \in X$, but $\langle F(x^*), \tilde{x} - x^* \rangle < 0$, which is a contradiction.

($\Leftarrow$) See the example in Section 3 of [44]. □

When the constraints are explicitly given as equalities and inequalities with a suitable constraint qualification holding, we can compute a solution to the equilibrium problems from their associated MCP and vice versa. Throughout this section, by a suitable constraint qualification we mean a constraint qualification satisfying the Guignard constraint qualification [41], for example the Mangasarian-Fromovitz or the Slater constraint qualification. Also when we say a constraint qualification holds at x, we imply that it holds at $x_i \in K_i(x_{-i})$ for each agent i.

Proposition 1.3. *Suppose that* $K_i(x_{-i}) = \{x_i \in [l_i, u_i] \mid h_i(x_i, x_{-i}) = 0, g_i(x_i, x_{-i}) \leq 0\}$ *where* $h_i(\cdot) : \mathbb{R}^n \to \mathbb{R}^{v_i}$ *is an affine function, each* $g_i(\cdot) : \mathbb{R}^n \to \mathbb{R}^{m_i}$ *is continuously differentiable and a convex function of* x_i *and* $l_i \leq u_i, l_i \in \mathbb{R}^{n_i} \cup \{-\infty\}^{n_i}$, *and* $u_i \in \mathbb{R}^{n_i} \cup \{\infty\}^{n_i}$. *With the same assumptions on* f_i *given in Proposition 1.1,* x^* *is a solution to the MOPEC if and only if* (x^*, λ^*, μ^*) *is a solution to the* MCP(B, F), *assuming that constraint qualification holds at* x^* *with*

$$
\begin{aligned}
B = \quad & \prod_{i=1}^{N+1} [l_i, u_i] \times \mathbb{R}^v \times \mathbb{R}^m_-, \quad v = \sum_{i=1}^{N+1} v_i, \quad m = \sum_{i=1}^{N+1} m_i, \\
F(x, \lambda, \mu) = \quad & ((\nabla_{x_1} f_1(x) - \nabla_{x_1} h_1(x)\lambda_1 - \nabla_{x_1} g_1(x)\mu_1)^{\mathrm{T}}, \ldots, \\
& (\nabla_{x_N} f_N(x) - \nabla_{x_N} h_N(x)\lambda_N - \nabla_{x_N} g_N(x)\mu_N)^{\mathrm{T}}, \\
& (G(x) - \nabla_{x_{N+1}} h_{N+1}(x)\lambda_{N+1} - \nabla_{x_{N+1}} g_{N+1}(x)\mu_{N+1})^{\mathrm{T}}, \\
& h_1(x)^{\mathrm{T}}, \ldots, h_{N+1}(x)^{\mathrm{T}}, \\
& g_1(x)^{\mathrm{T}}, \ldots, g_{N+1}(x)^{\mathrm{T}})^{\mathrm{T}}.
\end{aligned}
$$

Proof. ($\Rightarrow$) Let x^* be a solution to the MOPEC. Using the KKT conditions of each optimization agent and the VI, and constraint qualification at x^*, there exist (λ^*, μ^*) such that

$$
\begin{aligned}
\nabla_{x_i} f_i(x^*) - \nabla_{x_i} h_i(x^*)\lambda_i^* - \nabla_{x_i} g_i(x^*)\mu_i^* \quad & \perp \quad l_i \leq x_i^* \leq u_i, \quad \text{for} \quad i = 1, \ldots, N, \\
G(x^*) - \nabla_{x_i} h_i(x^*)\lambda_i^* - \nabla_{x_i} g_i(x^*)\mu_i^* \quad & \perp \quad l_i \leq x_i^* \leq u_i, \quad \text{for} \quad i = N+1, \\
h_i(x^*) \quad & \perp \quad \lambda_i^* \text{ free}, \qquad \text{for} \quad i = 1, \ldots, N+1, \\
g_i(x^*) \quad & \perp \quad \mu_i^* \leq 0, \qquad \text{for} \quad i = 1, \ldots, N+1.
\end{aligned}
\tag{1.2}
$$

Thus (x^*, λ^*, μ^*) is a solution to the MCP(B, F).

($\Leftarrow$) Let (x^*, λ^*, μ^*) be a solution to the MCP(B, F). Then (x^*, λ^*, μ^*) satisfies (1.2). Since the constraint qualification holds at x^*, we have $N_{K_i(x^*_{-i})} = \{-\nabla_{x_i} h_i(x^*)\lambda_i - \nabla_{x_i} g_i(x^*)\mu_i \mid h_i(x^*) \perp \lambda_i, g_i(x^*) \perp \mu_i \leq 0\} + N_{[l_i,u_i]}(x_i^*)$ for $i = 1, \ldots, N+1$. The result follows from convexity. □

If the convexity assumptions on the objective functions and the constraints of optimization agents' problems do not hold, then one can easily check that with a suitable constraint qualification assumption a stationary point to MOPEC is a solution to the MCP model defined in Proposition 1.3 and vice versa. By a stationary point, we mean that x_i^* satisfies the first-order optimality conditions of each optimization agent i's problem, and x_{N+1}^* is a solution to the equilibrium agent's problem.

Finally, we present the equivalence between QVIs and MCPs.

Proposition 1.4. *For a given QVI(K, F), suppose that $K(x) = \{l \leq y \leq u \mid h(y, x) = 0, g(y, x) \leq 0\}$ where $h : \mathbb{R}^{n\times n} \to \mathbb{R}^v$ and $g : \mathbb{R}^{n\times n} \to \mathbb{R}^m$. Assuming that constraint qualifications hold, x^* is a solution to the QVI(K, F) if and only if (x^*, λ^*, μ^*) is a solution to the* MCP$(B, \tilde{F})$ *where*

$$B = [l, u] \times \mathbb{R}^v \times \mathbb{R}^m_-,$$

$$\tilde{F}(x, \lambda, \mu) = \begin{bmatrix} F(x) - \nabla_y h(x, x)\lambda - \nabla_y g(x, x)\mu \\ h(x, x) \\ g(x, x) \end{bmatrix}$$

Proof. By applying similar techniques used in the proof of Proposition 1.3, we get the

desired result. □

Chapter 2

Solving equilibrium problems using extended mathematical programming framework

We introduce an extended mathematical programming framework for specifying equilibrium problems and their variational representations, such as generalized Nash equilibrium, multiple optimization problems with equilibrium constraints, and (quasi-) variational inequalities, and computing solutions of them from modeling languages. We define a new set of constructs with which users annotate variables and equations of the model to describe the equilibrium and variational problems. Our constructs enable a natural translation of the model from one formulation to another more computationally tractable form without requiring the modeler to supply derivatives. In the context of many independent agents in the equilibrium, we facilitate expression of sophisticated structures such as shared constraints

and additional constraints on their solutions. We define a new concept, shared variables, and demonstrate its uses for sparse reformulation, equilibrium problems with equilibrium constraints, mixed pricing behavior of agents, and so on. We give some equilibrium and variational examples from the literature and describe how to formulate them using our framework. Experimental results comparing performance of various complementarity formulations for shared variables are given. Our framework has been implemented and is available within GAMS/EMP.

2.1 Introduction

In this chapter, we present an extended mathematical programming (EMP) framework for specifying equilibrium problems and their variational representations and computing solutions of them in modeling languages such as AMPL, GAMS, or Julia [6, 11, 39]. Equilibrium problems of interest are (generalized) Nash equilibrium problems (GNEP) and multiple optimization problems with equilibrium constraints (MOPEC), and we consider quasi-variational inequalities (QVI) as their variational forms. All of these problems have been used extensively in the literature, see for example [10, 29, 44, 65].

The GNEP is a Nash game between agents with non-disjoint strategy sets. For a given number of agents N, $x^* = (x_1^*, \ldots, x_N^*)$ is a solution to the GNEP if it satisfies

$$x_i^* \in \underset{x_i \in K_i(x_{-i}^*) \subset \mathbb{R}^{n_i}}{\arg\min} \quad f_i(x_i, x_{-i}^*), \quad \text{for} \quad i = 1, \ldots, N, \tag{GNEP}$$

where $f_i(x_i, x_{-i})$ is the objective function of agent i, and $K_i(x_{-i})$ is its feasible region. Note that the objective function and the feasible region of each agent are affected by the decisions

of other agents, denoted by $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$. If each agent's feasible region is independent of other agents' decisions, that is, $K_i(x_{-i}) \equiv K_i$ for some nonempty set K_i, then the problem is called a Nash equilibrium problem (NEP).

In addition to the GNEP or NEP setting, if we have an agent formulating some equilibrium conditions, such as market clearing conditions, as a variational inequality (VI), we call the problem multiple optimization problems with equilibrium constraints (MOPEC). For example, $x^* = (x_1^*, \ldots, x_N^*, x_{N+1}^*)$ is a solution to the MOPEC if it satisfies

$$\begin{aligned} x_i^* &\in \underset{x_i \in K_i(x_{-i}^*) \subset \mathbb{R}^{n_i}}{\arg\min} \quad f_i(x_i, x_{-i}^*), \quad \text{for} \quad i = 1, \ldots, N, \\ x_{N+1}^* &\in \mathrm{SOL}(K_{N+1}(x_{-(N+1)}^*), F(x_{N+1}, x_{-(N+1)}^*)), \end{aligned} \tag{MOPEC}$$

where $\mathrm{SOL}(K, F)$ denotes the solution set of a variational inequality $\mathrm{VI}(K, F)$, assuming that $K_{N+1}(x_{-(N+1)})$ is a nonempty closed convex set for each given $x_{-(N+1)}$ and $F(x)$ is a continuous function. We call agent i for $i = 1, \ldots, N$ an optimization agent and agent $(N + 1)$ an equilibrium agent.

Solutions of equilibrium problems and their variational forms using modeling languages are usually obtained by transforming the problem into their equivalent complementarity forms, such as a mixed complementarity problem (MCP), and then solving the complementarity problem using a specialized solver for example PATH [19]. This implies that users need to compute the Karush-Kuhn-Tucker (KKT) conditions of each optimization agent by hand and then manually specify the complementarity relationships within modeling languages [34, 74]. Similar transformations are needed to formulate equilibrium problems in their variational forms represented by QVIs as we show in Section 1.3.

This approach has several drawbacks. It is time-consuming and error-prone because of the derivative computation. The problem structure becomes lost once it is converted into the complementarity form: it is difficult to tell what the original model is and which agent controls what variables and equations (functions and constraints) by just reading the complementarity form. For QVI formulations, we lose the information about what variables are used as parameters to define the feasible region. All variables and equations are endogenous in that form. This may restrict opportunities for back-end solvers to detect and make full use of the problem structure. Modifying the model such as adding/removing variables and equations may not be easy: it typically involves a lot of derivative recomputation.

For more intuitive and efficient equilibrium programming, that is, formulating GNEP, MOPEC, or QVI in modeling languages, the paper [33] briefly mentioned that the EMP framework can be used to specify GNEPs and MOPECs. Its goal is to enable users to focus on the problem description itself rather than spending time and checking errors on the complementarity form derivation. Users annotate variables and equations of the problem in a similar way to its algebraic representation and write them into an `empinfo` file. The modeling language reads that file to identify high level structure of the problem such as the number of agents and agent's ownership of variables and equations. It then automatically constructs the corresponding complementarity form and solves it using complementarity solvers. However, neither detailed explanations about its underlying assumptions and how to use it are given, nor are the QVI formulations considered in [33].

In this chapter, we present detailed explanation of the existing EMP framework for equilibrium programming for the first time. We also describe its extensions to incorporate some new sophisticated structures, such as *shared constraints*, *shared variables*, and *QVI*

formulations, and their implications with examples from the literature. Our extensions allow a natural translation of the algebraic formulation into modeling languages while capturing high level structure of the problem so that the back-end solver can harness the structure for improved performance.

Specifically, our framework allows shared constraints to be represented without any replications and makes it easy to switch between different solution types associated with them, for example variational equilibrium [29, Definition 3.10]. We introduce a new concept, shared variables, and show their manifestations in the literature. Shared variables have potential for many different uses: i) they can be used to reduce the density of the model; ii) they can model some EPECs sharing the same variables and constraints to represent equilibrium constraints; iii) we can easily switch between price-taking and price-making agents in economics models; iv) they can be used to model shared objective functions. The last case opens the door for our framework to be used to model the block coordinate descent method, where agents now correspond to a block of coordinates. Finally, we define a new construct that allows QVI formulations to be specified in an intuitive and natural way. The new features have been implemented and are available within GAMS/EMP. In this case, we use a problem reformulation solver JAMS, and choose formulations if necessary in an option file `jams.opt`.

The rest of the chapter is organized as follows. Section 2.2 presents the underlying assumptions of the existing framework and shows how we can model equilibrium problems satisfying those assumptions. For the two subsequent sections, we present sophisticated structures that violate the assumptions and introduce our modifications to incorporate them into our framework. Thus, Section 2.3 describes shared constraints and presents

a new construct to define the type of solutions, either GNEP equilibria or variational equilibria, associated with them. In Section 2.4, we introduce shared variables and various complementarity formulations for them. Section 2.5 presents a new construct to specify QVIs and compares two equivalent ways of specifying equilibrium problems in either GNEP or QVI form. At the end of each section of Sections 2.2-2.5, we provide examples from the literature that can be neatly formulated using the feature of our framework. Section 2.6 concludes the chapter, pointing out some areas for future extensions.

2.2 Modeling equilibrium problems using the existing EMP framework

We now describe how to specify equilibrium problems in modeling languages using the EMP framework. We first present the underlying assumptions on the specification and discuss their limitations in Section 2.2.1. Examples from the literature are given in Section 2.2.2. In Sections 2.3-2.4, we relax these assumptions to take into account more sophisticated structures.

2.2.1 Specifying equilibrium problems and underlying assumptions

Standard equilibrium problems can be specified in modeling languages using our framework. Suppose that we are given the following NEP:

$$\begin{array}{lll} \text{find} & (x_1^*, \ldots, x_N^*) & \text{satisfying,} \\ x_i^* \in & \underset{x_i}{\arg\min} & f_i(x_i, x_{-i}^*), \\ & \text{subject to} & g_i(x_i) \leq 0, \text{ for } i = 1, \ldots, N. \end{array} \tag{2.1}$$

We need to specify each agent's variables, its objective function, and constraints. Functions and constraints are given as a closed-form in modeling languages: they are explicitly written using combinations of mathematical operators such as summation, multiplication, square root, log, and so on. The EMP partitions the variables, functions, and constraints among the agents using annotations given in an `empinfo` file. For example, we may formulate and solve (2.1) within GAMS/EMP as follows:

Listing 2.1: Modeling the NEP

```
1  variables obj(i), x(i);
2  equations deff(i), defg(i);

4  * Definitions of deff(i) and defg(i) are omitted for expository
       purposes.

6  model nep / deff, defg /;

8  file empinfo / '%emp.info%' /;
9  put empinfo 'equilibrium';
```

```
10 loop(i,
11   put / 'min', obj(i), x(i), deff(i), defg(i) ;
12 );
13 putclose;

15 solve nep using emp;
```

Let us explain Listing 2.1. Variable `obj(i)` holds the value of $f_i(x)$, `x(i)` represents variable x_i, and `deff(i)` and `defg(i)` are the closed-form definitions of the objective function $f_i(x)$ and the constraint $g_i(x)$, respectively, for $i = 1, \ldots, N$. Equations listed in the model statement and variables in those equations constitute the model `nep`.

Once the model is defined, a separate `empinfo` file is created to specify the equilibrium problem. In the above case, the `empinfo` file has the following contents:

```
equilibrium
min obj('1') x('1') deff('1') defg('1')
...
min obj('N') x('N') deff('N') defg('N')
```

The `equilibrium` keyword informs EMP that the annotations are for an equilibrium problem. A list of agents' problem definitions separated by either a `min` or `max` keyword for each optimization agent follows. For each `min` or `max` keyword, the objective variable to optimize and a list of agent's decision variables are given. After these variables, a list of equations that define the agent's objective function and constraints follows. We say that variables and equations listed are owned by the agent. Note that variables other than `x('1')` that appear in `deff('1')` or `defg('1')` are treated as parameters to the first

agent's problem; that is how we define x_{-i}. The way each agent's problem is specified closely resembles its algebraic formulation (2.1), and our framework reconstructs each agent's problem by reading the `empinfo` file.

The framework does not require any special keyword to distinguish between a NEP and a GNEP. If the function g_i is defined using other agents' decisions, that is, $g_i(x_i, x_{-i}) \leq 0$, the equilibrium model written in Listing 2.1 becomes a GNEP. The distinction between the NEP and the GNEP depends only on how the constraints are defined.

Note that in the `empinfo` file above, each variable and equation is owned exclusively by a single agent. There is no unassigned variable or equation. In the standard framework, neither multiple ownership nor missing ownership are allowed; otherwise an error is generated. Formally, the standard framework assumes the following:

Assumption 2.1. *A model of an equilibrium problem described by equations and variables is assumed to have the following properties in the empinfo file:*

- *Each equation of the model is owned by a single agent.*
- *Each variable of the model is owned by a single agent.*

An implication of Assumption 2.1 is that the current framework does not allow *shared objective functions*, *shared constraints*, and *shared variables*. Section 2.3 gives examples of problems that violate Assumption 2.1 and provides techniques to overcome or relax the requirements.

The MOPEC model can be defined in a very similar way. Suppose that we are given the following MOPEC:

$$\begin{aligned} &\text{find} && (x_1^*, \ldots, x_N^*, p^*) \quad \text{satisfying,} \\ x_i^* \in\ & \arg\min_{x_i} && f_i(x_i, x_{-i}^*), \\ & \text{subject to} && g_i(x_i, x_{-i}^*) \le 0, \ \text{for } i = 1, \ldots, N, \\ p^* \in\ & \text{SOL}(K(x^*), H(p, x^*)), \\ & \text{where} && K(x^*) := \{p \mid w(p, x^*) \le 0\} \end{aligned} \tag{2.2}$$

Assuming that $p \in \mathbb{R}^r$, we can then formulate (2.2) within GAMS/EMP in the following way:

Listing 2.2: Modeling the MOPEC

```
1  variables obj(i), x(i), p(j);
2  equations deff(i), defg(i), defH(j), defw;

4  model mopec / deff, defg, defH, defw /;

6  file empinfo / '%emp.info%' /;
7  put empinfo 'equilibrium' /;
8  loop(i,
9    put 'min', obj(i), x(i), deff(i), defg(i) /;
10 );
11 put 'vi defH p defw' /;
12 putclose empinfo;
```

In addition to optimization agents, we now have an equilibrium agent defined with the `'vi'` keyword in Listing 2.2. The `'vi'` keyword is followed by variables, function-variable pairs, and constraints. Functions paired with variables constitute a VI function, and the order of functions and variables appeared in the pair is used to determine which variable is assigned to which function when we compute the inner product in the (VI) definition. In this case, we say that each VI function is matched with each variable having the same order in the pair, i.e., `defH(j)` is matched with `p(j)` for each $j = 1, \ldots, r$. After all matching information is described, constraints follow. Hence, the VI function is `defH`, its variable is `p`, and `defw` is a constraint. The functions f_i and g_i, defined in `deff(i)` and `defg(i)` equations, respectively, may now include the variable p. One can easily verify that the specification in the `empinfo` file satisfies Assumption 2.1.

Variables that are used only to define the constraint set, and are owned by the VI agent, must be specified before any explicit function-variable pairs. In this case, we call those variables *preceding variables*. The interface automatically assigns them to a zero function, that is, a constant function having zero value. For example, if we specify `'vi z Fy y'` where `z` and `y` are variables and `Fy` is a VI function matched with `y`, then `z` is a preceding variable. In this case, our interface automatically creates an artificial function `Fz` defined by $F(z) \equiv 0$ and match it with variable `z`.

2.2.2 Examples

Examples of NEP, GNEP, and MOPEC taken from the literature are formulated in the following sections using the EMP framework.

NEP

We consider the following oligopolistic market equilibrium problem [42, 59]:

$$
\begin{aligned}
&\text{find} \quad (q_1^*, \ldots, q_5^*) \quad \text{satisfying,} \\
&q_i^* \in \quad \underset{q_i \geq 0}{\arg\max} \quad q_i p\left(\sum_{j=1, j\neq i}^{5} q_j^* + q_i\right) - f_i(q_i), \\
&\qquad \text{where} \quad p(Q) := 5000^{1/1.1} (Q)^{-1/1.1}, \\
&\qquad\qquad f_i(q_i) := c_i q_i + \frac{\beta_i}{\beta_i + 1} K_i^{-1/\beta_i} q_i^{(\beta_i+1)/\beta_i}, \\
&\qquad\qquad (c_i, K_i, \beta_i) \text{ is problem data,} \qquad\qquad \text{for} \quad i = 1, \ldots, 5.
\end{aligned}
\tag{2.3}
$$

There are five firms, and each firm provides a homogeneous product with amount q_i to the market while trying to maximize its profit in a noncooperative way. The function $p(\cdot)$ is the inverse demand function, and its value is determined by the sum of the products provided by all the firms. The function $f_i(\cdot)$ is the total cost of firm i. The problem (2.3) is a NEP.

Listing 2.3 shows an implementation of (2.3) within GAMS/EMP. As we see, the `empinfo` file is a natural translation of the algebraic form of (2.3). Using the same starting value as in [42, 59], our GAMS/EMP implementation computed a solution $q^* = (36.933, 41.818, 43.707, 42.659, 39.179)^{\mathrm{T}}$ that is consistent with the one reported in those papers.

Listing 2.3: Implementation of the NEP (2.3) within GAMS/EMP

```
1 sets i agents / 1*5 /;
2 alias(i,j);
```

```
4  parameters
5      c(i)     / 1  10, 2   8, 3   6, 4   4, 5   2 /
6      K(i)     / 1   5, 2   5, 3   5, 4   5, 5   5 /
7      beta(i)  / 1 1.2, 2 1.1, 3 1.0, 4 0.9, 5 0.8 /
8      ;

10 variables obj(i);
11 positive variables q(i);

13 equations
14     objdef(i)
15     ;

17 objdef(i)..
18     obj(i) =e= q(i)*5000**(1.0/1.1)*sum(j, q(j))**(-1.0/1.1) - (c(i)*q(
           i) + beta(i)/(beta(i)+1)*K(i)**(-1/beta(i))*q(i)**((beta(i)+1)/
           beta(i)));

20 model nep / objdef /;

22 file empinfo / '%emp.info%' /;
23 put empinfo 'equilibrium' /;
24 loop(i,
25     put 'max', obj(i), q(i), objdef(i) /;
26 );
27 putclose empinfo;

29 q.l(i) = 10;
```

```
30 solve nep using emp;
```

GNEP

We use the following GNEP example derived from the QVI example in [62, page 14]:

$$\begin{array}{lll} \text{find} & (x_1^*, x_2^*) & \text{satisfying,} \\ x_1^* \in & \underset{0\le x_1\le 11}{\arg\min} & x_1^2 + \frac{8}{3}x_1 x_2^* - \frac{100}{3}x_1, \\ & \text{subject to} & x_1 + x_2^* \le 15, \\ x_2^* \in & \underset{0\le x_2\le 11}{\arg\min} & x_2^2 + \frac{5}{4}x_1^* x_2 - 22.5x_2, \\ & \text{subject to} & x_1^* + x_2 \le 20. \end{array} \tag{2.4}$$

In (2.4), each agent solves a strongly convex optimization problem. Not only the objective functions but also the feasible region of each agent is affected by other agent's decision. Hence it is a GNEP. Listing 2.4 shows an implementation of (2.4) within GAMS/EMP. Our model has computed a solution $(x_1^*, x_2^*) = (10, 5)$ that is consistent with the one reported in [62]. In Section 2.5.2, we show that (2.4) can be equivalently formulated as a QVI using our extension to the EMP framework.

Listing 2.4: Implementation of the GNEP (2.4) within GAMS/EMP

```
1 set i / 1*2 /;
2 alias(i,j);

4 variable obj(i);
5 positive variable x(i);
```

```
7  equation defobj(i), cons(i);

9  defobj(i)..
10     obj(i) =E=
11     (sqr(x(i)) + 8/3*x(i)*x('2') - 100/3*x(i))$(i.val eq 1) +
12     (sqr(x(i)) + 5/4*x('1')*x(i) - 22.5*x(i))$(i.val eq 2);

14 cons(i)..
15     sum(j, x(j)) =L= 15$(i.val eq 1) + 20$(i.val eq 2);

17 x.up(i) = 11;

19 model gnep / defobj, cons /;

21 file empinfo / '%emp.info%' /;
22 put empinfo 'equilibrium' /;
23 loop(i,
24     put 'min',obj(i),x(i),defobj(i),cons(i);
25 );
26 putclose empinfo;

28 solve gnep using emp;
```

MOPEC

We present a general equilibrium example in economics [58, Section 3] and model it as a MOPEC. While [58] formulated the problem as a complementarity problem by using the closed form of the utility maximizing demand function, we formulate it as a MOPEC by explicitly introducing a utility-maximizing optimization agent, the consumer, to compute the demand.

Let us briefly explain the general equilibrium problem we consider. We use the notations and explanation from [58]. There are three types of agents: i) profit-maximizing producers; ii) utility-maximizing consumers; iii) a market determining the price of commodities based on production and demand. The problem is given with a technology matrix A, an initial endowment b, and the demand function $d(p)$. The coefficient $a_{ij} > 0$ (or $a_{ij} < 0$) of A indicates output (or input) of commodity i for each unit activity of producer j. For a given price p, $d(p)$ is the demand of consumers maximizing their utilities within their budgets, where budgets depend on the price p and initial endowment b. Assuming that y, x, and p represent activity of producers, demands of consumers, and prices of commodities,

respectively, we say that (y^*, x^*, p^*) is a general equilibrium if it satisfies the following:

$$
\begin{aligned}
&\text{No positive profit for each activity} && -A^{\mathrm{T}}p^* \geq 0,\\
&\text{No excess demand} && b + Ay^* - x^* \geq 0,\\
&\text{Nonnegativity} && p^* \geq 0, y^* \geq 0,\\
&\text{No activity for earning negative profit} && (-A^{\mathrm{T}}p^*)^{\mathrm{T}}y^* = 0,\\
&\text{and positive activity implies balanced profit,} && \\
&\text{Zero price for excess supply} && p^{*\mathrm{T}}(b + Ay^* - x^*) = 0,\\
&\text{and market clearance for positive price,} && \\
&\text{Utility maximizing demand} && x^* \in \arg\max_x \quad \text{utility}(x),\\
& && \text{subject to} \quad p^{*\mathrm{T}}x \leq p^{*\mathrm{T}}b.
\end{aligned}
\tag{2.5}
$$

We consider a market where there are a single producer, a single consumer, and three commodities. To compute the demand function without using its closed form, we introduce a utility-maximizing consumer explicitly in the model. Our GAMS/EMP model finds a solution $y^* = 3, x^* = (3, 2, 0)^{\mathrm{T}}, p^* = (6, 1, 5)^{\mathrm{T}}$ for $\alpha = 0.9$ that is consistent with the one in [58].

Listing 2.5: Implementation of the MOPEC within GAMS/EMP

```
1  set i commodities / 1*3 /;

3  parameters
4      ATmat(i)  technology matrix  / 1 1  , 2 -1 , 3 -1 /
5      s(i)      budget share       / 1 0.9, 2 0.1, 3 0 /
6      b(i)      endowment          / 1 0  , 2 5  , 3 3 /;
```

```
8  variable u    utility of the consumer;
9  positive variables
10     y         activity of the producer
11     x(i)      Marshallian demand of the consumer
12     p(i)      prices;

14 equations
15     mkt(i)    constraint on excess demand
16     profit    profit of activity
17     udef      Cobb-Douglas utility function
18     budget    budget constraint;

20 mkt(i)..
21     b(i) + ATmat(i)*y - x(i) =G= 0;

23 profit..
24     sum(i, -ATmat(i)*p(i)) =G= 0;

26 udef..
27     u =E= sum(i, s(i)*log(x(i)));

29 budget..
30     sum(i, p(i)*x(i)) =L= sum(i, p(i)*b(i));

32 model mopec / mkt, profit, udef, budget /;

34 file empinfo / '%emp.info%' /;
```

```
35 put empinfo 'equilibrium' /;
36 put 'max', u, 'x', udef, budget /;
37 * We have mkt perp p and profit perp y, the fourth and fifth conditions
       of (6).
38 put 'vi mkt p profit y' /;
39 putclose empinfo;

41 * The second commodity is used as a numeraire.
42 p.fx('2') = 1;
43 x.l(i) = 1;

45 solve mopec using emp;
```

2.3 Modeling equilibrium problems with shared constraints

This section describes our first extension to model shared constraints and to compute different types of solutions associated with them.

2.3.1 Shared constraints and limitations of the existing framework

We first define shared constraints in equilibrium problems, specifically when they are explicitly given as equalities or inequalities.

Definition 2.2. *In equilibrium problems, if the same constraint given explicitly as an equality or an inequality appears multiple times in different agents' problem definition,*

then it is a shared constraint.

For example, constraint $h(x) \leq 0$ (with no subscript i on h) is a shared constraint in the following GNEP:

Example 2.3. *Find* $(x_1^*, \ldots, x_N^*)$ *satisfying*

$$\begin{aligned} x_i^* \in \quad & \arg\min_{x_i} \quad f_i(x_i, x_{-i}^*), \\ & \textit{subject to} \quad g_i(x_i, x_{-i}^*) \leq 0, \\ & \qquad\qquad h(x_i, x_{-i}^*) \leq 0, \quad \textit{for} \quad i = 1, \ldots, N. \end{aligned}$$

Our definition of a shared constraint allows each agent's feasible region to be defined with a combination of shared and non-shared constraints. Our definition subsumes the cases in [28, 29], where each agent's feasible region is defined by the shared constraint only: there are no $g_i(x)$'s. In our framework, the shared constraint can also be defined over some subset of agents. For expository ease throughout this section, we use Example 2.3, but the extension to the more general setting is straightforward.

Shared constraints are mainly used to model shared resources among agents. In the tragedy of commons example [61, Section 1.1.2], agents share a capped channel formulated as a shared constraint $\sum_{i=1}^{N} x_i \leq 1$. Another example is the river basin pollution game in [45, 52], where the total amount of pollutant thrown in the river by the agents is restricted. The environmental constraints are shared constraints in this case. More details on how we model these examples are found in Section 2.3.3.

There are two types of solutions when shared constraints are present. Assume a suitable constraint qualification holds for each solution x^* of Example 2.3. Let μ_i^* be a multiplier associated with the shared constraint $h(x)$ for agent i at the solution x^*. If $\mu_1^* = \cdots = \mu_N^*$, then we call the solution a *variational equilibrium*. The name of the solution stems from the fact that if there are no $g_i(x)$'s, then x^* is a solution to the VI(X, F) and vice versa of Proposition 1.2, where $X = \{x \in \mathbb{R}^n \mid h(x) \leq 0\}$. In all other cases, we call a solution a GNEP equilibrium.

An interpretation from the economics point of view is that, at a variational equilibrium, agents have the same marginal value on the resources associated with the shared constraint (as the multiplier values are the same), whereas at a GNEP equilibrium each agent may have a different marginal value.

A shared constraint may not be easily modeled using the existing EMP framework. As each equation must be assigned to a single agent, we currently need to create a replica of the shared constraint for each agent. For Example 2.3, we may model the problem within GAMS/EMP as follows:

Listing 2.6: Modeling the GNEP equilibrium via replications

```
1 variables obj(i), x(i);
2 equations deff(i), defg(i), defh(i);

4 model gnep_shared / deff, defg, defh /;

6 file empinfo / '%emp.info%' /;
7 put empinfo 'equilibrium';
8 loop(i,
```

```
 9    put / 'min', obj(i), x(i), deff(i), defg(i), defh(i);
10  );
11  putclose empinfo;
```

In Listing 2.6, each `defh(i)` is defined exactly in the same way for all $i = 1, \ldots, N$: each of them is a replica of the same equation. This approach is neither natural nor intuitive compared to its algebraic formulation. It is also difficult to tell if the equation `defh` is a shared constraint by just reading the `empinfo` file. The information that `defh` is a shared constraint is lost. This could potentially prevent applying different solution methods such as [76] tailored for the equilibrium problems containing shared constraints.

Another difficulty lies in modeling the variational equilibrium. To compute it, we need to have the multipliers associated with the shared constraints the same among the agents. Additional constraints may be required for such conditions to hold; there is no easy way to force equality without changing the model in the existing EMP framework.

2.3.2 Extensions to model shared constraints

Our extensions have two new features: i) we provide a syntactic enhancement that enables shared constraints to be naturally and succinctly specified in a similar way to the algebraic formulation; ii) we define a new EMP keyword that enables switching between the GNEP and variational equilibrium without modifying each agent's problem definition.

To implement shared constraints, we modify Assumption 2.1 as follows:

Assumption 2.4. *A model of an equilibrium problem described by equations and variables is assumed to have the following properties in the empinfo file:*

- *Each objective or* VI *function of the model is owned by a single agent.*
- *Each constraint of the model is owned by at least one agent. If a constraint appears multiple times in different agents' problem definitions, then it is regarded as a shared constraint, and it is owned by those agents.*
- *Each variable is owned by a single agent.*

Using Assumption 2.4, we define shared constraints by placing the same constraint in multiple agents' problems. For example, we can model Example 2.3 without replications by changing lines 2 and 8-10 of Listing 2.6 into the following:

Listing 2.7: Modeling a shared constraint using a single copy

```
1 equation deff(i), defg(i), defh;

3 loop(i,
4   put / 'min', obj(i), x(i), deff(i), defg(i), defh;
5 );
```

In Listing 2.7, a single instance of an equation, `defh`, representing the shared constraint $h(x) \leq 0$ is created and placed in each agent's problem description. Our framework then recognizes it as a shared constraint. This is exactly the same way as its algebraic formulation is specified. Also the `empinfo` file does not lose the problem structure: we can easily identify that `defh` is a shared constraint by reading the file, as it appears multiple times in different agents' problem definitions. To allow shared constraints, we need to specify `SharedEqu` in the option file `jams.opt`. Otherwise, multiple occurrences of the same

constraint are regarded as an error. This is simply a safety check to stop non-expert users creating incorrect models.

In addition to the syntactic extension, we define a new EMP keyword `visol` to compute a variational equilibrium associated with shared constraints. By default, a GNEP equilibrium is computed if no `visol` keyword is specified. Hence Listing 2.7 computes a GNEP equilibrium. If we place the following line in the `empinfo` file before the agents' problem descriptions begin, that is, before line 3 in Listing 2.7, then a variational equilibrium is computed. The keyword `visol` is followed by a list of shared constraints for which each agent owning the constraint must use the same multiplier.

Listing 2.8: Computing a variational equilibrium

```
1 put / 'visol defh';
```

Depending on the solution type requested, our framework creates different MCPs. For a GNEP equilibrium, the framework replicates the shared constraint and assigns a separate multiplier for each agent owning it. For Example 2.3, the following MCP(z, F) is generated:

$$
\begin{aligned}
&F(z) = ((F_i(z)^{\mathrm{T}})_{i=1}^{N})^{\mathrm{T}}, && z = ((z_i^{\mathrm{T}})_{i=1}^{N})^{\mathrm{T}},\\
&F_i(z) = \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x)\lambda_i - \nabla_{x_i} h(x)\mu_i \\ g_i(x) \\ h(x) \end{bmatrix}, && z_i = \begin{bmatrix} x_i \\ \lambda_i \le 0 \\ \mu_i \le 0 \end{bmatrix}, \quad \text{for} \quad i = 1, \ldots, N.
\end{aligned}
\tag{2.6}
$$

Note that the same equation $h(\cdot)$ is replicated, and a separate multiplier μ_i is assigned in (2.6) for each agent i for $i = 1, \ldots, N$.

If a variational equilibrium is requested, then our framework creates a single instance

of the shared constraint, and a single multiplier is used for that constraint among agents. Accordingly, we construct the following MCP(z, F) for Example 2.3:

$$
\begin{aligned}
&F(z) = ((F_i(z)^{\mathrm{T}})_{i=1}^{N}, F_h(z)^{\mathrm{T}})^{\mathrm{T}}, && z = ((z_i^{\mathrm{T}})_{i=1}^{N}, z_h^{\mathrm{T}})^{\mathrm{T}},\\
&F_i(z) = \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x)\lambda_i - \nabla_{x_i} h(x)\mu \\ g_i(x) \end{bmatrix}, && z_i = \begin{bmatrix} x_i \\ \lambda_i \leq 0 \end{bmatrix}, \qquad \text{for} \quad i = 1, \ldots, N,\\
&F_h(z) = \Big[h(x) \Big], && z_h = \Big[\mu \leq 0 \Big].
\end{aligned}
\tag{2.7}
$$

In (2.7), a single multiplier μ is assigned to the shared constraint $h(x)$, and $h(x)$ appears only once in the MCP. If there are no $g_i(x)$'s, then with a constraint qualification the problem exactly corresponds to VI(X, F) of Proposition 1.2 with the set X defined as $X := \{x \mid h(x) \leq 0\}$.

2.3.3 Examples

We present two GNEP examples having shared constraints in the following sections, respectively. The first example has a unique solution that is a variational equilibrium. Thus, with or without the `visol` keyword, our framework computes the same solution. In the second example, multiple solutions exist. Our framework computes solutions of different types depending on the existence of the `visol` keyword in this case.

GNEP with a shared constraint: tragedy of the commons

We consider the tragedy of the commons example [61, Section 1.1.2]:

$$\begin{aligned} &\text{find} && (x_1^*, \ldots, x_N^*) && \text{satisfying,} \\ x_i^* \in\ &\underset{0 \le x_i \le 1}{\arg\max} && x_i \left(1 - \left(x_i + \sum_{j=1, j\neq i}^{N} x_j^*\right)\right), \\ &\text{subject to} && x_i + \sum_{j=1, j\neq i}^{N} x_j^* \le 1. \end{aligned} \tag{2.8}$$

There is a shared channel with capacity 1, represented as a shared constraint $\sum_{j=1}^{N} x_j \le 1$, through which each agent i sends x_i units of flow. The value agent i obtains by sending x_i units is $x_i \left(1 - \sum_{j=1}^{N} x_j\right)$, and each agent tries to maximize its value. By the form of the problem, (2.8) is a GNEP with a shared constraint.

The problem has a unique equilibrium $x_i^* = 1/(N+1)$ for $i = 1, \ldots, N$. The value of agent i is then $1/(N+1)^2$, and the total value over all agents is $N/(N+1)^2 \approx 1/N$. As noted in [61], if agents choose to use $\sum_{i=1}^{N} x_i = 1/2$, then the total value will be 1/4 which is much larger than $1/N$ for large enough N. This is why the problem is called the tragedy of the commons.

We model (2.8) within GAMS/EMP in Listing 2.9. A single constraint `cap` is defined for the shared constraint, and the same equation `cap` appears in each agent's problem definition in the `empinfo` file.

Listing 2.9: Implementation of the GNEP (2.8) within GAMS/EMP

```
1 $if not set N $set N 5
```

```
3  set i / 1*%N% /;

5  alias(i,j);

7  variables obj(i);
8  positive variables x(i);

10 equations defobj(i), cap;

12 defobj(i)..
13     obj(i) =E= x(i)*(1 - sum(j, x(j)));

15 cap..
16     sum(i, x(i)) =L= 1;

18 model m / defobj, cap /;

20 file info / '%emp.info%' /;
21 put info 'equilibrium';
22 loop(i,
23     put / 'max', obj(i), x(i), defobj(i), cap
24 );
25 putclose;

27 x.up(i) = 1;

29 * Specify SharedEqu option in the jams.opt file to allow shared
      constraints.
```

```
30 $echo SharedEqu > jams.opt
31 m.optfile = 1;

33 solve m using emp;
```

By default, a GNEP equilibrium is computed. If we want to compute a variational equilibrium, we just need to place the following line between lines 21-22 in Listing 2.9.

```
1 put 'visol cap' /;
```

As the solution is unique $x_i^* = 1/(N+1)$ with multiplier $\mu_i^* = 0$ for $i = 1, \ldots, N$, our framework computes the same solution in both cases.

GNEP with shared constraints: river basin pollution game

We present another example where we have different solutions for GNEP and variational equilibria. The example is the river basin example [45, 52] described below:

$$
\begin{aligned}
&\text{find} \quad (x_1^*, x_2^*, x_3^*) \quad \text{satisfying,} \\
&x_i^* \in \quad \underset{x_i \geq 0}{\arg\min} \quad (c_{1i} + c_{2i} x_i) x_i - \left(d_1 - d_2 \left(\sum_{j=1, j \neq i}^{3} x_j^* + x_i \right) \right) x_i, \\
&\qquad \text{subject to} \quad \sum_{j=1, j \neq i}^{3} \left(u_{jm} e_j x_j^* \right) + u_{im} e_i x_i \leq K_m, \\
&\qquad \qquad \text{for} \quad m = 1, 2, \; i = 1, 2, 3, \\
&\qquad \text{where} \quad (c, d, e, u, K) \text{ is problem data.}
\end{aligned}
\tag{2.9}
$$

It has two shared constraints, and they are shared by all the three agents.

Let us briefly explain the model. There are three agents near a river, each of which pursues maximum profit by producing some commodities. The term $(c_{1i} + c_{2i}x_i)x_i$ denotes the total cost of agent i and $(d_1 - d_2(\sum_{j=1,j\neq i}^{3} x_j^* + x_i))x_i$ the revenue. Each agent can throw pollutant in the river, but its amount is limited by the two shared constraints in (2.9).

Listing 2.10 shows an implementation of (2.9) within GAMS/EMP. The two shared constraints are represented in the equations `cons(m)`. We first compute a variational equilibrium. A solution computed by our framework is $x^* = (21.145, 16.028, 2.726)$ with multipliers $\mu^*_{\texttt{cons1}} = -0.574$ and $\mu^*_{\texttt{cons2}} = 0$ for the shared constraints `cons('1')` and `cons('2')`, respectively.[1]

If we compute a GNEP equilibrium by deleting line 42 in Listing 2.10, then we find a solution $x^* = (0, 6.473, 22.281)$. In this case, multiplier values associated with the shared constraints for each agent are as follows:

$$\mu^*_{\texttt{cons1,1}} = -0.804, \quad \mu^*_{\texttt{cons1,2}} = -1.504, \quad \mu^*_{\texttt{cons1,3}} = -0.459,$$
$$\mu^*_{\texttt{cons2,1}} = \mu^*_{\texttt{cons2,2}} = \mu^*_{\texttt{cons2,3}} = 0$$

Listing 2.10: Implementation of (2.9) within GAMS/EMP

```
1  sets i / 1*3 /
2       m / 1*2 /
3       ;

5  alias(i,j);
```

[1]Note that we used the vector form for the constraints when we declare the equation `cons` for each agent so that we do not have to loop through the set `m`.

```
7  parameters
8      K(m) / 1 100, 2 100 /
9      d1   /   3     /
10     d2   /   0.01  /
11     e(i) / 1 0.5, 2 0.25, 3 0.75 /;

13 table c(m,i)
14         1      2      3
15   1   0.1    0.12   0.15
16   2   0.01   0.05   0.01;

18 table u(i,m)
19         1      2
20   1   6.5     4.583
21   2   5.0     6.250
22   3   5.5     3.750;

24 variables obj(i);
25 positive variables x(i);

27 equations
28     objdef(i)
29     cons(m)
30     ;

32 objdef(i)..
33     obj(i) =E= (c('1',i) + c('2',i)*x(i))*x(i) - (d1 - d2*sum(j, x(j)))
           *x(i);
```

```
35 cons(m)..
36     sum(i, u(i,m)*e(i)*x(i)) =L= K(m);

38 model m_shared / objdef, cons /;

40 file empinfo / '%emp.info%' /;
41 put empinfo 'equilibrium' /;
42 put 'visol cons' /;
43 loop(i,
44     put 'min', obj(i), x(i), objdef(i), 'cons' /;
45 );
46 putclose empinfo;

48 $echo SharedEqu > jams.opt
49 m_shared.optfile = 1;

51 solve m_shared using emp;

53 * Uncomment the code below to retrieve multipliers when a GNEP solution
       is computed.
54 * parameters cons_m(m,i);
55 * execute_load '%gams.scrdir%/u', cons_m=cons;
```

Note that since we only have one constraint cons in the modeling system, the lines 53-55 show how to recover the multiple values for each constraint multiplier for each agent.

2.4 Modeling equilibrium problems using shared variables

In this section, we introduce *implicit variables* and their uses as *shared variables*. Roughly speaking, the values of implicit variables are implicitly defined by other variable values. Shared variables are implicit variables whose values are shared by multiple agents. For example, state variables controlled by multiple agents, but that need to have the same values across them, could be shared variables. In this case, our framework allows a single variable to represent such shared variables. This not only improves clarity of the model and facilitates deployment of different mixed behavior models, but also provides a way of significantly improving performance with efficient formulations. In Section 2.4.1, implicit variables and shared variables are defined. Section 2.4.2 presents various MCP formulations for them. Finally, in Section 2.4.4, we present examples of using shared variables and experimental results comparing various MCP formulations.

2.4.1 Implicit variables and shared variables

Definition 2.5. *We call a variable y an implicit variable if for each x there is at most one y satisfying $(y, x) \in X$. Here the set X is called the defining constraint of variable y.*

Note that Definition 2.5 is not associated directly with equilibrium problems. It states that there exists one and only one implicit function $g(\cdot)$ such that $(g(x), x) \in X$. A simple example is $X = \{(y, x) \mid y = \sum_{i=1}^{n} x_i\}$. We do not check for uniqueness however. Our current implementation only allows the defining constraint X to be represented as a system of equations and the implicit variable y to be declared as a free variable. Constraints including

bounds on variable y can be introduced by explicitly declaring them. This is for allowing different solution types discussed in Section 2.3 to be associated with them.

Based on Definition 2.5, we define a shared variable.

Definition 2.6. *In equilibrium problems, variables y_i's are shared variables if*

- *The feasible region of agent i is given by*

$$K_i(x_{-i}) := \{(y_i, x_i) \in \mathbb{R}^{n_y \times n_i} \mid (y_i, x_i) \in X_i(x_{-i}), (y_i, x_i, x_{-i}) \in X\}, \text{ for } i = 1, \ldots, N. \tag{2.10}$$

- *y_i's are implicit variables with the same defining constraint X.*

Basically, shared variables are implicit variables with an additional condition that they have the same defining constraint. One can easily verify that if $(y_1, \ldots, y_N, x) \in K(x) := \prod_{i=1}^{N} K_i(x_{-i})$, then $y_1 = \cdots = y_N$, that is, variables y_i share their values. An extension to the case where variables are shared by some subset of agents is straightforward.

An equilibrium in the case in which shared variables y_i are present is defined as follows:

$$\begin{aligned} &\text{find} && (y^*, x_1^*, \ldots, x_N^*, x_{N+1}^*) \quad \text{satisfying,} \\ &(y^*, x_i^*) && \in \underset{(y, x_i) \in K_i(x_{-i}^*)}{\arg\min} \quad f_i(y, x_i, x_{-i}^*), && \text{for} \quad i = 1, \ldots, N, \\ &x_{N+1}^* && \in \mathrm{SOL}(K_{N+1}(x_{-(N+1)}^*), F(x_{N+1}, x_{-(N+1)}^*)). \end{aligned} \tag{2.11}$$

Example 2.7 presents the use of a shared variable assuming that y is an implicit variable with its defining constraint $X := \{(y, x) \mid H(y, x) = 0\}$.

Example 2.7. *The variable y is a shared variable of the following equilibrium problem:*

$$
\begin{array}{lll}
\textit{find} & (y^*, x_1^*, \ldots, x_N^*) & \textit{satisfying}, \\
(y^*, x_i^*) \in & \underset{y, x_i}{\arg\min} & f_i(y, x_i, x_{-i}^*), \\
& \textit{subject to} & H(y, x_i, x_{-i}^*) = 0, \ \textit{for } i = 1, \ldots, N, \\
& \textit{where} & H : \mathbb{R}^{m+n} \to \mathbb{R}^m, y \in \mathbb{R}^m
\end{array}
$$

Listing 2.11 presents GAMS code to model Example 2.7. We introduce a new keyword `implicit` to declare an implicit variable and its defining constraint. The `implicit` keyword is followed by a list of variables and constraints, and our framework augments them to form a single vector of implicit variables and its defining constraint. It is required that the keyword should come first before any agent's problem definition. We can identify that y is a shared variable in this case as it appears multiple times in agents' problem definitions. As the defining equation is assumed to belong to the implicit variable, we do not place H in each agent's problem definition (informally the variable y owns H).

Listing 2.11: Modeling a shared variable

```
1 variables obj(i), x(i), y;
2 equations deff(i), defH;

4 model shared_implicit / deff, defH /;

6 file empinfo / '%emp.info%' /;
7 put empinfo 'equilibrium' /;
```

```
put 'implicit y defH' /;
loop(i,
  put 'min', obj(i), x(i), y, deff(i) /;
);
putclose empinfo;
```

Unlike other variables, the value of a shared (more exactly, implicit) variable can be defined via its defining constraint although it is not owned by any agent. Hence we allow missing ownership of shared variables, and this is especially useful to model mixed behavior as described in Section 2.4.4. For missing ownership of shared variables, our framework treats them as a VI agent: H becomes a VI function, and y is its matching variable in Example 2.7.

As we now allow shared variables, Assumption 2.4 is modified as follows:

Assumption 2.8. *A model of an equilibrium problem described by equations and variables is assumed to have the following properties in the empinfo file:*

- *Each* VI *function of the model is owned by a single agent. Each objective function of the model is owned by at least one agent. The objective function can be owned by multiple agents when its objective variable is declared as an implicit variable.*
- *Each constraint of the model is owned by at least one agent. If a constraint appears multiple times in different agents' problem definitions, then it is regarded as a shared constraint owned by those agents.*
- *Each variable of the model is owned by at least one agent except for an implicit variable. If a variable appears multiple times in different agents' problem definition,*

then it is regarded as a shared variable owned by those agents, and it must be an implicit variable. If there is a variable not owned by any agent, then it must be an implicit variable.

2.4.2 Various MCP formulations for shared variables

This section describes various MCP formulations for equilibrium problems containing shared variables. For clarity, we will use Example 2.7 to demonstrate our formulations throughout this section. Each formulation described in Sections 2.4.2-2.4.2 shares the same GAMS code of Listing 2.11. Different formulations can be obtained by specifying an appropriate value for the option `ImplVarModel` in the file `jams.opt`. In Section 2.4.4, we present experimental results comparing the sizes and performance of these formulations.

Replicating shared variables for each agent

In this reformulation, we replicate each shared variable for each agent owning it and compute the corresponding MCP. For Example 2.7, our framework creates a variable y_i for agent i, that is a replication of variable y, then computes the KKT conditions. The following MCP(z, F) is formulated by collecting those KKT conditions:

$$
\begin{aligned}
F(z) &= \left[(F_i(z)^{\mathrm{T}})_{i=1}^{N}\right]^{\mathrm{T}}, & z &= \left[(z_i^{\mathrm{T}})_{i=1}^{N}\right]^{\mathrm{T}}, \\
F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x,y) - (\nabla_{x_i} H(y,x))\mu_i \\ \nabla_{y_i} f_i(x,y) - (\nabla_{y_i} H(y,x))\mu_i \\ H(y_i,x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ y_i \\ \mu_i \end{bmatrix}.
\end{aligned}
\tag{2.12}
$$

The size of (2.12) is $(n + 2mN)$ where the first term is from $n = \sum_{i=1}^{N} |x_i|$ and the second one is from $N \times (|y_i| + |\mu_i|)$ with $|y_i| = |\mu_i| = m$ for each $i = 1, \ldots, N$. Note that the same constraints H and shared variable y are replicated N times in the MCP form. Table 2.1 summarizes the sizes of the MCP formulations depending on the strategy. (2.12) can be obtained by specifying an option `ImplVarModel=Replication` in `jams.opt`.

Switching shared variables with multipliers

We introduce a switching strategy that can be applied when i) the defining constraint is given as an equation as in Example 2.7, ii) the dimension of its image space is the same as the shared variable, and iii) the shared variable is a free variable. The switching strategy uses the fact that in an MCP we can exchange free variables of the same size in the complementarity conditions without changing solutions. For example, if an MCP is given by

$$\begin{bmatrix} F_1(z) \\ F_2(z) \end{bmatrix} \perp \begin{bmatrix} z_1 \\ z_2 \end{bmatrix},$$

where z_i's are free variables, then a solution to the MCP is a solution to the following MCP and vice versa:

$$\begin{bmatrix} F_1(z) \\ F_2(z) \end{bmatrix} \perp \begin{bmatrix} z_2 \\ z_1 \end{bmatrix}.$$

Applying the switching technique to shared variables, we switch each shared variable with the multipliers associated with its defining equations. This is possible because each shared variable is a free variable and its defining equations are of the same size as the shared variable. As a by-product, we do not have to replicate the shared variables and their

Table 2.1: The size of the MCP of equilibrium problems containing shared variables according to the formulation strategy for Example 2.7

Strategy	Size of the MCP
replication	$(n + 2mN)$
switching	$(n + mN + m)$
substitution (implicit)	$(n + nm + m)$
substitution (explicit)	$(n + m)$

defining constraints. Thus we may be able to reduce the size of the resultant MCP.

The MCP(z, F) obtained by applying the switching technique to Example 2.7 is as follows:

$$\begin{aligned} F(z) &= \left[(F_i(z)^{\mathrm{T}})_{i=1}^{N}, F_h(z)^{\mathrm{T}}\right]^{\mathrm{T}}, & z &= \left[(z_i^{\mathrm{T}})_{i=1}^{N}, z_h^{\mathrm{T}}\right]^{\mathrm{T}}, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x,y) - (\nabla_{x_i} H(y,x))\mu_i \\ \nabla_y f_i(x,y) - (\nabla_y H(y,x))\mu_i \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \mu_i \end{bmatrix}, \\ F_h(z) &= \Big[H(y,x)\Big], & z_h &= \Big[y\Big]. \end{aligned} \tag{2.13}$$

The size of (2.13) is $(n + mN + m)$. Note that compared to the replication strategy the size is reduced by $(N - 1)m$. The number $(N - 1)m$ exactly corresponds to the number of additional replications of the shared variable y. The formulation can be obtained by specifying an option `ImplVarModel=Switching` in `jams.opt`. This is also a default value for `ImplVarModel`.

Substituting out multipliers

We can apply our last strategy when the conditions of Section 2.4.2 are satisfied, and the implicit function theorem holds for the defining constraints. By the implicit function theorem, we mean for $(\bar{y}, \bar{x})$ satisfying $H(\bar{y}, \bar{x}) = 0$ there exists a continuously differentiable

function $h : \mathbb{R}^n \to \mathbb{R}^m$ such that $H(h(x), x) = 0$ for all x in some neighborhood of $\bar{x}$. In other words, the value of y is implicitly determined by the value of x near $\bar{x}$.

In a single optimization problem with H taking the special form, $H(y, x) = y - h(x)$, a similar definition was made in the AMPL modeling system, and the variable y is called a *defined variable* in this case [39, See A.8.1].

The basic idea is to regard the shared variable y as a function of other non-shared variables and apply the total derivative. At each solution (y^*, x^*) to the problem, there exists a locally defined implicit function $h_{x^*}(x)$ such that $y^* = h_{x^*}(x^*)$ and $H(h_{x^*}(x), x) = 0$ for each x in some neighborhood of x^* by the implicit function theorem. We can then remove variable y by replacing it with the implicit function $h_{x^*}(x)$ near (y^*, x^*). Thus the objective function $f_i(x_i, x_{-i}, y)$ of agent i on the feasible set $H(y, x) = 0$ near (y^*, x^*) can be equivalently represented as $f_i(x_i, x_{-i}, h_{x^*}(x))$. Consequently, the KKT conditions near (y^*, x^*) only involve variable x:

$$\frac{d}{dx_i} f_i(x_i, x_{-i}, h_{x^*}(x)) = \nabla_{x_i} f_i(x_i, x_{-i}, h_{x^*}(x)) + \nabla_{x_i} h_{x^*}(x) \nabla_y f_i(x_i, x_{-i}, h_{x^*}(x)),$$
$$y = h_{x^*}(x),$$

where d/dx_i represents the total derivative with respect to variable x_i.

By the implicit function theorem, we have

$$\nabla_{x_i} h_{x^*}(x) = -\nabla_{x_i} H(y, x) \nabla_y H(y, x)^{-1}.$$

Therefore the KKT conditions of agent i's problem of Example 2.7 can be represented as

follows:

$$\begin{aligned} 0 &= \nabla_{x_i} f_i - \nabla_{x_i} H (\nabla_y H)^{-1} \nabla_y f_i \quad \perp \quad x_i \quad \text{free}, \quad \text{for} \quad i = 1, \ldots, N, \\ 0 &= H(y, x) \qquad\qquad\qquad\qquad\quad \perp \quad y \quad \text{free}, \end{aligned} \tag{2.14}$$

where we also applied the switching technique in Section 2.4.2.

We can derive the same formulation (2.14) from another perspective. At a solution (y^*, x^*, μ^*) to the problem, the matrix $\nabla_y H(y^*, x^*)$ is non-singular by the implicit function theorem. Thus we have

$$0 = \nabla_y f_i(x_i^*, x_{-i}^*, y^*) - (\nabla_y H(y^*, x^*))\mu_i^* \quad \Longrightarrow \quad \mu_i^* = (\nabla_y H(y^*, x^*))^{-1} \nabla_y f_i(x_i^*, x_{-i}^*, y^*). \tag{2.15}$$

We can then substitute out every occurrence of μ_i by the right-hand side of (2.15) and remove the left-hand side from consideration. The result is the formulation (2.14).

A critical issue with applying the formulation (2.14) is that in general we do not have the explicit algebraic representation of $(\nabla_y H)^{-1}$. Computing it explicitly may be quite expensive and cause numerical issues.

Instead of explicitly computing it, we introduce new variables Λ_i to replace $\nabla_{x_i} H (\nabla_y H)^{-1}$ with a system of equations:

$$\Lambda_i \nabla_y H(y, x) = \nabla_{x_i} H(y, x), \quad \text{for} \quad i = 1, \ldots, N.$$

One can easily verify that for each solution (y^*, x^*) to (2.14) there exists Λ_i^* satisfying the

following and vice versa:

$$\begin{aligned} 0 &= \nabla_{x_i} f_i - \Lambda_i \nabla_y f_i \quad \perp \quad x_i \quad \text{free}, \\ 0 &= \Lambda_i \nabla_y H - \nabla_{x_i} H \quad \perp \quad \Lambda_i \quad \text{free}, \quad \text{for} \quad i = 1, \ldots, N \\ 0 &= H(y, x) \qquad\quad \perp \quad y \quad \text{free}. \end{aligned} \tag{2.16}$$

Consequently, the following MCP(z, F) is formulated in this case:

$$\begin{aligned} F(z) &= \left[(F_i(z)^{\mathrm{T}})_{i=1}^{N}, F_h(z)^{\mathrm{T}} \right]^{\mathrm{T}}, & z &= \left[(z_i^{\mathrm{T}})_{i=1}^{N}, z_h^{\mathrm{T}} \right]^{\mathrm{T}}, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x, y) - \Lambda_i \nabla_y f_i(x, y) \\ \Lambda_i \nabla_y H(y, x) - \nabla_{x_i} H(y, x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \Lambda_i \end{bmatrix}, \\ F_h(z) &= \Big[H(y, x) \Big], & z_h &= \Big[y \Big]. \end{aligned} \tag{2.17}$$

The size of (2.17) is $(n + mn + m)$. This could be much larger than the one obtained when we apply the switching strategy, whose size is $(n + mN + m)$, because we usually have $n \gg N$. Comparing the size to the case where we replicate the implicit variables, we have $(n + nm + m) \leq (n + 2mN)$ if and only if $N \geq (n + 1)/2$.

The size of the substitution strategy can be significantly reduced when the shared variable is *explicitly* defined, that is, $H(y, x) = y - h(x)$. In this case, the algebraic representation of $(\nabla_y H)^{-1}$ is in a favorable form: an identity matrix. We do not have to introduce new variables and their corresponding system of equations, which caused the significant size increase. As we know the explicit algebraic formulation of $\nabla_{x_i} H$, the following MCP is

formulated:

$$
\begin{aligned}
F(z) &= \left[(F_i(z)^{\mathrm{T}})_{i=1}^{N}, F_h(z)^{\mathrm{T}} \right]^{\mathrm{T}}, & z &= \left[(z_i^{\mathrm{T}})_{i=1}^{N}, z_h^{\mathrm{T}} \right]^{\mathrm{T}}, \\
F_i(z) &= \left[\nabla_{x_i} f_i(x,y) - \nabla_{x_i} H(y,x) \nabla_y f_i(x,y) \right], & z_i &= \left[x_i \right], \\
F_h(z) &= \left[H(y,x) \right], & z_h &= \left[y \right].
\end{aligned}
\tag{2.18}
$$

Note that the size of (2.18) is $(n + m)$. This is a huge saving compared to other formulations. Our framework automatically detects if a shared variable is given in the explicit form and substitutes out the multipliers if it is. Otherwise, (2.17) is formulated. The formulation can be obtained by specifying an option `ImplVarModel=Substitution` in `jams.opt`.

2.4.3 Exploiting the structure of shared variables

We present how Path, our back-end solver for the framework, can exploit the structure of the problem associated with shared variables when the implicit function theorem holds. We exploit the fact that the values of shared variables and their associated multipliers are uniquely determined by the implicit function theorem. Extensions to the cases where there are more than one implicit variable are straightforward.

Suppose that Path solves one of the MCPs presented in Section 2.4.2. As Path does not maintain feasibility during iterations, each iterate (x^k, y^k, μ^k) may be highly infeasible. This can in turn lead to a numerically unstable and longer path to a solution. Especially, the main infeasibility often comes from (y^k, μ^k): it does not satisfy $H(y^k, x^k) = 0$ and the system of equations, either (2.15) or the second equation of (2.16). We may restore

the feasibility, say $(\tilde{y}^k, \tilde{\mu}^k)$, by exploiting the uniqueness property of the implicit function theorem. In many cases, this new point $(x^k, \tilde{y}^k, \tilde{\mu}^k)$ has much smaller residual, that is, $\phi(x^k, \tilde{y}^k, \tilde{\mu}^k) \ll \phi(x^k, y^k, \mu^k)$, with $\phi(x, y, \mu)$ being a merit function. This will lead PATH to a more robust and shorter path to a solution.

We therefore introduce *spacer steps* such that for each given (x^k, y^k, μ^k) we compute a unique feasible pair $(\tilde{y}^k, \tilde{\mu}^k)$, evaluate the residual at that $(x^k, \tilde{y}^k, \tilde{\mu}^k)$, and choose the point if it has smaller merit function value than the one of (x^k, y^k, μ^k). With those spacer steps, we were able to find a solution in a more numerically stable fashion and with fewer iterations as reported in Table 2.5.

2.4.4 Examples

In this section, we introduce three models that use shared variables. Section 2.4.4 describes an example where we can improve its sparsity significantly by introducing a shared variable. This enables the problem, previously known as computationally intractable, to be efficiently solved. Section 2.4.4 presents an EPEC model where each agent tries to maximize its welfare in the Nash way while trading goods with other agents subject to general equilibrium conditions. The general equilibrium conditions define a set of state variables that are shared by all the agents. We can then use the constructs for shared variables to define the model. In Section 2.4.4, we present an example of modeling mixed pricing behavior of agents. More examples on using shared variables for shared objective functions can be found at [27]. All experiments were performed on a Linux machine with Intel(R) Core(TM) i5-3340M CPU@2.70 GHz processor and 8GB of memory. PATH was set to use the UMFPACK [18] as its basis computation engine.

Improving sparsity using a shared variable

We consider an oligopolistic energy market equilibrium example [55, Section 4] formulated as a GNEP. We show that its sparsity can be significantly improved by introducing a shared variable, which makes the problem, known as computationally intractable in [55], solvable. The example is defined as follows:

$$
\begin{aligned}
&\text{find} \quad (q_0^*, q_1^*, \ldots, q_5^*) \quad \text{satisfying,} \\
&q_0^* \in \underset{0 \le q_0 \le U_0}{\arg\max} \quad p\left(\sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^*\right)\left(\sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^*\right) - \sum_{i=1}^{5} c_i(q_i^*) - Pq_0, \\
&\qquad \text{subject to} \quad q_0 + \sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^* = d, \\
&q_i^* \in \underset{0 \le q_i \le U_i}{\arg\max} \quad p\left(\sum_{j=1, j\neq i}^{5}\sum_{k=1}^{n_j} q_{jk}^* + \sum_{k=1}^{n_i} q_{ik}\right)\sum_{k=1}^{n_i} q_{ik} - c_i(q_i), \\
&\qquad \text{subject to} \quad q_0^* + \sum_{j=1, j\neq i}^{5}\sum_{k=1}^{n_j} q_{jk}^* + \sum_{k=1}^{n_i} q_{ik} = d, \\
&\qquad \text{where} \quad c_i(q_i) = \frac{1}{2} q_i^{\mathrm{T}} M_i q_i + b_i^{\mathrm{T}} q_i, \\
&\qquad\qquad\quad p(Q) := \left(\frac{-P}{(1.5d)^2} Q^2 + P\right), \\
&\qquad\qquad\quad (P, d, M_i, b_i, U_i, n_i) \text{ is problem data,} \quad \text{for} \quad i = 1, \ldots, 5.
\end{aligned}
\tag{2.19}
$$

Let us briefly describe (2.19). There are six agents. The first agent is an ISO agent which controls variable $q_0 \in \mathbb{R}$ measuring deficit of energy. It tries to maximize the total profit of all the energy supplying agents less the penalty caused by being unable to meet the fixed demand d. The parameter P represents how much penalty we put on the deficit q_0. Each agent i, controlling $q_i = (q_{i1}, \ldots, q_{in_i})$ for $i = 1, \ldots, 5$, is a profit-maximizing agent

that produces homogeneous energy generated from its n_i number of plants. Its decision variable q_{ik} denotes the amount of energy produced from its kth plant for $k = 1, \ldots, n_i$. The function $p(Q)$ is a concave inverse demand function, and $c_i(q_i)$ is the total cost of producing energy $\sum_{k=1}^{n_i} q_{ik}$. The matrix M_i is a diagonal matrix having positive diagonal entries, hence $c_i(\cdot)$ is a strongly convex function. All the six agents share the same demand constraint $q_0 + \sum_{i=1}^{5} \sum_{k=1}^{n_i} q_{ik} = d$; it is a shared constraint. We use n, $n = \sum_{i=1}^{5} n_i$, to denote the total number of plants, and each energy-producing agent has the same number of plants, $n_i = n/5$ for $i = 1, \ldots, 5$.

In [55], a variational equilibrium was computed by formulating a VI and solving it using Path. The paper reported that Path started to get much slower for the problem of size $n = 2,500$, and it was not able to solve problems of sizes $n = 5,000$ and $n = 10,000$ due to out of memory error.

We have observed that the memory error was due to the high density of the Jacobian matrix of the MCP: it was almost 100% for all problems. Consequently, the MCP will have a large number of nonzero entries requiring a huge amount of memory. Also the linear algebra computation (required by Path for basis computations) time will be much slower in this case.

The root cause of such a highly dense Jacobian matrix was because of the term $\sum_{i=1}^{5} \sum_{k=1}^{n_i} q_{ik}$ in the price function $p(\cdot)$: for each q_{ik}, the term $\partial p(\cdot)/\partial q_{ik}$ has all the variables $q_{i'k'}$. We can make the problem much sparser by introducing a shared variable $z := \sum_{i=1}^{5} \sum_{k=1}^{n_i} q_{ik}$ as implemented in Listing 2.12. We also used the `visol` keyword to compute a variational equilibrium. We formulate each agent's problem as a minimization problem by flipping the sign of its objective function. Therefore, each agent i's objective

function for $i = 1, \ldots, 5$ is strongly convex, and the ISO agent's objective function is linear.

Listing 2.12: Implementation of (2.19) using a shared variable within GAMS/EMP

```
$if not set n $set n 100
$if not set num_agents $set num_agents 5
$eval num_plants %n%/%num_agents%
$set P 120

sets i      / 1*%num_agents% /
     k      / 1*%num_plants% /;

alias(i,j);

variables iso_obj, agent_obj(i), z;
positive variables  q0, q(i,k);
equations iso_defobj, agent_defobj(i), demand, defz;
parameters U0, U(i,k), M(i,k), b(i,k), d, a;

U0 = 5;
U(i,k) = uniform(0,10);
M(i,k) = uniform(0.4,0.8);
b(i,k) = uniform(30,60);
d = 0.8 * sum((i,k), U(i,k));
a = -%P% / (1.5 * d)**2;

q0.up = U0;
q.up(i,k) = U(i,k);
q.l(i,k) = 0.8*U(i,k);
```

```
26 z.l = sum((i,k), q.l(i,k));

28 iso_defobj..
29     iso_obj =E= %P%*q0
30     + sum(i, 0.5*sum(k, M(i,k)*q(i,k)*q(i,k)) + sum(k, b(i,k)*q(i,k)))
31     - (a*sqr(z) + %P%)*z;

33 agent_defobj(i)..
34     agent_obj(i) =E=
35     0.5*sum(k, M(i,k)*q(i,k)*q(i,k)) + sum(k, b(i,k)*q(i,k))
36     - (a*sqr(z) + %P%)*sum(k, q(i,k));

38 demand..
39     q0 + z =E= d;

41 defz..
42     z =E= sum((i,k), q(i,k));

44 model m_oligop / iso_defobj, agent_defobj, demand, defz /;

46 file empinfo / '%emp.info%' /;
47 put empinfo 'equilibrium' /;
48 put 'implicit z defz' /;
49 put 'visol demand' /;
50 put 'min', iso_obj, q0, iso_defobj, demand /;
51 loop(i,
52     put 'min', agent_obj(i);
53     loop(k, put q(i,k);));
```

Size (n)	Original		Switching		Substitution	
	Size	Density (%)	Size	Density (%)	Size	Density (%)
2,500	2,502	99.92	2,508	0.20	2,503	20.07
5,000	5,002	99.96	5,008	0.10	5,003	20.04
10,000	10,002	99.98	10,008	0.05	10,003	20.02
25,000	-	-	25,008	0.02	-	-
50,000	-	-	50,008	0.01	-	-

(a) MCP model statistics when we have 1 ISO agent and 5 energy-producing agents

Size (n)	Original		Switching		Substitution	
	(Major,Minor)	Time (secs)	(Major,Minor)	Time (secs)	(Major,Minor)	Time (secs)
2,500	(2,2639)	57.78	(1,2630)	1.30	(1,2630)	13.18
5,000	(2,5368)	420.92	(1,5353)	5.83	(1,5353)	91.01
10,000	-	-	(1,10517)	22.01	(1,10517)	652.03
25,000	-	-	(1,26408)	148.08	-	-
50,000	-	-	(1,52946)	651.14	-	-

(b) Performance comparison when we have 1 ISO agent and 5 energy-producing agents

Table 2.2: Model statistics and performance comparison of (2.19) using PATH

```
54      put z, agent_defobj(i), demand /;
55  );
56  putclose empinfo;

58  solve m_oligop using emp;
```

Tables 2.2 and 2.3 describe the statistics and performance of (2.19) over various sizes of plants and agents. The '-' symbol represents that we were not able to obtain the results because of memory issue. In Table 2.2, we used the same setup as in [55]. First, note that the MCP size of the original formulation was the smallest, but it had the highest density. This resulted in a computationally intractable model for large $n \geq 10,000$. In contrast, using a shared variable and the switching strategy, we were able to generate much sparser models and consequently to solve all of them. However, the substitution strategy suffered a

Size (n)	Switching		Substitution	
	Size	Density (%)	Size	Density (%)
2,500	3,753	0.12	2,503	0.20
5,000	7,503	0.06	5,003	0.10
10,000	15,003	0.03	10,003	0.05
25,000	37,503	0.01	25,003	0.02
50,000	75,003	0.01	50,003	0.01

(a) MCP model statistics when we have 1 ISO agent and $n/2$ energy-producing agents

Size (n)	Switching		Substitution	
	(Major,Minor)	Time (secs)	(Major,Minor)	Time (secs)
2,500	(1,2650)	1.43	(1,2650)	0.88
5,000	(1,5359)	5.89	(1,5359)	3.61
10,000	(1,10526)	25.05	(1,10526)	15.70
25,000	(1,26400)	176.94	(1,26400)	107.45
50,000	(1,52950)	800.75	(1,52950)	471.51

(b) Performance comparison when we have 1 ISO agent and $n/2$ energy-producing agents

Table 2.3: Model statistics and performance comparison with $n/2$ energy-producing agents using PATH

similar issue: its high density generated computationally intractable models for $n = 25,000$ and $50,000$. This was due to the total derivative computation. The term $\sum_{ik} q_{ik}$ remained in each component of the MCP function $F_i \in \mathbb{R}^{n_i}$ for each agent i. This resulted in a block diagonal Jacobian matrix consisting of 5 100% dense blocks of size $n_i \times n_i$ for $i = 1, \ldots, 5$.

To see the effect of many agents, we generated problems where each agent now has 2 plants. Thus for a given n there are $n/2$ number of energy-producing agents. Table 2.3 reports the model statistics and performance comparison of the switching and substitution strategies. We did not report experimental results using the original formulation as the MCP size and the density of its Jacobian matrix were the same as before. In this case, the substitution strategy showed the best performance. Its Jacobian matrix was still block diagonal consisting of $n/2$ blocks, but each block size was just 2×2. This improved the sparsity of the model significantly. The MCP size of the switching strategy was much

larger than that of the substitution as its size is proportional to the number of agents (see Table 2.1). This made the strategy two times slower than the substitution strategy.

Modeling equilibrium problems with equilibrium constraints

We construct an EPEC model[2] where data was taken from the GTAP (Global Trade Analysis Project) 9 database [4]. The model is an exchange model having 23 agents (countries) where each agent tries to maximize its welfare with respect to economic variables (equivalently, state variables) and its strategic policy variables in the Nash way while trading goods with other agents subject to the general equilibrium conditions. Mathematically, the model is represented as follows:

$$\begin{array}{lll} \text{find} & (w^*, z^*, t^*) & \text{satisfying,} \\ (w^*, z^*, t_i^*) \in & \underset{w,z,t_i \in T_i}{\arg\max} & w_i, \\ & \text{subject to} & H(w, z, t) = 0, \\ & & \text{for} \quad i = 1, \ldots, 23, \end{array} \tag{2.20}$$

where w_i is a welfare index variable of agent i, z is a vector of endogenous economic variables such as prices, quantities, and so on, t_i represents a vector of strategic policy variables of agent i that determine the tariffs on the imports, and $H(\cdot) : \mathbb{R}^{253\times 506} \to \mathbb{R}^{253}$ is a system of nonlinear equations that represents the general equilibrium conditions.

A distinguishing feature of the model is that the state variables (w, z) are shared by the agents, and their values are implicitly determined by the general equilibrium conditions.

[2]The original model was written by Thomas Rutherford, and was solved by applying the diagonalization method (Gauss-Seidel) to the nonlinear problem (2.20) by fixing t variable values belonging to other agents. We modified the model to use our EMP framework, and it was subsequently solved by PATH.

# Agents	Replication		Switching		Substitution	
	Size	Density (%)	Size	Density (%)	Size	Density (%)
5	570	1.66	350	3.34	1,230	0.77
10	2,290	0.72	1,300	1.70	10,210	0.14
15	5,160	0.50	2,850	1.28	35,190	0.06
20	9,180	0.40	5,000	1.10	84,420	0.03
23	12,144	0.37	6,578	1.03	129,030	0.02

# Agents	Replication		Switching		Substitution	
	(Major,Minor)	Time (secs)	(Major,Minor)	Time (secs)	(Major,Minor)	Time (secs)
5	(18,164)	0.33	(18,173)	0.22	(11,29)	0.38
10	(17,279)	1.52	(17,301)	1.48	(15,436)	8.14
15	(8,22)	1.81	(8,22)	1.68	(129,19806)	814.73
20	(9,28)	4.90	(9,28)	4.73	(13,461)	104.00
23	(9,41)	10.07	(9,41)	8.02	(20,1451)	368.99

Table 2.4: MCP model statistics and performance comparison of the EPEC model

This implies that (w, z) are shared variables, and the function H is their defining constraint. In this case, (w, z) are not given as an explicit function of t in H.

In Table 2.4, we present experimental results of the three formulations over various problem sizes by changing the number of agents. The size of H changes accordingly. We use the replication strategy as a baseline to compare the size and performance of the MCP models. We do not describe the implementation within GAMS/EMP as the number of lines is long. Refer to [27] for the implementation.

In all settings, the switching strategy generated the smallest MCP as it did not replicate or create variables and equations. Also its Jacobian matrix was quite sparse. Consequently, it showed the best performance in terms of the elapsed time: it was up to 6 times faster than the replication strategy and 50 times than the substitution strategy. We did not include the 15-agent problem in the comparison as we think the slowest performance of the substitution strategy is due to some numerical difficulties PATH encountered. Although it performed more number of iterations on the problem having 10 agents, its time was still faster than

# Agents	Substitution	
	(Major,Minor)	Time (secs)
5	(12,49)	1.09
10	(7,18)	8.97
15	(7,21)	45.94
20	(7,26)	141.77
23	(7,29)	263.30

Table 2.5: Performance of the substitution strategy with spacer step

that of the replication strategy. We believe that the smaller problem size led to faster linear algebra computation.

The substitution strategy was of the largest problem size and showed the slowest elapsed time. The large size was due to the newly introduced variables and equations as described in Table 2.1. Although the density of it was the smallest, the number of nonzero entries was the largest. Hence linear algebra computation became much slower.

The numerical difficulties PATH encountered when we used the substitution strategy can be avoided by using the spacer steps described in Section 2.4.3. The main reason for such difficulties was the high infeasibility of the newly constructed equations and variables. By projecting them into feasible region, we were able to compute a solution with much fewer iterations and in a more robust way as reported in Table 2.5.

Modeling mixed behavior: price-taking and price-making agents

In this example, we show that mixed behavior of firms, switching between price-takers and price-makers, can be easily modeled using a shared variable. We revisit the oligopolistic market equilibrium problem in Section 2.2.2. Previously, the market was an oligopolistic market where all the firms were price-makers: they can directly affect the price by changing their productions. If they have no control over the price, they become price-takers, that

is, the price is an exogenous variable for each firm. In this case, the market is perfect competitive.

Listing 2.13 implements our mixed behavior model. We introduce an implicit variable z that represents the price $p(Q)$ defined in (2.3). If a firm has ownership of variable z, then it becomes a price-maker as it has a direct control of it. Otherwise, it is a price-taker. The first solve on line 36 computes a competitive market equilibrium. As no firms have ownership of variable z, they are all price-takers in this case. After the first solve, we compute five different mixed models where firms having indices less than or equal to j are price-makers at the jth mixed model for $j = 1, \ldots, 5$.

Listing 2.13: Implementation of mixed behavior of agents within GAMS/EMP

```
1  sets i agents / 1*5 /;
2  alias(i,j);

4  parameters
5      c(i)     / 1  10, 2   8, 3   6, 4   4, 5   2 /
6      K(i)     / 1   5, 2   5, 3   5, 4   5, 5   5 /
7      beta(i) / 1 1.2, 2 1.1, 3 1.0, 4 0.9, 5 0.8 /
8      ;

10 variables obj(i), z;
11 positive variables q(i);

13 equations
14     objdef(i),
15     zdef
16     ;
```

```
18 objdef(i)..
19     obj(i) =e= q(i)*z - (c(i)*q(i) + beta(i)/(beta(i)+1)*K(i)**(-1/beta
           (i))*q(i)**((beta(i)+1)/beta(i)));

21 zdef..
22     z =e= 5000**(1.0/1.1)*sum(i, q(i))**(-1.0/1.1);

24 model mixed / objdef, zdef /;

26 file empinfo / '%emp.info%' /;
27 put empinfo 'equilibrium' /;
28 put 'implicit', z, zdef /;
29 loop(i,
30     put 'max ', obj(i), q(i), objdef(i) /;
31 );
32 putclose empinfo;

34 q.l(i) = 10;
35 z.l = sum(i, q.l(i));

37 solve mixed using emp;

39 parameter objval(i,*), qval(i,*), pval(*), totalobjval(*),
       socialwelfare(*);

41 objval(i,'competitive') = obj.l(i);
42 qval(i,'competitive') = q.l(i);
```

```
43 pval('competitive') = z.l;
44 totalobjval('competitive') = sum(i, objval(i,'competitive'));
45 socialwelfare('competitive') = (5000**(1.0/1.1)*11*sum(i, q.l(i))
       **(0.1/1.1)
46     - z.l*sum(i, q.l(i))) + totalobjval('competitive');

48 set kind / oligo1, oligo12, oligo123, oligo1234, oligo12345 /;

50 loop(kind,
51     put empinfo 'equilibrium' /;
52     put 'implicit', z, zdef /;
53     loop(i,
54         if (i.val le ord(kind),
55             put 'max ', obj(i), q(i), z, objdef(i) /;
56         else
57             put 'max ', obj(i), q(i), objdef(i) /;
58         );
59     );
60     putclose empinfo;

62     q.l(i) = 10;
63     z.l = sum(i, q.l(i));
64     solve mixed using emp;

66     objval(i,kind) = obj.l(i);
67     qval(i,kind) = q.l(i);
68     pval(kind) = z.l;
69     totalobjval(kind) = sum(i, objval(i,kind));
```

```
70      socialwelfare(kind) = (5000**(1.0/1.1)*11*sum(i, q.l(i))**(0.1/1.1)
71          - z.l*sum(i, q.l(i))) + totalobjval(kind);
72  );

74  option pval:3:0:1, totalobjval:3:0:1, socialwelfare:3:0:1;
75  display objval, qval, pval, totalobjval, socialwelfare;
```

Table 2.6 presents profits of the firms and social welfare of various mixed models. We computed social welfare by adding the consumer surplus to the total profit of the firms. The consumer surplus was computed by integrating the inverse demand function less the amount paid by the consumer. In columns starting with "Oligo", indices of firms that are price-makers are attached to it. Thus Oligo123 implies that firms with indices between 1 and 3 are price-makers, and others are price-takers. As expected, i) the total profit of the firms was the smallest in the competitive case and the largest in the oligopolistic case; ii) each firm made more profit as it switched from price-taker to price-maker; iii) the social welfare was the maximized when all firms were price-takers. Interestingly, switching from a price-taker to a price-maker of a firm made profits of other firms increase much larger than the one of itself. Similar observation was made in [43] and was explained as an externality effect.

2.5 Modeling quasi-variational inequalities

This section introduces a new construct for specifying QVIs within our framework and presents an example comparing two equivalent ways of defining the equilibrium problems in either GNEP or QVI form.

Profit	Competitive	Oligo1	Oligo12	Oligo123	Oligo1234	Oligo12345
Firm 1	123.834	125.513	145.591	167.015	185.958	199.934
Firm 2	195.314	216.446	219.632	243.593	264.469	279.716
Firm 3	257.807	278.984	306.174	309.986	331.189	346.590
Firm 4	302.863	322.512	347.477	373.457	376.697	391.279
Firm 5	327.591	344.819	366.543	388.972	408.308	410.357
Total profit	1207.410	1288.273	1385.417	1483.023	1566.621	1627.875
Social welfare	39063.824	39050.191	39034.577	39022.469	39016.373	39015.125

Table 2.6: Profits of the firms and social welfare of various mixed models of Listing 2.13

2.5.1 Specifying quasi-variational inequalities using our framework

Assuming that the feasible region of a QVI(K, F) takes the form $K(x) := \{y \in \mathbb{R}^n \mid h(y, x) = 0, g(y, x) \leq 0\}$, Listing 2.15 shows generic way of specifying the QVI(K, F) using our framework. In this case, we call x a parameter variable and y a variable of interest. Parameter variables could appear in the constraints, however, the QVI function F must be defined by only variables of interest.

Listing 2.14: Modeling the QVI

```
1  variables x(i), y(i);
2  equations defF(i), defh, defg;

4  * Definitions of defF(i), defh, and defg are omitted for expository
       purposes.

6  model qvi / defF, defh, defg /;

8  file empinfo / '%emp.info%' /;
9  putclose empinfo 'qvi defF y x defh defg';

11 solve qvi using emp;
```

To specify QVIs, the `empinfo` file starts with a new keyword `qvi`. The syntax is similar to the one for VIs as described in Section 2.2.1 except that additional variables could follow right after each function-variable pair. In this case, those additional variables become parameter variables, and the size of them must be the same as the size of variables of interest in the preceding pair. Our framework then constructs matching information between parameter variables and variables of interest. The same applies to each preceding variable that is assigned to a zero function. Therefore, in Listing 2.15, variables y and x are the variable of interest and the parameter variable, respectively, and each x_i is matched with y_i. When our framework formulates the corresponding MCP, for each constraint it takes the derivative with respect to y, and each occurrence of x_i is replaced with y_i using the matching information. Note that if there are no parameter variables, that is, no variables follow each function-variable pair and each preceding variable, then the problem becomes a VI. In this case, the feasible region is a fixed set, $K(x) := K$.

2.5.2 Example

We consider the following QVI(K, F) example in [62, page 14]:

$$\begin{aligned} F(y) &= \begin{bmatrix} -\frac{100}{3} + 2y_1 + \frac{8}{3}y_2 \\ -22.5 + \frac{5}{4}y_1 + 2y_2 \end{bmatrix}, \\ K(x) &= \{0 \le y \le 11 \mid y_1 + x_2 \le 15, x_1 + y_2 \le 20\} \end{aligned} \tag{2.21}$$

Listing 2.15 describes an implementation of (2.21). As in (2.21), we use x as a parameter

variable in the implementation. The implementation is a natural translation of its algebraic form so that users can focus on the QVI specification itself. Also the `empinfo` file retains information about variable types so that we can easily identify which variables are parameter variables and which are variables of interest. This information can be potentially exploited for the efficient implementation of solution methods for QVIs. Our framework computes a solution $x^* = (10, 5)$ that is consistent with the one reported in [62].

Listing 2.15: Implementation of (2.21) within GAMS/EMP

```
1  sets i / 1*2 /;
2  alias(i,j);

4  parameter A(i,j);
5  A('1','1') = 2;
6  A('1','2') = 8/3;
7  A('2','1') = 5/4;
8  A('2','2') = 2;

10 parameter b(i);
11 b('1') = 100/3;
12 b('2') = 22.5;

14 parameter Cy(i,j), Cx(i,j);
15 Cy(i,j)$(sameas(i,j)) = 1;
16 Cx(i,j)$(not sameas(i,j)) = 1;

18 parameter rhs(i) / 1 15, 2 20 /;
```

```
20 variables y(j), x(j);
21 equations F(i), g(i);

23 F(i)..
24     sum(j, A(i,j)*y(j)) - b(i) =N= 0;

26 g(i)..
27     sum(j, Cy(i,j)*y(j)) + sum(j, Cx(i,j)*x(j)) - rhs(i) =L= 0;

29 model qvi / F, g /;

31 file empinfo / '%emp.info%' /;
32 putclose empinfo 'qvi F y x g';

34 * If bounds on y and x are different, then an intersection of them is
       taken.
35 y.lo(j) = 0; y.up(j) = 11;
36 x.lo(j) = 0; x.up(j) = 11;

38 solve qvi using emp;
```

One can easily check that the QVI (2.21) is equivalent to the GNEP (2.4) in Section 2.2.2 in terms of solutions. Actually, all the equilibrium examples described in previous sections can be equivalently formulated as QVIs in the manner of Proposition 1.1.

However, the information provided to our framework could be different depending on the formulations. The GNEP formulation (2.4) gives us each agent's information: for example its objective function and ownership of variables and constraints. It may not be

easy to recover this information from the QVI formulation. In general, we can collect more information from an equilibrium formulation. This could result in different solutions methods such as spacer steps for the PATH solver or a Gauss-Seidel method and its variants, while it may not be possible to collect similar information from the QVI formulation. Therefore, for equilibrium problems, it may be better to not use the QVI formulation. Since our QVI framework is not just limited to QVIs derived from equilibrium problems, it can be used to explicitly model other types of QVIs with possible specialized algorithms for solution.

2.6 Conclusions

We have presented an extended mathematical programming framework for equilibrium programming. The framework defines a new set of constructs that enable equilibrium problems with shared constraints and shared variables and their variational forms to be specified in modeling languages. Its syntax is a natural translation of the corresponding algebraic formulation of the problem that captures high-level structure. This allows more readable and less error prone models to be specified compared to the traditional complementarity based models that require the derivative computation of the Lagrangian. Different solution types such as variational equilibria associated with shared constraints can be easily specified and computed using our framework. We define shared variables and their associated constructs that can be used to model sparse formulations, some forms of EPECs, price-taking and price-making agents, shared objective functions, and so on. Shared variables where the implicit function theorem holds enable back-end solvers to exploit the problem structure

for more robust performance and faster computation time. We introduce a new construct for specifying QVIs.

There is potential for future work. Using the high-level information captured by our framework, we can design decomposition algorithms to solve large-scale equilibrium problems that may involve a huge number of agents. We intend to allow implicit variables defined using nonsmooth equations [69]. We plan to extend our framework to incorporate equilibrium problems including agents solving stochastic programs, bilevel programming, other forms of EPECs, all with consideration of shared constraints and shared variables, and to implement EMP in other modeling systems such as AMPL and Julia.

Chapter 3

A structure-preserving pivotal method for affine variational inequalities

Affine variational inequalities (AVI) are an important problem class that subsumes systems of linear equations, linear complementarity problems and optimality conditions for quadratic programs. This chapter describes PathAVI, a structure-preserving pivotal approach, that can efficiently process (solve or determine infeasible) large-scale sparse instances of the problem with theoretical guarantees and at high accuracy. PathAVI implements a strategy known to process models with good theoretical properties without reducing the problem to specialized forms, since such reductions may destroy sparsity in the models and can lead to very long computational times. We demonstrate formally that PathAVI implicitly follows the theoretically sound iteration paths, and can be implemented in a large scale setting using existing sparse linear algebra and linear programming techniques without employing a reduction. We also extend the class of problems that PathAVI can process. The chapter

illustrates the effectiveness of our approach by comparison to the PATH solver used on a complementarity reformulation of the AVI in the context of applications in friction contact and Nash Equilibria. PATHAVI is a general purpose solver, and freely available under the same conditions as PATH.

3.1 Introduction

In this chapter, we present PATHAVI, a structure-preserving pivotal method for affine variational inequalities (AVIs) in $\mathbb{R}^n$. An AVI(C, q, M) is defined as follows: given a polyhedral convex set C, find $z \in C$ such that

$$\langle Mz + q, y - z\rangle \geq 0, \quad \forall y \in C, \tag{AVI}$$

where $M \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$ and $\langle\cdot,\cdot\rangle$ is the usual Euclidean inner product. An AVI is a linear generalized equations [66] and we refer to [30] for results on existence, uniqueness, and stability theory for such systems.

PATHAVI tries to solve an AVI(C, q, M) by computing a zero of the normal map [67] associated with the AVI. The normal map $M_C : \mathbb{R}^n \to \mathbb{R}^n$ is defined as follows:

$$M_C(x) \coloneqq M(\pi_C(x)) + q + x - \pi_C(x), \tag{normal map}$$

with $\pi_C(\cdot)$ denoting the Euclidean projector onto the set C. One can easily see that $M_C(x^*) = 0$ if and only if $z^* = \pi_C(x^*)$ where $x^* = z^* - (Mz^* + q)$ is a solution to the AVI(C, q, M). To compute a zero of $M_C(x)$, our method employs the complementary pivoting method

[24, 53] with a ray start: the piecewise-linear (PL) map $G_C : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ is defined as

$$G_C(x, t) \coloneqq M_C(x) - tr, \tag{3.1}$$

with $r \in \mathbb{R}^n$ denoting a covering vector and t an auxiliary variable. A path defined as $G_C^{-1}(0)$ is followed through complementary pivoting. The algorithm terminates when either t becomes zero (a solution to the AVI is found) or a secondary ray is generated. Under some additional assumptions this latter outcome can be interpreted in terms of the feasibility of the AVI.

The main challenge in applying the complementary pivoting method lies in the starting phase. For good theoretical properties, a ray start is required, and it is well-defined at an extreme point. However, when C contains lines there is no extreme point. To tackle this case, the previous approach [14] performs a reduction, transforming the given AVI(C, q, M) to a reduced AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ to eliminate lines in C so that an extreme point is found in $\tilde{C}$, and it solves the reduced AVI. A similar approach of factoring out lines in C is used in [67, Proposition 4.1] to show a Lipschitzian homeomorphism of the normal map M_C.

A critical disadvantage of solving the reduced AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ is that we may lose the original structure in C and M. The matrix $\tilde{M}$ is constructed from a Schur complement computation and the polyhedral constraints defining $\tilde{C}$ are computed by multiplying with orthonormal matrices. In particular, if the original AVI is sparse, there is no guarantee that the resulting reduced AVI would enjoy the same property. We provide an instance where this happens in Section 3.6.2. In sharp contrast, PathAVI does not require any reduction at all. Therefore, our method is able to take advantage of a sparse structure, whereas the method in [14] often needs to perform dense linear algebra computations.

To perform a ray start in the case where there is no extreme point, PathAVI finds an *implicit extreme point* which generalizes the notion of an extreme point when the underlying feasible region contains lines. Roughly speaking, if we project an implicit extreme point of C on the subset where all lines are removed, we obtain an extreme point. We show that there is an implicit extreme point satisfying the sufficient conditions for a ray start. We explain how phase 1 of the simplex method can be used to find such a point.

We show that PathAVI can process an AVI(C, q, M) whenever M is an L-matrix with respect to the recession cone of C [14, Definition 4.2]. We also exhibit two new classes of AVI where PathAVI finds a solution. The first one stems from the study of friction contact problems from an AVI perspective, and the second one can be seen as a generalization of a known existence result for LCP for copositive matrices. In contrast with the previous results in [14], the conditions involve both M and q.

A widely used method for solving an AVI is the Path solver [19], which is considered one of the most robust and efficient solvers for mixed complementarity problems (MCPs). It is well known [21, 30] that an AVI can be reformulated as a linear MCP, and Path uses this approach when it solves an AVI. However, the MCP reformulation does not exploit the polyhedral structure of the set C, in that complementary pivoting of Path is done over a different PL-manifold from PathAVI's. We compare theoretical properties of the two formulations, and present computational results showing improved performance of PathAVI.

This chapter is organized as follows. In Section 3.2, we briefly describe how the complementary pivoting method on a PL-manifold computes a zero of the normal map associated with a given AVI. Section 3.3 presents our main theoretical results: firstly, we

discuss sufficient conditions for a ray start, we define the notion of an implicit extreme point, and prove the existence of such a point satisfying the conditions for a ray start. Secondly, we show that PathAVI can process L-matrices and we show new types of AVIs processable by PathAVI. In Section 3.4, we present the computational procedure to start PathAVI. Section 3.5 introduces the MCP reformulation of the AVI and analyzes worst-case performance of the two formulations. We present computational results in Section 3.6, and Section 3.8 concludes this chapter.

A word about our notation is in order. Let S be a convex set in $\mathbb{R}^n$. The lineality space of S is denoted by $\operatorname{lin} S$. The symbol $\operatorname{ri} S$ denotes the relative interior of S. The affine hull of S is denoted by $\operatorname{aff} S$. By $\operatorname{par} S$, we mean the subspace parallel to $\operatorname{aff} S$ such that $\operatorname{aff} S = s + \operatorname{par} S$ for each $s \in S$. The identity matrix in $\mathbb{R}^n$ is denoted by I_n and the zero vector is 0_n. When ordered index sets are used as subscripts on a matrix, they define a submatrix: for ordered index sets $\alpha \subset \{1, \ldots, m\}$ and $\beta \subset \{1, \ldots, n\}$ $M_{\alpha\beta}$ denotes a submatrix of M consisting of rows and columns of M in the order of α and β, respectively. When matrices are used as subscripts on a matrix, they define another matrix: for matrices Q and $\bar{Q}$ having appropriate dimensions $M_{Q\bar{Q}}$ denotes $Q^T M \bar{Q}$. For an AVI(C, q, M), C is assumed to be the set $\{z \in \mathbb{R}^n \mid Az - b \in K, l \leq z \leq u\}$ with $l_j, u_j \in \mathbb{R} \cup \{-\infty, \infty\}, b_i \in \mathbb{R}$, $A_{i\bullet} \neq 0$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$, and the set K is a Cartesian product of $\mathbb{R}_+$, $\{0\}$, or $\mathbb{R}_-$ to accommodate constraints of the form $\geq$, $=$, or $\leq$, respectively. For a closed convex cone K, the dual cone of K is denoted by $K^D := \{y \mid \langle y, k\rangle \geq 0, \forall k \in K\}$. For the rest of this chapter, Q and $\bar{Q}$ denote orthonormal basis matrices for the lineality space of C and its orthogonal complement, respectively.

3.2 Background

In this section, we briefly describe how to compute a zero of the normal map associated with a given AVI(C, q, M) using the complementary pivoting method with a ray start. We also introduce some concepts related to processability of AVIs. The reader is referred to [14, 24, 53, 67] for more details.

The basic procedure of the complementary pivoting method to compute a zero of the normal map associated with an AVI(C, q, M) is as follows: i) compute an initial solution (x^0, t^0) such that $G_C(x^0, t^0) = 0$, and the point (x^0, t^0) lies on a ray, called a starting ray, consisting of points $(x(t), t)$ with $G_C(x(t), t) = 0$ and $\pi_C(x(t)) = \pi_C(x^0)$ for all $t \geq t^0$; then ii) starting from (x^0, t^0) follow a path $G_C^{-1}(0) = \{(x, t) \in \mathbb{R}^n \times \mathbb{R}_+ \mid G_C(x, t) = 0\}$ using the complementary pivoting method until t becomes zero or a secondary ray is generated. As we will see, PathAVI generates a starting ray at an implicit extreme point of C, i.e., $\pi_C(x^0)$ is an implicit extreme point.

Computationally, finding an initial solution (x^0, t^0) amounts to computing a complementary basic solution having $z = \pi_C(x^0)$ for the following system of equations:

$$\begin{aligned} Mz + q - A^{\mathrm{T}}\lambda - w + v &= 0, \\ Az - b &= s, \end{aligned} \tag{3.2}$$

with complementarity between variables

$$\begin{aligned} K \ni s \quad &\perp \quad \lambda \in K^D, \\ 0 \le z - l \quad &\perp \quad w \ge 0, \\ 0 \le u - z \quad &\perp \quad v \ge 0. \end{aligned} \tag{3.3}$$

The complementary basic solution satisfies the sufficient conditions for a ray start as defined in Section 3.3. Then by adding $-tr$ with $r \in \mathrm{ri}(N_C(z))$ to the first equation in (3.2) and pivoting in the t variable, we generate an almost complementary feasible basis and start complementary pivoting.

Geometrically, the map $G_C(x,t)$ is defined over a PL$(n+1)$-manifold $\mathcal{M}_C$, where the definition of a manifold follows from [24, Section 4]. The manifold $\mathcal{M}_C$ consists of a pair $(\mathbb{R}^n \times \mathbb{R}_+, \{\sigma_i \times \mathbb{R}_+ \mid i \in \mathcal{I}\})$ such that each σ_i is a set formed by $\sigma_i = F_i + N_{F_i}$, where F_i is from a collection of the nonempty faces $\{F_i \mid i \in \mathcal{I}\}$ of C, and N_{F_i} is a normal cone having constant value on $\mathrm{ri}\, F_i$. The manifold $\mathcal{M}_C$ is constructed from the normal manifold $\mathcal{N}_C$ consisting of a pair $(\mathbb{R}^n, \{\sigma_i \mid i \in \mathcal{I}\})$ by doing a Cartesian product each σ_i with $\mathbb{R}_+$. Note that the collection of the sets $\{\sigma_i \mid i \in \mathcal{I}\}$ is a subdivision of $\mathbb{R}^n$. Consequently, $\{\sigma_i \times \mathbb{R}_+ \mid i \in \mathcal{I}\}$ is a subdivision of $\mathbb{R}^n \times \mathbb{R}_+$. The k-dimensional faces of the $\sigma_i \times \mathbb{R}_+$ are called the k-cells of $\mathcal{M}_C$. Similarly, the k-dimensional faces of the σ_i are called the k-cells of $\mathcal{N}_C$. The map G_C coincides with some affine transformation on each $(n+1)$-cell $\sigma_i \times \mathbb{R}_+$ as the normal map M_C does on each n-cell σ_i [67, Proposition 2.5]. Note that the starting ray $(x(t), t)$ for $t \ge t^0 > 0$ lies in the interior to some $(n+1)$-cell $\sigma_i \times \mathbb{R}_+$ of $\mathcal{M}_C$, where (x^0, t^0) is a regular point. We call a point in $\mathcal{M}_C$ a regular point if it doesn't lie in any cell $\sigma \times \tau$ of $\mathcal{M}_C$ with $\dim(G_C(\sigma \times \tau)) < n$ [24, Section 8]. Under lexicographic

pivoting, each complementary pivoting generates each piece of the 1-manifold $G^{-1}(0)$ such that it starts from a boundary of a $(n+1)$-cell of $\mathcal{M}_C$ (except for the first piece containing the starting ray) and passes through the interior of that cell until it reaches a (different) boundary. If this does not append, then we say that a secondary ray is generated. The set of $(n+1)$-cells the 1-manifold passes through never repeats [24, Lemma 15.8]. As there is a finite number of $(n+1)$-cells of $\mathcal{M}_C$, either t reaches zero (equivalently we find a solution to the AVI(C, q, M)) or a secondary ray is generated [24, Lemma 15.13].

Processability is tied to the conditions under which a secondary ray occurs. As with the LCPs, the answer to this question involves specific matrix classes that we now define.

Definition 3.1 (Definition 4.1 [14])**.** *Let K be a closed convex cone. A matrix M is said to be* copositive *with respect to K if $\langle x, Mx\rangle \geq 0$ for all $x \in K$. If furthermore it holds that for all $x \in K$ $\langle x, Mx\rangle = 0$ implies $(M + M^{\mathrm{T}})x = 0$, then M is* copositive-plus *with respect to K.*

Definition 3.2. *Let K be a closed convex cone. A matrix M is said to be* semi-monotone *with respect to K if for every $q \in \mathrm{ri}(K^D)$, the solution set of the generalized complementarity problem*

$$z \in K, \qquad Mz + q \in K^D, \qquad z^{\mathrm{T}}(Mz + q) = 0 \tag{3.4}$$

is contained in $\lin K$.

Remark 3.3. *This definition is consistent with the existing semi-monotone property in the* LCP *literature, as given in [16, Definition 3.9.1]. In this case $K = \mathbb{R}^n_+$ and* $\lin K = \{0\}$. *Then the condition* (3.4) *is equivalent to* 0 *being the solution set of* LCP(M, q) *for all $q > 0$, which by Theorem 3.9.3 in [16] is equivalent to the standard definition of M being semi-monotone.*

Definition 3.4 (Definition 4.2 [14])**.** *Let K be a closed convex cone. A matrix M is said to be an L-matrix with respect to K if both*

(a) M is semi-monotone with respect to K

(b) For any $z \neq 0$ satisfying

$$z \in K, \qquad Mz \in K^D, \qquad z^{\mathrm{T}} Mz = 0, \tag{3.5}$$

there exists $z' \neq 0$ such that z' is contained in every face of K containing z and $-M^{\mathrm{T}} z'$ is contained in every face of K^D containing Mz.

Lemma 3.5 (Lemma 4.5 [14])**.** *If a matrix M is copositive-plus with respect to a closed convex cone K, then it is an L-matrix with respect to K.*

The main existing result on the processability using a path following method is the following.

Theorem 3.6 (Theorem 4.4 [14])**.** *Suppose that C is a polyhedral convex set, and M is an L-matrix with respect to* rec C *which is invertible on the lineality space of C. Then exactly one of the following occurs:*

- *The method of [14] solves the* AVI(C, q, M).
- *The following system has no solution*

$$Mz + q \in (\mathrm{rec}\, C)^D.$$

3.3 Theoretical results

In this section, we show that an implementation of PathAVI in the original space enjoys the same properties as Theorem 3.6. We first identify sufficient conditions to allow a ray start. We define an *implicit extreme point*, which is a generalization of an extreme point when the lineality space is nontrivial, and show that there exists an implicit extreme point satisfying these sufficient conditions. A computational method for finding such an implicit extreme point is described in Section 3.4. Our conditions generalize those required for existing pivotal methods [14, 19, 53] for LCP, MCP, and AVI.

PathAVI can process L-matrices with respect to the recession cone of the feasible set of the AVI. To this end, we show that a 1-manifold (the path $G_C^{-1}(0)$) generated by PathAVI with a ray start at an implicit extreme point corresponds to a 1-manifold generated by the same pivotal method with a ray start at an extreme point in the reduced space. The reduced space is formed by projecting out the lineality space. This one-to-one correspondence is derived from the structural correspondence of the faces and the normal cones between the original space and the reduced one. Then by applying the existing processability result to the 1-manifold in the reduced space, we obtain the desired result.

3.3.1 Sufficient conditions for a ray start and processability of PathAVI

We first identify sufficient conditions to perform a ray start at a point.

Proposition 3.7. *Let an* AVI(C, q, M) *be given. If the following conditions are satisfied at a point* $\bar{z} \in C$ *with* $\bar{z} + \operatorname{lin} C$ *being a face of* C*, then we can perform a ray start at* $\bar{z}$*.*

- $M\bar{z} + q \in \text{aff}(N_C(\bar{z}))$.
- *Every point in the interior of the* $(n+1)$*-cell* $((\bar{z} + \text{lin}\, C) + N_C(\bar{z})) \times \mathbb{R}_+$ *is regular. (See Section 2 for the definition of a regular point.)*
- *There exists a complementary basis at* $\bar{z}$ *such that* $\text{aff}(N_C(\bar{z}))$ *is spanned by columns of the basic variables in* (λ, w, v).

Proof. Pick a vector $r \in \text{ri}(N_C(\bar{z}))$. Let (z, λ, w, v, s) be the complementary basic solution to (3.2) and (3.3) corresponding to the given complementary basis. Note that $z = \bar{z}$, thus s is feasible. Therefore, only basic variables in (λ, w, v) might be infeasible. The first and third conditions say that we have $Mz + q - A^{\mathrm{T}}\lambda - w + v = 0$. By the third condition, for each $t \geq 0$ we have a unique $(\lambda(t), w(t), v(t))$ satisfying $Mz + q - A^{\mathrm{T}}\lambda(t) - w(t) + v(t) - tr = 0$. As $r \in \text{ri}(N_C(\bar{z}))$, there exists $t^0 \geq 0$ such that for all $t \geq t^0$ we have $Mz + q - A^{\mathrm{T}}\lambda(t) - w(t) + v(t) - tr = 0$ and $(\lambda(t), w(t), v(t))$ are feasible variables. Then for all $t \geq t^0$ $(x(t), t)$ with $x(t) := \bar{z} - A^{\mathrm{T}}\lambda(t) - w(t) + v(t)$ lies in the cell $((\bar{z} + \text{lin}\, C) + N_C(\bar{z})) \times \mathbb{R}_+$ with $\pi_C(x(t)) = \bar{z}$ and $G_C(x(t), t) = 0$. By the second condition, the ray $(x(t), t)$ is generated at a regular point. By pivoting the t variable into the complementary basis, we see that we can perform a ray start at $\bar{z}$. □

Note that the sufficient conditions are satisfied at an extreme point. If z is an extreme point, then $\text{aff}(N_C(z)) \equiv \mathbb{R}^n$ thus the first condition is trivially satisfied. Each extreme point has a corresponding basic feasible solution (BFS) to $Ax - b = s$ [60, Section 3.4], and with that BFS we can construct a complementary basis satisfying the third condition as shown in Proposition 3.23 later in this chapter. The second condition is also satisfied as proved

in Proposition 3.11. As the existing pivotal methods [14, 19, 53] for LCP, MCP, and AVI perform a ray start at an extreme point, we see that the sufficient conditions generalize the existing result.

We now define an implicit extreme point, which is a generalization of an extreme point when the lineality space is nontrivial.

Definition 3.8. *Let C be a convex set in $\mathbb{R}^n$. A point $z \in C$ is called an implicit extreme point of C if $z = \lambda z^1 + (1-\lambda)z^2$ for any $z^1, z^2 \in C$ and $\lambda \in (0,1)$ implies that $z - z^1 \in \operatorname{lin} C$ and $z - z^2 \in \operatorname{lin} C$.*

Note that if the lineality space of C is trivial, that is, $\operatorname{lin} C = \{0\}$, then the definition of an implicit extreme point coincides with definition of an extreme point.

In the following four propositions, we provide some properties of implicit extreme points, which are generalization of the ones enjoyed by extreme points. They are used as a tool for showing the existence of an implicit extreme point satisfying the sufficient conditions and for structural analysis later in this section. We start with faces consisting of only implicit extreme points. This generalizes 0-dimensional faces that are equivalent to extreme points. As the proof is elementary, we omit it.

Proposition 3.9. *Let C be a nonempty convex set in $\mathbb{R}^n$ and $\ell = \dim(\operatorname{lin} C)$. Then every point in an ℓ-dimensional face of C is an implicit extreme point of C. Also, for each implicit extreme point z of C we have $F = z + \operatorname{lin} C$ is an ℓ-dimensional face of C.*

We prove next that the affine hull of the normal cone to C at an implicit extreme point is the orthogonal complement of the lineality space of C. This generalizes the fact that the normal cone to C at an extreme point is full-dimensional.

Proposition 3.10. *A point $z \in C$ is an implicit extreme point of a nonempty polyhedral convex set C in $\mathbb{R}^n$ if and only if* $\operatorname{aff}(N_C(z)) = (\operatorname{lin} C)^\perp$.

Proof. (only-if) Suppose that z is an implicit extreme point of C. Using Proposition 3.9, $F = z + \operatorname{lin} C$ is a face of C. We then have $\operatorname{par} F = \operatorname{lin} C$. By [67, Proposition 2.1], $\operatorname{par} F = (\operatorname{aff} N_F)^\perp$, where N_F represents the normal cone having the same value for all $\hat{z} \in \operatorname{ri} F$, i.e., $N_C(\hat{z}^1) = N_F = N_C(\hat{z}^2)$ for all $\hat{z}^1, \hat{z}^2 \in \operatorname{ri} F$. As $z \in \operatorname{ri} F$, it follows that $\operatorname{aff}(N_C(z)) = (\operatorname{lin} C)^\perp$.

(if) Suppose that $z \in C$ and $\operatorname{aff}(N_C(z)) = (\operatorname{lin} C)^\perp$. Pick a face F of C such that $z \in \operatorname{ri} F$. Such a face exists by [71, Theorem 18.2]. Then $N_C(z) = N_F$, where N_F is the normal cone having constant value on $\operatorname{ri} F$. As $\operatorname{par} F = (\operatorname{aff} N_F)^\perp$, we then have $\operatorname{par} F = \operatorname{lin} C$ and $F = z + \operatorname{lin} C$. By Proposition 3.9, z is an implicit extreme point of C. □

Next we show that the second condition in Proposition 3.7 is satisfied at an implicit extreme point. Note that in the proposition below we show $\dim(M_C(\sigma)) = n$, which implies that $\dim(G_C(\sigma \times \mathbb{R}_+)) = n$.

Proposition 3.11. *Let z be an implicit extreme point of a nonempty polyhedral convex set C in $\mathbb{R}^n$ and σ be the cell $((z + \operatorname{lin} C) + N_C(z))$ in the normal manifold of C. Then for an* AVI(C, q, M) *with M invertible on the lineality space of C, we have* $\dim(M_C(\sigma)) = n$.

Proof. By [67, Proposition 2.5], M_C coincides with some affine transformation A_σ on σ. In the basis $Z = (Q \;\; \bar{Q})$, we can represent the matrix $A_\sigma(\cdot) - A_\sigma(z)$ as follows:

$$\begin{bmatrix} Q^{\mathrm{T}} M Q & 0 \\ \bar{Q}^{\mathrm{T}} M Q & I \end{bmatrix}.$$

As $Q^{\mathrm{T}}MQ$ is invertible, the matrix $A_\sigma(\cdot) - A_\sigma(z)$ is invertible. As σ is n-dimensional, the result follows. □

Finally, let us consider a ℓ-dimensional face F with $\ell = \dim(\operatorname{lin} C)$ (hence consisting of only implicit extreme points by Proposition 3.9). Then there exists an implicit extreme point $z \in F$ such that $Mz + q \in \operatorname{aff}(N_C(z))$. This generalizes the fact that at each extreme point $\bar{z}$ we have $M\bar{z} + q \in \operatorname{aff}(N_C(\bar{z})) = \mathbb{R}^n$.

Proposition 3.12. *Let an* AVI(C, q, M) *problem be given and $z \in C$ be an implicit extreme point of C. Assume that M is invertible on the lineality space of C. Then there exists $\hat{z} \in z + \operatorname{lin} C$ such that $M\hat{z} + q \in \operatorname{aff}(N_C(\hat{z}))$.*

Proof. For any implicit extreme point $\hat{z}$ of C, $M\hat{z} + q \in \operatorname{aff}(N_C(\hat{z}))$ if and only if $\pi_{\operatorname{lin} C}(M\hat{z} + q) = 0$ by Proposition 3.10. By the assumption, M_{QQ} is invertible. Set

$$\hat{z} = z + Qy \quad \text{where} \quad y = -M_{QQ}^{-1}(Q^{\mathrm{T}}q + M_{Q\bar{Q}}\bar{Q}^{\mathrm{T}}z) - Q^{\mathrm{T}}z.$$

Then $\hat{z} \in z + \operatorname{lin} C$ thus $\hat{z}$ is an implicit extreme point of C by Proposition 3.9, and

$$\begin{aligned} Q^{\mathrm{T}}(M\hat{z} + q) &= Q^{\mathrm{T}}\left(M\begin{bmatrix} Q & \bar{Q} \end{bmatrix}\begin{bmatrix} Q^{\mathrm{T}} \\ \bar{Q}^{\mathrm{T}} \end{bmatrix}\hat{z} + q\right), \\ &= M_{QQ}(Q^{\mathrm{T}}\hat{z}) + M_{Q\bar{Q}}(\bar{Q}^{\mathrm{T}}\hat{z}) + Q^{\mathrm{T}}q, \\ &= M_{QQ}(Q^{\mathrm{T}}z + y) + M_{Q\bar{Q}}(\bar{Q}^{\mathrm{T}}z) + Q^{\mathrm{T}}q, \\ &= 0. \end{aligned}$$

It follows that $\pi_{\operatorname{lin} C}(M\hat{z} + q) = 0$. □

By Propositions 3.11 and 3.12, there exists an implicit extreme point satisfying the first two sufficient conditions for a ray start. We postpone checking the third condition to Section 3.4 as it requires a constructive proof. For the rest of this section, we assume that we have an implicit extreme point satisfying the sufficient conditions.

We now turn our attention to the processability of PathAVI. Assume that we perform a ray start at an implicit extreme point and generate a 1-manifold in the original space $\mathbb{R}^n$. Our basic idea of deriving processability is that this 1-manifold corresponds to a 1-manifold generated by the same pivotal method with a ray start at an extreme point defined in the reduced space having possibly smaller dimension. We can then apply the existing processability result [14, Theorem 4.4]. To establish the correspondence, we prove that there is a one-to-one correspondence between the faces, the normal cones, and the full-dimensional cells of the original space and reduced space.

Proposition 3.13. *Let C be a nonempty polyhedral convex set in $\mathbb{R}^n$ and $\tilde{C}$ be the set $\tilde{C} = \bar{Q}^{\mathrm{T}}C = \{\tilde{z} \mid \tilde{z} = \bar{Q}^{\mathrm{T}}z$ for some $z \in C\}$ defined in $\mathbb{R}^{n-\ell}$ where $\ell = \dim(\operatorname{lin} C)$. Then the followings hold.*

(a) z is an implicit extreme point of C if and only if $\tilde{z} = \bar{Q}^{\mathrm{T}}z$ is an extreme point of $\tilde{C}$.

(b) F is a face of C if and only if $\tilde{F} = \bar{Q}^{\mathrm{T}}F$ is a face of $\tilde{C}$.

(c) $v \in N_C(z)$ if and only if $v = \bar{Q}\tilde{v}$ for some $\tilde{v} \in N_{\tilde{C}}(\tilde{z})$ where $\tilde{z} = \bar{Q}^{\mathrm{T}}z$.

(d) σ is an n-cell of the normal manifold $\mathcal{N}_C$ of C if and only if $\tilde{\sigma} = \bar{Q}^{\mathrm{T}}\sigma$ is an $(n-\ell)$-cell of the normal manifold $\mathcal{N}_{\tilde{C}}$.

Proof. We prove in sequence. (a) (only-if) Let z be an implicit extreme point of C. Set $\tilde{z} = \bar{Q}^{\mathrm{T}} z$. We prove by contradiction. Suppose that $\exists \tilde{z}^1, \tilde{z}^2 \in \tilde{C}$ and $\lambda \in (0,1)$ such that $\tilde{z} = \lambda\tilde{z}^1 + (1-\lambda)\tilde{z}^2$ with $\tilde{z} \neq \tilde{z}^i$ for $i = 1,2$. By definition of $\tilde{C}$, we have $z^1, z^2 \in C$ such that $\tilde{z}^i = \bar{Q}^{\mathrm{T}} z^i$ for $i = 1,2$. As $C = \operatorname{lin} C \oplus ((\operatorname{lin} C)^{\perp} \cap C)$ [71, page 65] and $\bar{Q}^{\mathrm{T}} z = \bar{Q}^{\mathrm{T}}(\lambda z^1 + (1-\lambda)z^2)$, there exists $a \in \operatorname{lin} C$ such that $z = \lambda(a + z^1) + (1-\lambda)(a + z^2)$. As $\bar{Q}^{\mathrm{T}}(z - (a + z^i)) = \tilde{z} - \tilde{z}^i \neq 0$, we have $z - (a + z^i) \notin \operatorname{lin} C$ for $i = 1,2$, which contradicts our assumption that z is an implicit extreme point of C.

(if) Using similar proof technique, we can show that for an extreme point $\tilde{z} \in \tilde{C}$ z is an implicit extreme point of C when $\tilde{z} = \bar{Q}^{\mathrm{T}} z$.

(b) (only-if) Let F be a face of C. Set $\tilde{F} = \bar{Q}^{\mathrm{T}} F$. Clearly, $\tilde{F}$ is a convex subset of $\tilde{C}$. Let $\tilde{z}^1, \tilde{z}^2 \in \tilde{C}$ and $\lambda \in (0,1)$ satisfying $\lambda\tilde{z}^1 + (1-\lambda)\tilde{z}^2 \in \tilde{F}$. From $C = \operatorname{lin} C \oplus ((\operatorname{lin} C)^{\perp} \cap C)$, we have $\bar{Q}\tilde{z}^i \in C$ for $i = 1,2$. Then $\bar{Q}(\lambda\tilde{z}^1 + (1-\lambda)\tilde{z}^2) \in F$ so that $\bar{Q}\tilde{z}^1 \in F$ and $\bar{Q}\tilde{z}^2 \in F$. This shows that $\tilde{z}^i \in \tilde{F}$ for $i = 1,2$.

(if) Let $\tilde{F} = \bar{Q}^{\mathrm{T}} F$ be a face of $\tilde{C}$. By the definition of $\tilde{F}$, F is a convex subset of C. Let $z^1, z^2 \in C$ and $\lambda \in (0,1)$ such that $\lambda z^1 + (1-\lambda)z^2 \in F$. We have $\bar{Q}^{\mathrm{T}} z^i \in \tilde{C}$ for $i = 1,2$ and $\bar{Q}^{\mathrm{T}}(\lambda z^1 + (1-\lambda)z^2) \in \tilde{F}$. Thus, $\bar{Q}^{\mathrm{T}} z^i \in \tilde{F}$, hence $z^i \in F + \operatorname{lin} C$ for $i = 1,2$. Therefore, $z^i \in F$ for $i = 1,2$.

(c) For a vector $v \in \mathbb{R}^n$, we represent components of v in $\operatorname{lin} C$ and $(\operatorname{lin} C)^{\perp}$ in the basis $\begin{bmatrix} Q & \bar{Q} \end{bmatrix}$ by v_Q and $v_{\bar{Q}}$, respectively, so that $v = Qv_Q + \bar{Q}v_{\bar{Q}}$. If either $z \notin C$ or $\tilde{z} \notin \tilde{C}$, then we have nothing to prove. Therefore, we assume that $z \in C$ and $\tilde{z} \in \tilde{C}$ in the proof. (only-if) Let $v \in N_C(z)$. By the definition of the normal cone, for each $a \in \operatorname{lin} C$ we have $\langle v, (z + a) - z \rangle \leq 0$ and $\langle v, (z - a) - z \rangle \leq 0$. Thus, $\langle v, a \rangle = 0$ for all $a \in \operatorname{lin} C$. Whence

$N_C(z) \subset (\operatorname{lin} C)^\perp$ so that $v_Q = 0$ and $v = \bar{Q}v_{\bar{Q}}$. We then have

$$\begin{aligned} 0 &\geq \langle v, y - z\rangle, \quad \forall y \in C \\ &= \langle \bar{Q}v_{\bar{Q}}, Qy_Q + \bar{Q}y_{\bar{Q}} - (Qz_Q + \bar{Q}z_{\bar{Q}})\rangle \\ &= \langle \bar{Q}v_{\bar{Q}}, \bar{Q}(y_{\bar{Q}} - z_{\bar{Q}})\rangle \\ &= \langle v_{\bar{Q}}, y_{\bar{Q}} - z_{\bar{Q}}\rangle \end{aligned}$$

By setting $\tilde{v} = v_{\bar{Q}}$, $v = \bar{Q}\tilde{v}$ and $\tilde{v} \in N_{\tilde{C}}(\tilde{z})$.

(if) Let $\tilde{v} \in N_{\tilde{C}}(\tilde{z})$ and set $v = \bar{Q}\tilde{v}$. We have $\tilde{z} = \bar{Q}^{\mathrm{T}}z$ if and only if $z \in \operatorname{lin} C + \bar{Q}\tilde{z}$. Let $z \in \operatorname{lin} C + \bar{Q}\tilde{z}$. Then

$$\langle v, y - z\rangle = \langle \tilde{v}, y_{\bar{Q}} - z_{\bar{Q}}\rangle \leq 0, \quad y \in C$$

and the result follows.

(d) The conclusion follows from (b), (c), and the definition of the full-dimensional cells of the normal manifold. □

A similar result holds for the 1-manifold $G_C^{-1}(0)$.

Proposition 3.14. *Let an* AVI(C, q, M) *problem be given. Suppose that the matrix M is invertible on the lineality space of C, and $G_C(x^*, t^*) = 0$ with $r \in N_C(\pi_C(x^0))$ for some $x^0 \in \mathbb{R}^n$. Then the* PL *function $\tilde{G}_{\tilde{C}}(\tilde{x}, t) := \tilde{M}\pi_{\tilde{C}}(\tilde{x}) + \tilde{q} + \tilde{x} - \pi_{\tilde{C}}(\tilde{x}) - t\tilde{r}$ has value zero at*

$(\tilde{x}^*, t^*)$, *where*

$$\begin{aligned}
\tilde{x}^* &= \bar{Q}^{\mathrm{T}} x^* \\
Z &= \begin{bmatrix} Q & \bar{Q} \end{bmatrix}, \\
\tilde{M} &= (Z^{\mathrm{T}} M Z / M_{QQ}) = M_{\bar{Q}\bar{Q}} - M_{\bar{Q}Q} M_{QQ}^{-1} M_{Q\bar{Q}}, \\
\tilde{C} &= \bar{Q}^{\mathrm{T}} C,\ \tilde{x}^0 = \bar{Q}^{\mathrm{T}} x^0,\ \tilde{q} = (\bar{Q}^{\mathrm{T}} - M_{\bar{Q}Q} M_{QQ}^{-1} Q^{\mathrm{T}}) q, \\
\tilde{r} &= \bar{Q}^{\mathrm{T}} r \in N_{\tilde{C}}(\pi_{\tilde{C}}(\tilde{x}^0)).
\end{aligned}$$

Conversely, if $\tilde{G}_{\tilde{C}}(\tilde{x}^*, t^*) = 0$ *then* $G_C(x^*, t^*) = 0$ *with* $x^* = \bar{Q}\tilde{x}^* + Qy^*$ *and*

$$y^* = -M_{QQ}^{-1} \left(M_{Q\bar{Q}} \pi_{\tilde{C}}(\tilde{x}^*) + Q^{\mathrm{T}} q \right).$$

Proof. Let (x^*, t^*) satisfying $G_C(x^*, t^*) = 0$ with $r \in N_C(\pi_C(x^0))$ for some x^0 be given. Then

$$\begin{aligned}
& M\pi_C(x^*) + q + x^* - \pi_C(x^*) - t^* r = 0, \\
(\Rightarrow) & \begin{bmatrix} Q^{\mathrm{T}} \\ \bar{Q}^{\mathrm{T}} \end{bmatrix} M \begin{bmatrix} Q & \bar{Q} \end{bmatrix} \begin{bmatrix} Q^{\mathrm{T}} \\ \bar{Q}^{\mathrm{T}} \end{bmatrix} \pi_C(x^*) + \begin{bmatrix} Q^{\mathrm{T}} \\ \bar{Q}^{\mathrm{T}} \end{bmatrix} (q + x^* - \pi_C(x^*) - t^* r) = 0, \\
(\Rightarrow) & \begin{bmatrix} M_{QQ} & M_{Q\bar{Q}} \\ M_{\bar{Q}Q} & M_{\bar{Q}\bar{Q}} \end{bmatrix} \begin{bmatrix} Q^{\mathrm{T}} \pi_C(x^*) \\ \bar{Q}^{\mathrm{T}} \pi_C(x^*) \end{bmatrix} + \begin{bmatrix} Q^{\mathrm{T}} q \\ \bar{Q}^{\mathrm{T}} (q + x^* - \pi_C(x^*) - t^* r) \end{bmatrix} = 0, \\
(\Rightarrow) & \tilde{M} \bar{Q}^{\mathrm{T}} \pi_C(x^*) + \tilde{q} + \bar{Q}^{\mathrm{T}} (x^* - \pi_C(x^*)) - t^* \tilde{r} = 0, \\
& \qquad \text{using } Q^{\mathrm{T}} \pi_C(x^*) = -M_{QQ}^{-1} (M_{Q\bar{Q}} \bar{Q}^{\mathrm{T}} \pi_C(x^*) + Q^{\mathrm{T}} q), \\
(\Rightarrow) & \tilde{M} \pi_{\tilde{C}}(\tilde{x}^*) + \tilde{q} + \tilde{x}^* - \pi_{\tilde{C}}(\tilde{x}^*) - t^* \tilde{r} = 0.
\end{aligned}$$

The second $(\Rightarrow)$ holds because $N_C(z) \subset (\operatorname{lin} C)^{\perp}, \forall z \in C$ as shown in the proof of Proposi-

tion 3.13(c). The last ($\Rightarrow$) holds because $\bar{Q}^{\mathrm{T}}\pi_C(x) = \pi_{\bar{Q}^{\mathrm{T}}C}(\bar{Q}^{\mathrm{T}}x)$ by [13, Lemma 2.1]. Also, $\tilde{r} \in N_{\tilde{C}}(\pi_{\tilde{C}}(\tilde{x}^0))$ by Proposition 3.13(d).

Conversely, let $\tilde{G}_{\tilde{C}}(\tilde{x}^*, t^*) = 0$. Set $x^* = \bar{Q}\tilde{x}^* + Qy^*$ with y^* as specified in the proposition. Then

$$\begin{aligned}\pi_C(x^*) &= \pi_{C\cap(\operatorname{lin} C)^\perp}(x^*) + \pi_{\operatorname{lin} C}(x^*)\\ &= \bar{Q}\pi_{\tilde{C}}(\tilde{x}^*) + Qy^*.\end{aligned}$$

Therefore, $Q^{\mathrm{T}}\pi_C(x^*) = y^*$. By the definition of y^*, all the converse directions also hold. □

Note that the AVI($\tilde{C}, \tilde{q}, \tilde{M}$) with $\tilde{C}, \tilde{q}$, and $\tilde{M}$ as in Proposition 3.14 is the same problem obtained by applying the stage 1 reduction [14, page 49] to the AVI(C, q, M). Also, $\tilde{G}_{\tilde{C}}$ is the PL function defined on the $(n - \dim(\operatorname{lin} C) + 1)$-manifold $\mathcal{M}_{\tilde{C}}$ of $\tilde{C}$ to find a zero of the normal map associated with the AVI($\tilde{C}, \tilde{q}, \tilde{M}$).

An implication of Proposition 3.14 is that if $G_C(x + \theta\Delta x, t + \theta\Delta t) = 0$ with $(x + \theta\Delta x, t + \theta\Delta t) \in \sigma \times \mathbb{R}_+$ for all $\theta \in [0, \nu]$ with $\nu > 0$ (possibly $\nu = \infty$) and $\sigma \times \mathbb{R}_+$ is an $(n+1)$-cell of $\mathcal{M}_C$, then we have $\tilde{G}_{\tilde{C}}(\tilde{x} + \theta\Delta\tilde{x}, t + \theta\Delta t) = 0$ with $(\tilde{x} + \theta\Delta\tilde{x}, t + \theta\Delta t) \in \tilde{\sigma} \times \mathbb{R}_+$ for all $\theta \in [0, \nu]$, where $\tilde{\sigma} = \bar{Q}^{\mathrm{T}}\sigma$ and $\Delta\tilde{x} = \bar{Q}^{\mathrm{T}}\Delta x$. The converse also holds by setting $\Delta x = \bar{Q}\Delta\tilde{x} + Q\Delta y$ with $\Delta y = -M_{QQ}^{-1}M_{Q\bar{Q}}H_{\tilde{\sigma}}\Delta\tilde{x}$, where $H_{\tilde{\sigma}}$ is an orthogonal projector onto par $\tilde{F}$ with $\tilde{\sigma} = \tilde{F} + N_{\tilde{F}}$ [67, see the proof of Proposition 2.5].

Therefore, the projection of each piece of $G_C^{-1}(0)$ onto $\mathcal{M}_{\tilde{C}}$ corresponds to each piece of $\tilde{G}_{\tilde{C}}^{-1}(0)$ and vice versa. As a consequence, if $G_C^{-1}(0)$ contains a ray, i.e., there exists $(\Delta x, \Delta t) \neq 0$ with $\nu = \infty$ on some $(n+1)$-cell $\sigma \times \mathbb{R}_+$ of $\mathcal{M}_C$, and the corresponding value $(\Delta\tilde{x}, \Delta t)$ is not zero, then the corresponding piece of $\tilde{G}_{\tilde{C}}^{-1}(0)$ is also a ray. The following proposition shows that whenever there is a ray in $G_C^{-1}(0)$ with $\Delta x \neq 0$, then we have $\Delta\tilde{x} \neq 0$ so that the corresponding piece of $\tilde{G}_{\tilde{C}}^{-1}(0)$ is also a ray. Note that the converse automatically

holds as $\Delta x \neq 0$ for each $\Delta\tilde{x} \neq 0$.

Proposition 3.15. *For an* AVI(C, q, M)*, suppose that* PathAVI *generates* $G_C^{-1}(0)$ *with a ray start at an implicit extreme point. For each ray in* $G_C^{-1}(0)$ *in the direction of* $(\Delta x, \Delta t) \neq 0$*, if* $\Delta x \neq 0$ *then* $\Delta\tilde{x} := \bar{Q}^{\mathrm{T}}\Delta x$ *is a ray in* $\tilde{G}_{\tilde{C}}^{-1}(0)$ *that is nonzero under the assumption that either* 0 *is a regular value or we perform lexicographic pivoting.*

Proof. Let z be an implicit extreme point at which PathAVI performs a ray start. By construction, $\Delta\tilde{x} = 0$ if and only if $\Delta x \in \operatorname{lin} C$. For the starting ray we have $\Delta x \in N_C(z)$ so that $\Delta x \notin \operatorname{lin} C$ by Proposition 3.10. Thus, $\Delta\tilde{x} \neq 0$.

We now assume that there is a ray in $G_C^{-1}(0)$ other than the starting ray. Suppose that $\Delta x \in \operatorname{lin} C$. We proceed by contradiction. Let us assume that the ray is generated at the $(k+1)$th iteration of complementary pivoting, and that it starts from (x^{k+1}, t^{k+1}). We know that $(x^{k+1}, t^{k+1}) \in (\sigma^{k+1} \times \mathbb{R}_+) \cap (\sigma^k \times \mathbb{R}_+)$, where $\sigma^k \times \mathbb{R}_+$ is the $(n+1)$-cell of $\mathcal{M}_C$ PathAVI passes through at the kth complementary pivoting iteration. As $\operatorname{lin} C \subset \operatorname{lin} \sigma$ for each $(n+1)$-cell $\sigma \times \mathbb{R}_+$ of $\mathcal{M}_C$, $x^{k+1} + \theta\Delta x \in \sigma^k$ for all $\theta \geq 0$. This contradicts the fact that $G_C^{-1}(0)$ is a 1-manifold neat in $\mathcal{M}_C$ [24, Theorem 9.1 or Lemma 15.5], that is, $G^{-1}(0) \cap (\sigma^k \times \mathbb{R}_+)$ must be expressed as an intersection of $\sigma^k \times \mathbb{R}_+$ with a line. Therefore, $\Delta x \notin \operatorname{lin} C$ and the result follows. □

From Lemma 3.26 (in the Appendix), if M is semi-monotone with respect to $\operatorname{rec} C$ and invertible on $\operatorname{lin} C$, we have $\Delta t = 0$ whenever PathAVI generates a ray in the direction of $(\Delta x, \Delta t)$. Matrix classes having the property $\Delta t = 0$ include the L-matrix class and the new matrix classes defined in Section 3.3.2. Therefore, whenever PathAVI generates a ray

in $G_C^{-1}(0)$ for those classes of matrices the corresponding piece in $\tilde{G}_{\tilde{C}}^{-1}(0)$ is also a ray by Proposition 3.15.

Equipped with Propositions 3.13–3.15, we now show that PathAVI can process L-matrices. In contrast to [14], we do not resort to a reduction to show the result.

Theorem 3.16. *Suppose that C is a polyhedral convex set, and M is an L-matrix with respect to* rec C *which is invertible on the lineality space of C. Then exactly one of the following occurs:*

- PathAVI *solves the* AVI(C, q, M).
- *The following system has no solution*

$$Mz + q \in (\operatorname{rec} C)^D.$$

Proof. By Propositions 3.13–3.14, for a 1-manifold $G_C^{-1}(0)$ generated by PathAVI there corresponds to a 1-manifold $\tilde{G}_{\tilde{C}}^{-1}(0)$ in the reduced space generated by the same pivotal method with a ray start at an extreme point of $\tilde{C}$ with $\tilde{M}$ an L-matrix with respect to rec $\tilde{C}$. If there is a secondary ray in $G_C^{-1}(0)$, then so is in $\tilde{G}_{\tilde{C}}^{-1}(0)$ by Proposition 3.15. Therefore, there exists directions $(\Delta\tilde{x}, \Delta\tilde{z}, \Delta\tilde{\lambda}, \Delta\tilde{s}, \Delta t)$ in the reduced space satisfying

$$\begin{aligned} \Delta\tilde{x} - \Delta\tilde{z} &= -\tilde{M}\Delta\tilde{z} + \tilde{r}\Delta t, \\ A_{\mathcal{A}\bullet}\Delta\tilde{z} &= 0, \\ A_{\bar{\mathcal{A}}\bullet}\Delta\tilde{z} - \Delta\tilde{s}_{\bar{\mathcal{A}}} &= 0, \\ \Delta\tilde{x} - \Delta\tilde{z} &= -A_{\mathcal{A}\bullet}^{\mathrm{T}}\Delta\tilde{\lambda}_{\mathcal{A}} \end{aligned} \tag{3.6}$$

where we have included bound constraints in the matrix A for clarity, and $\mathcal{A}$ and $\bar{\mathcal{A}}$ denote the active and inactive sets, respectively. We then apply Theorem 3.6 to (3.6) to get the desired result. □

3.3.2 Additional processability results

Let us now extend the classes of AVIs that PathAVI is able to process. The results in Lemmas 3.17–3.19 consider the structure of the whole AVI, not only M and C. As stated in the paragraph following Proposition 3.14, a 1-manifold generated in $\mathcal{M}_C$ corresponds to another one in $\mathcal{M}_{\tilde{C}}$. Hence, in the following we denote by AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ the AVI corresponding to AVI(C, q, M) with the lineality space projected out as explained in Proposition 3.14. If M is invertible on $\operatorname{lin} C$, the results can then be applied to the original AVI by noticing that the projections of the directions of the rays on $G_C^{-1}(0)$ are solution to the system of equations (3.6) in the reduced space.

In Section 3.6.1, we present a friction contact problem where Theorem 3.16 cannot be applied but the following lemma is appropriate.

Lemma 3.17. *Consider an* AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ *with* $\operatorname{lin} \tilde{C} = \{0\}$. *Suppose that* $\tilde{M}$ *is semi-monotone with respect to* $\operatorname{rec} \tilde{C}$ *and that for any solution* $z \neq 0$ *of the problem*

$$z \in \operatorname{rec} \tilde{C}, \quad \tilde{M}z \in (\operatorname{rec} \tilde{C})^D, \quad z^{\mathrm{T}} \tilde{M} z = 0, \tag{3.7}$$

it holds that

$$z^{\mathrm{T}}(\tilde{M}z' + \tilde{q}) \geq 0, \quad \forall z' \in \tilde{C}. \tag{3.8}$$

Then PathAVI *solves the* AVI$(\tilde{C}, \tilde{q}, \tilde{M})$.

Proof. The pivotal method used in PathAVI fails if an unbounded ray is generated at some iterate (x^k, t^k), $k > 0$. Now suppose that the method generates an unbounded ray. From Lemma 3.26, we know that $\Delta t = 0$, and $\Delta z \neq 0$ is a solution to (3.7). This means that for any point x^{k+1} on the ray, we have $\tilde{G}_{\tilde{C}}(x^{k+1}, t^{k+1}) = 0$, implying that

$$\langle \Delta z, \tilde{G}_{\tilde{C}}(x^{k+1}, t^k) \rangle = \langle \Delta z, \tilde{M} z^{k+1} + \tilde{q} \rangle + \langle \Delta z, x^{k+1} - z^{k+1} \rangle + \langle \Delta z, -t^k r \rangle = 0. \tag{3.9}$$

The first term is non-negative by our assumption, as well as the second one by the normal cone definition. The third one is strictly positive since $-t^k r \in \operatorname{int}(\operatorname{rec} \tilde{C})^D$. Hence, we reached a contradiction. □

An additional property on $\tilde{M}$ allows easier checking of condition (3.8) of Lemma 3.17.

Corollary 3.18. *If for any solution z of* (3.7) *we have $\langle z', \tilde{M}^{\mathrm{T}} z \rangle \geq 0$, for all $z' \in \tilde{C}$, then the condition* (3.8) *reduces to $z^{\mathrm{T}} \tilde{q} \geq 0$ whenever z is a solution to* (3.7)*.*

We introduce an additional problem class PathAVI can process.

Lemma 3.19. *Consider an* AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ *with* $\operatorname{lin} \tilde{C} = \{0\}$*. Suppose that $\tilde{C}$ is a proper cone, $\tilde{M}$ is copositive with respect to $\tilde{C}$ and that the following implication holds:*

$$z \in \operatorname{rec} \tilde{C}, \quad \tilde{M} z \in (\operatorname{rec} \tilde{C})^D, \quad z^{\mathrm{T}} \tilde{M} z = 0 \quad \Rightarrow \quad z^{\mathrm{T}} \tilde{q} \geq 0. \tag{3.10}$$

Then the AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ *has a solution and* PathAVI *finds it.*

Proof. Recall from [14, Lemma 4.3], that a copositive matrix is also semi-monotone. This implies that $\Delta t = 0$ and that $\Delta z \neq 0$ satisfies the left-hand side of (3.10). Now let us suppose

that at the current iterate x_k, there exists an unbounded ray. Letting $z_{k+1} = z_k + \theta\Delta z$ and computing the inner product $\langle z_{k+1}, \tilde{G}_{\tilde{C}}(x_{k+1}, t_k)\rangle$ yields

$$0 = \langle z_{k+1}, \tilde{G}_{\tilde{C}}(x_{k+1}, t_k)\rangle = \langle z_{k+1}, \tilde{M}z_{k+1}\rangle + \langle z_{k+1}, \tilde{q}\rangle + \langle z_{k+1}, x_{k+1} - z_{k+1}\rangle + \langle z_{k+1}, -t_k r\rangle. \quad (3.11)$$

Note that since $\tilde{C}$ is pointed, $\langle z_{k+1}, x_{k+1} - z_{k+1}\rangle \geq 0$ by the definition of the normal cone. The first term is quadratic in θ while the second and third are linear in θ. Therefore, if $\langle \Delta z, \tilde{M}\Delta z\rangle > 0$, then $\langle z_{k+1}, \tilde{G}_{\tilde{C}}(x_{k+1}, t_k)\rangle > 0$ for θ large enough and we reach a contradiction. We are left with the case $\langle \Delta z, \tilde{M}\Delta z\rangle = 0$:

$$0 = \langle z_k, \tilde{q}\rangle - \langle z_k, t_k r\rangle + \langle z_{k+1}, x_{k+1} - z_{k+1}\rangle + \langle z_{k+1}, \tilde{M}z_{k+1}\rangle + \theta(\langle \Delta z, \tilde{q}\rangle + \langle \Delta z, -t_k r\rangle). \quad (3.12)$$

The sum multiplied by θ is positive since $-t_k r \in \operatorname{int}(\operatorname{rec}\tilde{C})^D$. Now the first two terms are constant and the third and fourth ones are nonnegative. Whence for θ large enough, $\langle z_{k+1}, \tilde{G}_{\tilde{C}}(x_{k+1}, t_k)\rangle$ is positive, which concludes the proof. □

Remark 3.20. *Lemma 3.19 was already known for the LCP case (that is $\tilde{C} = \mathbb{R}^n_+$): the existence of a solution is given in [16, Theorem 3.8.6]. Here we are able to provide a constructive proof for an* AVI$(\tilde{C}, \tilde{q}, \tilde{M})$ *over a proper cone.*

Let us present an AVI(C, q, M) that satisfies the conditions of Lemma 3.19 where M is not an L-matrix. Suppose that $C \subseteq \mathbb{R}^{n+1}_+$ is a polyhedral solid cone, $M = \begin{pmatrix} I_n & 0 \\ \mathbf{1}_n^{\mathrm{T}} & 0 \end{pmatrix}$, with $\mathbf{1}_n$ the vector of ones of size n and $q = (0_n, 1)^{\mathrm{T}}$. The solution set of the system $x \in C$, $Mx = 0$ and $x^{\mathrm{T}}Mx = 0$ is $\{(0_n, \alpha)^{\mathrm{T}}, \alpha \geq 0\}$. Note that if $x = (0_n, \alpha)^{\mathrm{T}}$, $\alpha > 0$, then $Mx = 0$ and $x^{\mathrm{T}}Mx = 0$. However, for any nonzero vector $x' = (x_1'^{\mathrm{T}}, \alpha')^{\mathrm{T}}$ in C,

$-M^{\mathrm{T}}x' = (-I_n x_1'^{\mathrm{T}} - \alpha'\mathbf{1}^{\mathrm{T}}, 0)^{\mathrm{T}} \notin C^D$. Therefore, condition (b) of the L-matrix fails to hold. On the other hand, we can readily check that M is copositive with respect to C and that for any $x = (0_n, \alpha)^{\mathrm{T}}$, $\alpha \geq 0$, $x^{\mathrm{T}}q = \alpha \geq 0$, so that Lemma 3.19 can be used.

3.4 Computing an implicit extreme point for a ray start

In this section, we describe how to compute an implicit extreme point satisfying the sufficient conditions for a ray start and the complementary basis associated with it so that we can start complementary pivoting at that implicit extreme point. The overall procedure is as follows: i) we first compute an initial basic feasible solution using a linear programming (LP) solver, i.e., CPLEX or GUROBI; ii) as the initial solution might not be an implicit extreme point, we may perform additional pivoting to move to an implicit extreme point; iii) using the basis information associated with the implicit extreme point, we then construct a complementary system of equations such that a unique solution to that system of equations is an implicit extreme point satisfying the sufficient conditions for a ray start. The use of the existing LP solver, which has a fast sparse linear algebra engine and pivoting method, as well as the use of sparse linear algebra engine for complementary pivoting enables PathAVI to fully exploit the sparse representation of the given AVI. This makes our method efficient for large-scale AVI problems as illustrated by the examples in Section 3.6.2. More details on the overall computational procedure are given in the Appendix as Algorithm 2.

We start with an introduction to some terminology and notational conventions for describing a basic solution of an LP problem. We follow notation used in [9]. Suppose that we run an LP solver over an LP problem: minimize $c^{\mathrm{T}}z$ subject to $Az - b \in K$ and

$l \leq z \leq u$. Without loss of generality, we assume that we have eliminated all fixed variables. For each solution z obtained from the LP solver, we have four index sets, B, N_l, N_u, and N_{fr}, for variables and two index sets, $\mathcal{A}$ and $\bar{\mathcal{A}}$, for constraints described by A and b[1]. Table 3.1 lists the properties of the index sets and the solution z. In Table 3.1, if $l_B \leq z_B \leq u_B$, we say that z is a basic feasible solution. Otherwise, we say that z is a basic solution. Note that we have $|\mathcal{A}| = |B|$ in Table 3.1 as the basis matrix $\mathbf{B}$ is invertible. Hence, the submatrix $A_{\mathcal{A}B}$ of $\mathbf{B}$ is square and invertible.

We first describe how to compute an implicit extreme point of C. For a given AVI(C, q, M), we formulate and solve the following LP problem using an LP solver:

$$\begin{aligned} \text{minimize} \quad & 0^{\mathrm{T}} z \\ \text{subject to} \quad & Az - b \in K \\ & l \leq z \leq u \end{aligned} \tag{LP}$$

We put zero objective coefficients in the (LP) so that the (LP) returns whenever it finds a basic feasible solution. If we have an intuition about where to start complementary pivoting, then we could try to solve the (LP) with different objective coefficients.

Assuming that the (LP) is feasible, a basic feasible solution z^0 from the LP solver with the corresponding index sets is an extreme point if $N_{fr} = \emptyset$. When $N_{fr} \neq \emptyset$, z^0 might not be an implicit extreme point. In this case, we move from z^0 to an implicit extreme point by doing additional pivoting in a way that forces as many nonbasic free variables to become basic variables. Algorithm 1 in the Appendix describes the pivoting procedure. After applying Algorithm 1, for each $j \in N_{fr}$ and $d^j = A_{\mathcal{A}B}^{-1} A_{\mathcal{A},j}$ if there exists k such that $d_k^j \neq 0$,

[1]These index sets can be obtained using `CPXgetbase()` for CPLEX, for example.

Table 3.1: Index sets and a basis matrix describing a basic solution z of an LP problem. Assume that $z \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$

$B \cup N_l \cup N_u \cup N_{fr} = \{1, \ldots, n\}$ and $B, N_l, N_u,$ and N_{fr} are mutually exclusive.
$B :=$ a set of basic variables indices
$N_l :=$ a set of nonbasic variables indices at their finite lower bounds
$N_u :=$ a set of nonbasic variables indices at their finite upper bounds
$N_{fr} :=$ a set of nonbasic free variables indices
$\mathcal{A} \cup \bar{\mathcal{A}} = \{1, \ldots, m\}$ with $\mathcal{A} \cap \bar{\mathcal{A}} = \emptyset$
$\mathcal{A} :=$ a set of active constraints indices, i.e., $A_{\mathcal{A}\bullet} z = b_{\mathcal{A}}$
$\bar{\mathcal{A}} :=$ a set of inactive constraints indices
$\mathbf{B} = \begin{bmatrix} A_{\mathcal{A}B} & 0 \\ A_{\bar{\mathcal{A}}B} & \pm I_{\bar{\mathcal{A}}} \end{bmatrix}$ is an invertible basis matrix where $I_{\bar{\mathcal{A}}}$ is an identity matrix of size $|\bar{\mathcal{A}}| \times |\bar{\mathcal{A}}|$
$z_B = A_{\mathcal{A}B}^{-1}(b_{\mathcal{A}} - A_{\mathcal{A}N} z_N)$, $z_{N_l} = l_{N_l}$, $z_{N_u} = u_{N_u}$, $z_{N_{fr}} = 0$, $N = N_l \cup N_u \cup N_{fr}$

then the basic variable corresponding to the kth position in B is a free variable. Otherwise, the variable z_j must have been pivoted in by Algorithm 1. Also, note that Algorithm 1 doesn't change the properties described in Table 3.1. Using Algorithm 1, we obtain the following result.

Proposition 3.21. *Suppose that we have applied Algorithm 1. Then the new point, denoted by $\bar{z}^0$, constructed from z^0 through Algorithm 1 is an implicit extreme point of C. We have* $\dim(\operatorname{lin} C) = |N_{fr}|$ *and the following set of vectors is a basis for the lineality space of C:*

$$\bigcup_{j\in N_{fr}} \{v^j\}, \quad v_k^j = \begin{cases} (A_{\mathcal{A}B}^{-1} A_{\mathcal{A},j})_k & \text{if } k \in B, \\ 0 & \text{if } k \in N_l \cup N_u, \\ 0 & \text{if } k \in N_{fr}, k \neq j, \\ 1 & \text{if } k = j. \end{cases}$$

Proof. Clearly, $\bar{z}^0 \in C$ as we do a ratio test to move the point. We first show that $\operatorname{lin} C = |N_{fr}|$ and $\{v^j\}_{j\in N_{fr}}$ is a basis for the lineality space of C. For each $j \in N_{fr}$, if $v_k^j \neq 0$ for

$k \in B$, then we have $l_k = -\infty$ and $u_k = \infty$ as discussed in the previous paragraph. It follows that $\bar{z}^0 + \lambda v^j \in C$ for all $\lambda \in \mathbb{R}$. By [71, Theorem 8.3], $v^j \in \operatorname{rec} C \cap (-\operatorname{rec} C)$. Thus $v^j \in \operatorname{lin} C$. By construction of v^j, we see that v^j's are linearly independent. This implies that $\dim(\operatorname{lin} C) \geq |N_{fr}|$. As $\dim(N_C(\bar{z}^0)) \geq |B| + |N_l| + |N_u|$ and $N_C(\bar{z}^0) \subset (\operatorname{lin} C)^\perp$ as shown in Proposition 3.13(d), it follows that $\dim(\operatorname{lin} C) = |N_{fr}|$ and $\{v^j\}_{j \in N_{fr}}$ is a basis for the lineality space of C.

We now prove that $\bar{z}^0$ is an implicit extreme point of C. Suppose that $\bar{z}^0 = \lambda z^1 + (1-\lambda) z^2$ for some $z^1, z^2 \in C$ and $\lambda \in (0,1)$. Define $d^k = \sum_{j \in N_{fr}} (-z_j^k v^j)$ and set $\tilde{z}^k = z^k + d^k$ for $k = 1, 2$. We then have $\tilde{z}_j^k = 0$ for $j \in N_{fr}$ and $\tilde{z}^k \in C$ as $d^k \in \operatorname{lin} C$ for $k = 1, 2$. As $\bar{z}^0 = \lambda z^1 + (1-\lambda) z^2$, $\bar{z}^0 = \lambda \tilde{z}^1 + (1-\lambda)\tilde{z}^2 - (\lambda d^1 + (1-\lambda) d^2)$. We have $\lambda d^1 + (1-\lambda) d^2 = \sum_{j \in N_{fr}} \left(-(\lambda z_j^1 + (1-\lambda) z_j^2) v^j \right)$. As $\bar{z}^0_{N_{fr}} = \tilde{z}^1_{N_{fr}} = \tilde{z}^2_{N_{fr}} = 0$, $v_j^j = 1$, and $v_h^j = 0$ for $h \in N_{fr}, h \neq j$, we see that $\lambda d^1 + (1-\lambda) d^2 = 0$. Therefore, $\bar{z}^0 = \lambda \tilde{z}^1 + (1-\lambda)\tilde{z}^2$. It follows that $\bar{z}^0 = \tilde{z}^1 = \tilde{z}^2$. Thus, $\bar{z}^0 - z^k = d^k \in \operatorname{lin} C$ for $k = 1, 2$, which implies that $\bar{z}^0$ is an implicit extreme point of C. □

Using the implicit extreme point $\bar{z}^0$ of C and the index sets $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$ associated with it, we construct an initial complementary basis and compute an implicit extreme point satisfying the sufficient conditions for a ray start from that complementary basis. To prove the invertibility of our initial complementary basis, we first introduce the following technical result derived from [54, Lemma 3.6].

Corollary 3.22. *Suppose that we have index sets* $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$ *associated with an* AVI(C, q, M) *with a nonempty* N_{fr}*. Then* Z *is invertible if and only if* $\tilde{W}^{\mathrm{T}} \tilde{M} \tilde{W}$ *is invertible,*

where

$$Z = \begin{bmatrix} M_{BB} & M_{BN_{fr}} & -A_{\mathcal{A}B}^{\mathrm{T}} \\ M_{N_{fr}B} & M_{N_{fr}N_{fr}} & -A_{\mathcal{A}N_{fr}}^{\mathrm{T}} \\ A_{\mathcal{A}B} & A_{\mathcal{A}N_{fr}} & 0_{\mathcal{A}} \end{bmatrix}, \quad \tilde{M} = \begin{bmatrix} M_{BB} & M_{BN_{fr}} \\ M_{N_{fr}B} & M_{N_{fr}N_{fr}} \end{bmatrix}, \quad \tilde{W} = \begin{bmatrix} -A_{\mathcal{A}B}^{-1} A_{\mathcal{A}N_{fr}} \\ I_{N_{fr}} \end{bmatrix}.$$

Proof. As $A_{\mathcal{A}B}$ is square and invertible, $\ker \begin{bmatrix} A_{\mathcal{A}B} & A_{\mathcal{A}N_{fr}} \end{bmatrix} = \operatorname{im} \tilde{W}$. The result follows from [54, Lemma 3.6]. □

We are now ready to present our initial complementary basis and an implicit extreme point satisfying the sufficient conditions for a ray start.

Proposition 3.23. *For a given* AVI(C, q, M)*, suppose that we have an implicit extreme point* $\bar{z}^0$ *and the index sets* $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$ *associated with* $\bar{z}^0$*. Then the matrix on the left-hand side of the following system of equations is invertible if and only if* M *is invertible on the lineality space of* C*. Also* $z = (z_B, z_{N_{fr}}, \bar{z}^0_{N_l}, \bar{z}^0_{N_u})$ *in a solution to the system of equations satisfies* $z \in \bar{z}^0 + \operatorname{lin} C$*, i.e.,* z *is an implicit extreme point of* C *by Proposition 3.9 in Section 3.3, and* $Mz + q \in \operatorname{aff}(N_C(z))$*.*

$$\begin{bmatrix} M_{BB} & M_{BN_{fr}} & -A_{\mathcal{A}B}^{\mathrm{T}} & 0 & 0 & 0 \\ M_{N_lB} & M_{N_lN_{fr}} & -A_{\mathcal{A}N_l}^{\mathrm{T}} & -I_{N_l} & 0 & 0 \\ M_{N_uB} & M_{N_uN_{fr}} & -A_{\mathcal{A}N_u}^{\mathrm{T}} & 0 & I_{N_u} & 0 \\ M_{N_{fr}B} & M_{N_{fr}N_{fr}} & -A_{\mathcal{A}N_{fr}}^{\mathrm{T}} & 0 & 0 & 0 \\ A_{\mathcal{A}B} & A_{\mathcal{A}N_{fr}} & 0 & 0 & 0 & 0 \\ A_{\bar{\mathcal{A}}B} & A_{\bar{\mathcal{A}}N_{fr}} & 0 & 0 & 0 & -I_{\bar{\mathcal{A}}} \end{bmatrix} \begin{bmatrix} z_B \\ z_{N_{fr}} \\ \lambda_{\mathcal{A}} \\ w_{N_l} \\ v_{N_u} \\ s_{\bar{\mathcal{A}}} \end{bmatrix} = \begin{bmatrix} -q_B - M_{BN}\bar{z}^0_N \\ -q_{N_l} - M_{N_lN}\bar{z}^0_N \\ -q_{N_u} - M_{N_uN}\bar{z}^0_N \\ -q_{N_{fr}} - M_{N_{fr}N}\bar{z}^0_N \\ b_{\mathcal{A}} - A_{\mathcal{A}N}\bar{z}^0_N \\ b_{\bar{\mathcal{A}}} - A_{\bar{\mathcal{A}}N}\bar{z}^0_N \end{bmatrix}.$$

Proof. The matrix on the left-hand side of the system of equations is invertible if and only if the matrix Z defined in Corollary 3.22 is invertible. This is because of the identity submatrices of it, $-I_{N_l}$, I_{N_u}, and $-I_{\bar{\mathcal{A}}}$. The columns of the matrix $W = (-A_{\mathcal{A}B}^{-1}A_{\mathcal{A}N_{fr}} \quad I_{N_{fr}} \quad 0_{N_l} \quad 0_{N_u})^{\mathrm{T}}$ form a basis of the lineality space of C. Note that $W^{\mathrm{T}}MW = \tilde{W}^{\mathrm{T}}\tilde{M}\tilde{W}$ where $\tilde{W}$ and $\tilde{M}$ are the matrices defined in Corollary 3.22. Therefore, the matrix is invertible if and only if M is invertible on the lineality space of C.

We now show that the z constructed from the solution to the linear system satisfies $z \in \bar{z}^0 + \operatorname{lin} C$. The fifth equation gives us

$$z_B = -A_{\mathcal{A}B}^{-1}A_{\mathcal{A}N_{fr}}z_{N_{fr}} + A_{\mathcal{A}B}^{-1}(b_{\mathcal{A}} - A_{\mathcal{A}N}\bar{z}_N^0).$$

If $z_{N_{fr}} = 0$, then $z_B = \bar{z}_B^0$ and $z = \bar{z}^0$. For $z_{N_{fr}} \neq 0$, we have $z = \bar{z}^0 + Wz_{N_{fr}}$. As W is a basis for the lineality space of C, it follows that $z \in \bar{z}^0 + \operatorname{lin} C$. Since $\bar{z}^0$ is an implicit extreme point, z enjoys the same property by Proposition 3.9.

From the first four equations of the given system, we see that $Mz + q \in \operatorname{aff}(N_C(z))$. □

3.5 Worst-case performance comparison: AVI vs MCP reformulation

In this section, we introduce the MCP reformulation of an AVI and analyze worst-case performance of the two formulations in Sections 3.5.1 and 3.5.2, respectively. We assume that both problems are solved using the same complementary pivoting method. Computational

results comparing the two formulations are presented in Section 3.6, and demonstrate the effectiveness of working on the original manifold $\mathcal{M}_C$ (see Tables 3.2–3.4 and Fig. 3.3).

3.5.1 MCP reformulation

A linear MCP is defined as follows: for an affine function $F(z) = Mz + q$ and a box constraint $B_1 := \Pi_{j=1}^{n}[l_j, u_j]$, z is a solution to the MCP(B_1, q, M) if $Mz + q = w - v$ with $z \in B_1, w, v \in \mathbb{R}^n_+, (z-l)^{\mathrm{T}}w = 0$, and $(u-z)^{\mathrm{T}}v = 0$.

It is well known [20, page 4] that an AVI(C, q, M) can be reformulated as an MCP$(B_1 \times B_2, \tilde{q}, \tilde{M})$, where

$$\begin{aligned} &B_1 = \Pi_{j=1}^{n}[l_j, u_j], \quad B_2 = \{\lambda \in \mathbb{R}^m \mid \lambda \in K^D\}, \\ &\tilde{M} = \begin{bmatrix} M & -A^{\mathrm{T}} \\ A & 0 \end{bmatrix}, \quad \tilde{q} = \begin{bmatrix} q \\ -b \end{bmatrix}. \end{aligned} \tag{MCP-reform}$$

By [30, Proposition 1.2.1], z^* is a solution to the AVI(C, q, M) if and only if there exists λ^* such that (z^*, λ^*) is a solution to the MCP$(B_1 \times B_2, \tilde{q}, \tilde{M})$. Therefore, we can solve an AVI(C, q, M) by solving its MCP$(B_1 \times B_2, \tilde{q}, \tilde{M})$ reformulation and vice versa. The solver Path [19], one of the most efficient MCP solvers, uses this MCP reformulation when it processes an AVI.

Although the two formulations are equivalent, they do not share the same theoretical properties. This is mainly because they look at different feasible regions, which also results in different PL manifolds on which the complementary pivoting is performed. For the MCP$(B_1 \times B_2, \tilde{q}, \tilde{M})$ reformulation, a PL $(n+m+1)$-manifold $\mathcal{M}_{B_1 \times B_2}$ is built where the full-dimensional cells are defined by the nonempty faces and the normal cones of the set

$B_1 \times B_2$, which doesn't consider the polyhedral constraints $Az - b \in K$. For the AVI(C, q, M) formulation, a PL $(n+1)$-manifold $\mathcal{M}_C$ is constructed based on the nonempty faces and normal cones of C, which includes the polyhedral constraints $Az - b \in K$ explicitly.

3.5.2 Worst-case performance analysis

In the worst case, the complementary pivoting method ends up going through all the full-dimensional cells of the underlying PL manifold. As each iteration of the complementary pivoting method corresponds to the traversal of one full-dimensional cell assuming nondegeneracy or lexicographic pivoting, the maximum number of iterations is the total number of the full-dimensional cells, which is finite but could be exponential in the number of constraints. Therefore, we compare worst-case performance of the two formulations by counting the number of the full-dimensional cells of the PL manifold that each formulation generates.

By construction, the number of the full-dimensional cells is equivalent to the number of the nonempty faces of the polyhedral convex set being considered [67, page 6]. Thus, we count the number of the nonempty faces of both $B_1 \times B_2$ and C.

Let NNF(S) denote the number of the nonempty faces of a polyhedral convex set S. To count the number of the nonempty faces, we start with building blocks defining a polyhedral convex set: intervals $[l, u]$ in $\mathbb{R}$ and linear constraints $a^{\mathrm{T}} z - b \in K$. For a closed interval

$[l, u]$ in $\mathbb{R}$, the number of the nonempty faces is as follows:

$$\mathrm{NNF}([l,u]) = \begin{cases} 1 & \text{if } -\infty = l < u = \infty \text{ or } -\infty < l = u < \infty, \\ 2 & \text{if } -\infty = l < u < \infty \text{ or } -\infty < l < u = \infty, \\ 3 & \text{if } -\infty < l < u < \infty. \end{cases} \tag{3.13}$$

For a halfspace or a hyperplane defined by a linear constraint $a^{\mathrm{T}}z - b \in K$ where $a \neq 0$ and $b \in \mathbb{R}$, the number of the nonempty faces is as follows:

$$\mathrm{NNF}(\{z \in \mathbb{R}^n \mid a^{\mathrm{T}}z - b \in K\}) = \begin{cases} 2 & \text{if } K = \mathbb{R}_+ \text{ or } K = \mathbb{R}_-, \\ 1 & \text{if } K = \{0\}. \end{cases} \tag{3.14}$$

Based on (3.13) and (3.14), we can compute an upper bound on the number of the nonempty faces of a polyhedral convex set.

Lemma 3.24. *Let C be a polyhedral convex set defined by $C = \{z \in \mathbb{R}^n \mid Az - b \in K, l \leq z \leq u\}$. Then*

$$NNF(C) \leq \Pi_{j=1}^{n} NNF([l_j, u_j]) \times \Pi_{i=1}^{m} NNF(\{z \in \mathbb{R}^n \mid A_{i\bullet}^{\mathrm{T}}z - b_i \in K_i\}), \tag{3.15}$$

where the symbol Π_j denotes multiplication over indexed terms.

Proof. Let $C_j = \{z \in \mathbb{R}^n \mid z_j \in [l_j, u_j]\}$ for $j = 1, \ldots, n$ and $C_{n+i} = \{z \in \mathbb{R}^n \mid A_{i,:}^{\mathrm{T}}z - b_i \in K_i\}$ for $i = 1, \ldots, m$. Then $C = \cap_{i=1}^{n+m} C_i$. By [70, Corollary 4.2.15], F is a face of C if and only if $F = \cap_{i=1}^{n+m} F_i$ where F_i is a face of C_i for $i = 1, \ldots, n + m$. The result follows. □

In Lemma 3.24, there could be a large gap between $\mathrm{NNF}(C)$ and its upper bound. The

upper bound counts all the possible combinations of the faces of each constraint regardless of their feasibility. When C has only box constraints, i.e., $C = \{z \in \mathbb{R}^n \mid l \leq z \leq u\}$, then equality holds in (3.15). But, in other cases, the upper bound could be much larger than NNF(C) as not every combination corresponds to a nonempty face of C. For example, if $C = \{z \in \mathbb{R}^2 \mid z_1 + z_2 \geq -1, -z_1 + z_2 \geq -1, z_1 - z_2 \geq -1, -z_1 - z_2 \geq -1, -1 \leq z_1, z_2 \leq 1\}$, we have NNF($C$) = 9. However, the upper bound is 144. It turns out that there are many infeasible combinations, i.e., all the combinations having $z_1 = -1$ and $z_2 = 1$.

Using Lemma 3.24, we prove that the maximum number of cells for the AVI(C, q, M) manifold is smaller or equal to the cells in the MCP($B_1 \times B_2, \tilde{q}, \tilde{M}$) manifold.

Proposition 3.25. *Let an* AVI(C, q, M) *formulation and its* MCP($B_1 \times B_2, \tilde{q}, \tilde{M}$) *reformulation defined in* (MCP-reform) *be given. Then the number of the full-dimensional cells of the* PL $(n+1)$*-manifold* $\mathcal{M}_C$ *is less than or equal to the number of the full-dimensional cells of the* PL $(n+m+1)$*-manifold* $\mathcal{M}_{B_1 \times B_2}$.

Proof. By [70, Proposition 4.2.12], NNF($B_1 \times B_2$) = NNF(B_1) $\times$ NNF(B_2). By applying the same proposition, we have NNF(B_1) = $\Pi_{j=1}^{n}$NNF($[l_j, u_j]$) and NNF(B_2) = $\Pi_{i=1}^{m}$NNF($[l_i^\lambda, u_i^\lambda]$) where l_i^λ and u_i^λ are lower and upper bounds on λ_i variable. Using (3.13) and (3.14), we see that $\Pi_{i=1}^{m}$NNF($\{z \in \mathbb{R}^n \mid A_{i\bullet}^{\mathrm{T}} z - b_i \in K_i\}$) = $\Pi_{i=1}^{m}$NNF($[l_i^\lambda, u_i^\lambda]$). By Lemma 3.24, the result follows. □

Based on Proposition 3.25, we expect that PathAVI will take fewer iterations than Path, which solves the MCP reformulation, since in the worst case both may visit every cell in the manifold. This is confirmed by the computational results in Sections 3.6.3–3.6.5.

3.6 Computational results

In this section, we present computational results of PathAVI highlighting its computational benefits of preserving the problem structure and its robustness and efficiency compared to Path version 4.7 [19, 37], an established solver for AVIs which uses the MCP reformulation. The majority of the examples are based on friction contact models, which we briefly describe in Section 3.6.1. Section 3.6.2 shows improved performance of PathAVI using the original AVI formulation containing nontrivial lineality space over its equivalent reduced form that does not contain lines. Sections 3.6.3–3.6.5 compare performance of PathAVI and Path over friction contact problems, compact sets, and Nash equilibrium problems, respectively and demonstrate the advantages of the stronger theory associated with PathAVI.

All experiments were performed on a Linux machine with Intel Xeon(R) E7-4850 2.00GHz processor and 256GB of memory. PathAVI was compiled using GNU gcc version 4.4.7 and its interfaces were linked to GAMS. All problem instances were written in GAMS using the EMP syntax for variational inequalities [33]. We set the time limit to 1 hour and major/minor iteration limits to 20 and 10^5, respectively.

3.6.1 Friction contact problem

Coulomb or dry friction is a ubiquitous phenomenon when mechanical systems interact via contact with each other. Consider a mechanical system with n_{dof} degrees of freedom, n_d bodies and n_c contacts. The number of degrees of freedom depends on the type of system we consider, i.e., if we have rigid bodies, $n_{\text{dof}} = 6n_d$. However, if we have deformable bodies, then this number is typically larger and depends on the modeling used for the bodies. For

each two bodies in contact at a single point, we denote by $u^{(k)} \coloneqq (u_n^{(k)}, u_t^{(k)})^{\mathrm{T}} \in \mathbb{R}_+ \times \mathbb{R}^2$ the relative (or local) velocity between them and the reaction force is given by $r^{(k)} = (r_n^{(k)}, r_t^{(k)})$. One of the numerous ways (see [2] for a list of them) to model the dynamics of a system with Coulomb friction is:

$$\begin{aligned} -u_n^{(k)} &\in N_{\mathbb{R}_+}(r_n^{(k)}) \qquad k = 1, \dots, n_c \qquad \text{and} \qquad Mv = Hr + f \\ -u_t^{(k)} &\in N_{r_n^{(k)}\mu^{(k)}D}(r_t^{(k)}) \qquad\qquad\qquad\qquad\quad u = H^T v + w, \end{aligned} \tag{3.16}$$

with $M \in \mathbb{R}^{n_{\mathrm{dof}} \times n_{\mathrm{dof}}}$, $H \in \mathbb{R}^{n_{\mathrm{dof}} \times 3n_c}$, $\mathbb{R}^{3n_c} \ni r \coloneqq [r_n^{(1)}, r_t^{(1)}, \dots, r_n^{(n_c)}, r_t^{(n_c)}]^T$ and $\mathbb{R}^{3n_c} \ni u \coloneqq [u_n^{(1)}, u_t^{(1)}, \dots, u_n^{(n_c)}, u_t^{(n_c)}]^T$, see Fig. 3.1 for an example. It is shown in [46] that (3.16) is equivalent to the following complementarity problem over a second order cone:

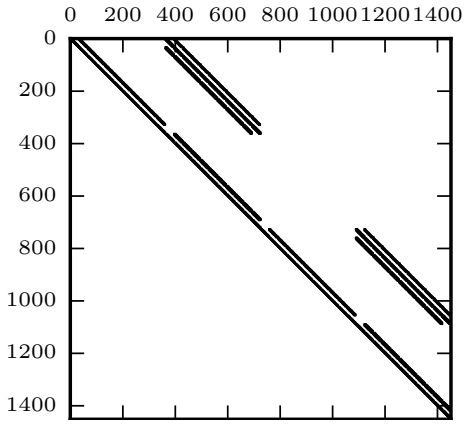
$$0 \in \begin{pmatrix} M & -H & 0 \\ H^{\mathrm{T}} & 0 & E \\ \bar{H}^{\mathrm{T}} & 0 & E \end{pmatrix} \begin{pmatrix} v \\ r \\ y \end{pmatrix} + \begin{pmatrix} -f \\ w \\ \bar{w} \end{pmatrix} + N_X \begin{pmatrix} v \\ r \\ y \end{pmatrix} \quad X \coloneqq \mathbb{R}^{n_{\mathrm{dof}}} \times K \times K, \tag{3.17}$$

where $K \coloneqq \Pi_{k=1}^{n_c} K_{\mu^k}$ and $K_{\mu^k} \coloneqq \{(t, tx) \mid t \in \mathbb{R}_+, x \in \mu^k D\}$, D being the unit disk in $\mathbb{R}^2$. If we split H as $[H_{1,n}, H_{1,t}, \dots, H_{i,n}, H_{i,t}, \dots, H_{n_c,n}, H_{n_c,t}]$ with $H_{k,n} \in \mathbb{R}^{n_{\mathrm{dof}}}, H_{k,t} \in \mathbb{R}^{n_{\mathrm{dof}} \times 2}$, then $\bar{H} \coloneqq [0_n, H_{1,t}, \dots, 0_n, H_{k,t}, \dots, 0_n, H_{n_c,t}]$. Similarly, letting $(w_{k,n}, w_{k,t}) \in \mathbb{R} \times \mathbb{R}^2$ we have $w = [w_{1,n}, w_{1,t}, \dots, w_{n_c,n}, w_{n_c,t}]$ and $\bar{w} \coloneqq [0, w_{1,t}, \dots, 0, w_{n_c,t}]$. Finally, the diagonal matrix $E \in \mathbb{R}^{3n_c \times 3n_c}$ is based on the vector $(1, 0, 0)^T$ repeated n_c times. The variable $y \coloneqq [y_{1,n}, y_{1,t}, \dots, y_{n_c,n}, y_{n_c,t}]^T$ with $(y_{k,n}, y_{k,t}) \in \mathbb{R} \times \mathbb{R}^2$, is introduced to ensure that the modified local velocity $u + Ey$ belongs to K^D. Since the cone K is not polyhedral, we need to approximate K to get an AVI from (3.17). Then, we have to solve a sequence of AVIs until one of the solutions also satisfies (3.17) up to the specified tolerance. Computationally, the

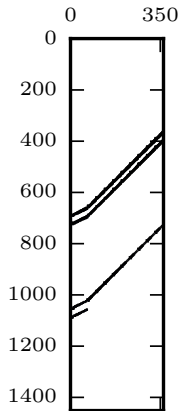
most demanding step is the solution of the first AVI in the sequence. Furthermore, we focus here on the case where it makes sense to perform a ray start. Hence, we solve the AVI that would correspond to the first iteration and with an "anisotropic" approximation of K. For each contact we construct a finitely representable polyhedral approximation D_p^k of the disk $\mu^k D$. Then, the cone K is approximated by $K_p \coloneqq \Pi_k K_p^k$, with $K_p^k \coloneqq \{(t, tx) \mid t \in \mathbb{R}_+, x \in D_p^k\}$. Finally, with a slight abuse of notation, we redefine $X \coloneqq \mathbb{R}^{n_{\text{dof}}} \times K_p \times K_p$ and refer to (3.17) as an AVI. It can be verified that PathAVI processes the AVI (3.17) if $w \in (\ker H \cap K)^D$ by applying Lemma 3.17. It is noteworthy that this condition is exactly the one given in [51] for the existence of solution to the complementarity problem over a second order cone (3.17). If we solely rely on the L-matrix property, we need to assume that $\ker H = \{0\}$, which fails in many instances, for example when a 4-legged chair is in contact with flat ground.

3.6.2 Computational benefits of preserving the problem structure

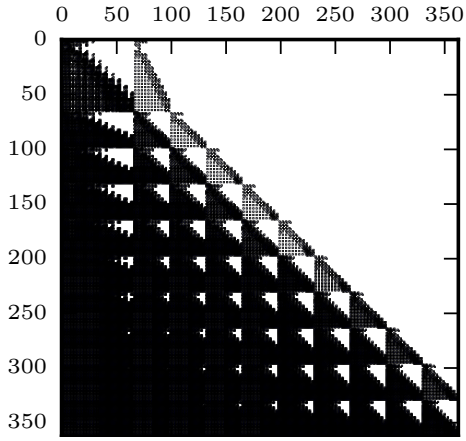
The problem data (M, H, f, w) for the following numerical results were obtained from simulations of deformable bodies with the LMGC90 [23] software and using a solver from Siconos [3]. In the following, we focus on a simple example where 2 deformable cubes are on top of another. During the simulation, the number of contacts varies between 80 and 120. The shape of M and H is given in Fig. 3.1. It is noteworthy that if we have to remove the lineality space, that is to compute W, then the sparse structure of the problem is destroyed (see Fig. 3.1(c)): the number of nonzero elements is increased by a factor of 5. It is expected that the linear algebra computations will be more expensive in the reduced space formulation than in the original one because of this large increase of nonzero



(a) Nonzero patterns of M



(b) Nonzero patterns of H



(c) Nonzero patterns of W

Figure 3.1: Nonzero patterns of the matrices M (size: 1452×1452, nnz: 11330), H (size: 1452×363, nnz: 1747) and $W := H^T M^{-1} H$ (size: 363×363, nnz: 56770).

entries. This has been verified on instances that have the same kind of structure as the matrices depicted in Fig. 3.1. Both problems are AVIs, but as shown on Fig. 3.2, PathAVI working in the original space is always faster and most of the time is at least twice as fast as PathAVI working in the reduced space. The time in the reduced space does not even take into account the transformation of the problem data, that is the computation of the W matrix.

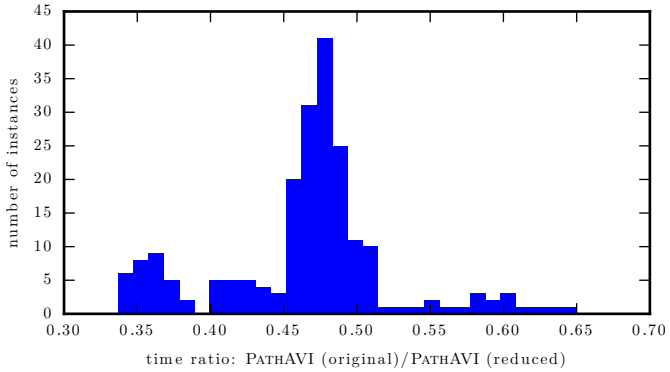


Figure 3.2: Comparison in terms of speed between the resolution in the original space and the reduced one. The number of iteration was the same for all the 209 instances.

3.6.3 Multibody friction contact problems

When the bodies are rigid, it is common in the contact mechanic community to eliminate the velocity v. The problem is formulated in a reduced space $K_p \times K_p$ (defined in Section 3.6.1) and the AVI is

$$0 \in \begin{pmatrix} W & E \\ \bar{W} & E \end{pmatrix} \begin{pmatrix} r \\ y \end{pmatrix} + \begin{pmatrix} \omega \\ \bar{\omega} \end{pmatrix} + N_{K_p \times K_p} \begin{pmatrix} r \\ y \end{pmatrix}, \tag{3.18}$$

where $W := H^{\mathrm{T}} M^{-1} H$ and $\bar{W} := \bar{H}^{\mathrm{T}} M^{-1} H$, $\omega := w + H^{\mathrm{T}} M^{-1} f$ and $\bar{\omega} := \bar{w} + \bar{H}^{\mathrm{T}} M^{-1} f$. The lineality space is then trivial in this formulation.

We present computational results using the problem data (W, μ and q) from the FCLIB collection[2] [1], which aims at providing challenging instances of the friction contact problem. Let us highlight a few facts based on the data presented in Table 3.2:

- PathAVI can solve all the instances with the linear algebra package UMFPACK [18] ("pathavi/UMFPACK") and is generally faster than Path.

[2]The collection of problem can be freely downloaded by visiting `http://fclib.gforge.inria.fr`

- Some problems are numerically challenging and the behavior of the solver changes with the linear algebra routines. Specifically, on those problems using LUSOL ("pathavi/default") leads to 20 failures. That can be reduced by using the block-LU updates [26] ("pathavi/LUSOL-blu"). These errors are caused by some numerical issues in the linear algebra package. This illustrates the importance of being able to change the linear algebra engine in PathAVI.

- Path is unable to perform a ray start in many instances (whenever ker W is not trivial); in these cases, PathAVI significantly outperforms Path (with or without the crash method).

Table 3.2: Statistics for 4579 friction contact problems of the form (3.18).

Solver/profile	# Failed	Failure type			
		Solver error	Stalled	Time	Iteration
pathavi/UMFPACK	0	0	0	0	0
pathavi/default	20	0	0	0	20
pathavi/LUSOL-blu	3	0	0	0	3
path/default	2060	535	1525	0	0
path/no crash	108	99	0	8	1

Let us explain the failure types: "Solver error" means that the first basis matrix could not be factorized, despite the use of artificial variables to overcome the rank deficiency. "Stalled" means that a solver tried various strategies but failed to find a solution at the requested accuracy and consequently gave up. Note that this never occurred with PathAVI on this set of problems. "Time" (or "Iteration") signals that the time (or iteration) limit has been reached. The convergence tolerance is set to a low value: $\sqrt{N} \cdot 10^{-9}$, where N is the

number of contacts. This value is lower than the default tolerance of PATH (that is already considered quite demanding).

The default behavior of PATH ("path/default") leads to many failures: the crash method is inappropriate for such models. However, even without the crash procedure ("path/no crash"), PATH still fails at a higher rate than PATHAVI. We further compare PATH and PATHAVI on their default settings on the subset of problems solved by both. The results are presented in Fig. 3.3 in terms of time ratios. First note that PATHAVI is faster than PATH in the majority of cases, and that it usually finds a solution in less than half the time of PATH. The spike on the right plot, when PATH finds the solution faster than PATHAVI, is explained by the fact that the crash procedure in PATH performed well in those instances (< 10% of the examples).

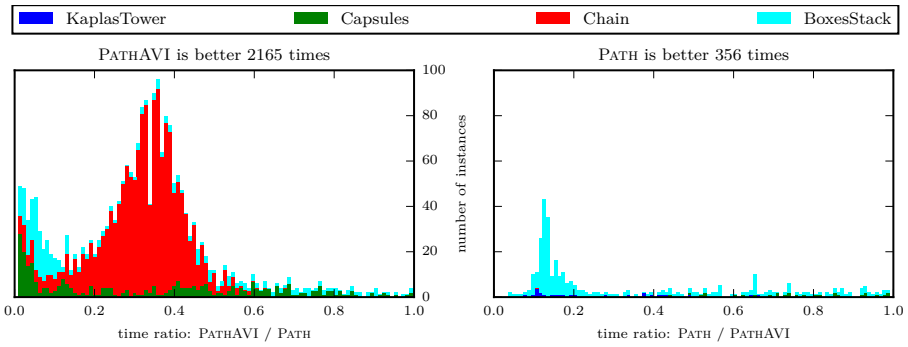


Figure 3.3: Time comparison between PATH and PATHAVI

3.6.4 AVIs over compact sets

One strong implication of Theorem 3.16 is that when C is compact (so that $\operatorname{rec} C = \{0\}$) PATHAVI can process an AVI(C, q, M) with arbitrary M and q. In contrast, this does not hold for the MCP reformulation as the underlying feasible region $[B_1 \times B_2]$ of it may not be compact although C is compact. This is because whenever the AVI contains polyhedral

constraints the associated λ variables in the MCP reformulation are only constrained to lie in the unbounded set B_2.

We construct 5 AVI instances by taking compact feasible regions from [57] having finite lower and upper bounds and by randomly generating M and q such that the resultant AVI has an M with negative eigenvalues.

Table 3.3 presents some computational results. As expected, PathAVI is able to solve all the instances, whereas Path fails to solve three of them. Also, on the two problem instances where both solvers are able to solve, PathAVI shows 10–30 times fewer iterations, and a similarly decreased elapsed time. These properties hold for a wide selection of instances and the above table is just provided for expository purposes.

Table 3.3: Performance of PathAVI and Path over compact sets

(a) Statistics of the compact sets

Name	(#constrs,#vars)	(nnz(A),nnz(M))
CVXQP1_M	(500, 1000)	(2495, 999)
CVXQP2_M	(250, 1000)	(1746, 999)
CVXQP3_M	(750, 1000)	(3244, 999)
CONT-050	(2401, 2597)	(14597, 6407)
CONT-100	(9801,10197)	(59197,98875)

(b) Performance of PathAVI and Path

Name	Number of iterations		Elapsed time (secs)	
	PathAVI	Path	PathAVI	Path
CVXQP1_M	3119	fail	0.459	fail
CVXQP2_M	33835	fail	2.827	fail
CVXQP3_M	360	3603	0.105	1.992
CONT-050	11	382	2.753	272.429
CONT-100	3	fail	174.267	fail

3.6.5 Nash equilibrium problems

Another application of AVIs is to Nash equilibrium problems. In a Nash equilibrium problem, there are multiple agents each of which minimizing its own objective function, and each agent's objective function not only depends on the agent's decision but also other agents' decisions. For example, a typical Nash equilibrium problem computes a solution satisfying

$$x_i^* \in \underset{x_i \in X_i}{\arg\min}\ h_i(x_i, x_{-i}^*), \quad \text{for } i = 1, \ldots, N, \tag{NEP}$$

where we note that each ith agent's objective function h_i takes its own decision, denoted by x_i, and other agents' decisions, denoted by x_{-i}.

We generated 6 instances of Nash equilibrium problems, where each X_i is a polyhedral convex set and h_i is continuously differentiable in x and convex quadratic in x_i for each fixed x_{-i}. Specifically, h_i takes the following form:

$$h_i(x_i, x_{-i}) = \frac{1}{2} x_i^{\mathrm{T}} Q_i x_i + x_i^{\mathrm{T}} Q_{-i} x_{-i} + c_i^{\mathrm{T}} x_i + d_i^{\mathrm{T}} x_{-i},$$

where Q_i is symmetric positive definite.

In this case, x is a solution to (NEP) if and only if it is a solution to the AVI(C, q, M) where $Mx + q = (\nabla_{x_i} h_i(x))_{i=1}^N$ and $C = \Pi_{i=1}^N X_i$. The number of agents ranges from 10 to 300.

Table 3.4 presents performance of PathAVI and Path over the NEPs. The number of iterations of PathAVI is up to 11 times fewer than Path. Elapsed time shows similar results except for the last three instances. In those instances, LUSOL has a great difficulty in computing PathAVI's intermediate basis matrices. If we change the linear algebra engine to UMFPACK, the computation time significantly reduces. Regarding Path's performance on

Table 3.4: Performance of PathAVI and Path over the NEPs

(a) Statistics of the NEPs

Name	(#constrs,#vars)	(nnz(A),nnz(M))
vimod1	(554,1138)	(4744,22577)
vimod2	(910,1723)	(7935,46137)
vimod3	(1101,2226)	(9117,67634)
vimod4	(870,1828)	(62056,154332)
vimod5	(1327,2586)	(133527,274004)
vimod6	(2210,4359)	(207408,417810)

(b) # Iterations and elapsed time of PathAVI and Path on the NEPs

Name	Number of iterations			Elapsed time (secs)		
	PathAVI	Path	PathAVI/ UMFPACK	PathAVI	Path	PathAVI/ UMFPACK
vimod1	367	2087	367	0.372	4.129	0.437
vimod2	319	3570	319	1.098	24.134	0.645
vimod3	590	4278	590	3.208	60.553	1.639
vimod4	1343	6146	1343	127.194	66.427	18.319
vimod5	2167	2768	2167	327.970	325.558	40.285
vimod6	3522	4222	3522	2341.193	1841.642	109.960

the last three instances, we would like to point out that the proximal perturbation technique of Path, which solves a sequence of perturbed MCPs by adding $\epsilon_k I$ with $\epsilon_k \to 0$ as $k \to \infty$ to the matrix $\tilde{M}$ in (MCP-reform), plays a significant role in its performance. Adding positive diagonals elements changes the elimination sequence and makes linear algebra computations much faster and more stable. When we turn off the proximal perturbation, Path either gets much slower than PathAVI or fails to solve the instance.

3.7 PathVI: a nonlinear extension of PathAVI for nonlinear VIs

We briefly describe our nonlinear extension of PathAVI for nonlinear VI(C, F)s where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear function and C is a polyhedral convex set. Similar to AVI(C, q, M), we define a normal map $F_C : \mathbb{R}^n \to \mathbb{R}^n$ such that $F_C(x) := F(\pi_C(x)) + x - \pi_C(x)$. It can be easily verified that $F_C(x^*) = 0$ if and only if $z^* := \pi_C(x^*)$ with $x^* = z^* - F(z^*)$ is a solution to the VI(C, F). By using the following linear approximation $A_k(\cdot)$ of F_C at a point x^k [68, Section 4] (or equivalently linearizing the generalized equation $0 \in F(x) + N_C(z)$ [47, Definition 5]),

$$A_k(x) = F(\pi_C(x^k)) + F'(\pi_C(x^k))(\pi_C(x) - \pi_C(x^k)) + x - \pi_C(x), \tag{3.19}$$

we apply the Newton's method with a sub-problem at iteration k being an AVI(C, q^k, M^k) where $q^k = F(\pi_C(x^k)) - F'(\pi_C(x^k))\pi_C(x^k)$ and $M^k = F'(\pi_C(x^k))$. PathAVI is used to solve each AVI(C, q^k, M^k). Similar to Path, we apply non-monotone stabilization scheme combined with Fischer-Burmeister type merit function [38] for global convergence of the Newton's method. We have implemented PathVI and is available within GAMS.

One caveat is that Path and PathVI will take an identical path after the first iteration of the Newton's method unless Path perturbs the problem using proximal perturbation. This is because the feasible region C is defined by linear constraints, and the linearization performed over those constraints does not change them. Thus a solution to the first Newton step will be in C. After that, the Newton step will maintain feasibility throughout the

iterations. For example, PATH computes a linearization over the following function:

$$\tilde{F}(z, \lambda) = \begin{bmatrix} F(z) - A^\top \lambda \\ Az - b \end{bmatrix}. \tag{3.20}$$

Then the row corresponding $Az - b$ would not change by the linearization, and complementarity will guarantee feasibility of all iterates after the first iteration.

3.8 Conclusions

We have presented PathAVI, a structure-preserving pivotal method for affine variational inequalities. Compared to existing methods, PathAVI can process an AVI without applying any reduction or transformation to the problem data even if the underlying feasible region contains lines. PathAVI can process some newly generated problem classes from applications in friction contact as well as the existing problem class (L-matrices [14]). A computational method for finding a point satisfying sufficient conditions for a ray start is detailed. Through worst-case analysis, we have shown that exploiting polyhedral structure for solving affine variational inequalities is expected to show better performance than using a mixed complementarity problem reformulation. Computational results over friction contact and Nash equilibrium problems illustrate that PathAVI compares favorably with PATH both in terms of robustness and efficiency.

Appendix

Lemma 3.26 (Theorem 4.4 [14])**.** *Consider an* AVI(C, q, M) *with* $\lin C = \{0\}$ *and let* M *be semi-monotone with respect to* $\rec C$*. Suppose that an unbounded ray occurs. Then the value of the auxiliary variable* t *is constant on that ray and* Δz*, the variation in* z*, is nonzero and satisfies*

$$\Delta z \in \rec C, \qquad M\Delta z \in (\rec C)^D, \qquad \textit{and} \qquad \Delta z^{\mathrm{T}} M \Delta z = 0. \tag{3.21}$$

Proof. The fact that t is constant and that Δz is a solution to (3.21) follows from the first part of the proof of Theorem 4.4 in [14]. To see that the direction Δz is nonzero, we proceed by contradiction: at the current iterate (x^k, t^k) we have

$$G_C(x^k, t^k) = Mz^k + q + x^k - z^k - t^k r = 0.$$

Let x^{k+1} belong to the unbounded ray and suppose that $\Delta z = 0$:

$$G_C(x^{k+1}, t^k) = Mz^k + q + x^{k+1} - z^k - t^k r = 0.$$

It immediately follows that $x^{k+1} = x^k$. □

Algorithm 1 Pivoting to make as many nonbasic free variables as basic variables

Input: a basic feasible solution z^0 and its index sets $(B^0, N_l^0, N_u^0, N_{fr}^0, \mathcal{A}^0, \bar{\mathcal{A}}^0)$

Output: a basic feasible solution $\bar{z}^0$ and its index sets $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$

Set $\bar{z}^0 \leftarrow z^0$.
Set $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}}) \leftarrow (B^0, N_l^0, N_u^0, N_{fr}^0, \mathcal{A}^0, \bar{\mathcal{A}}^0)$.
Set `changed` $\leftarrow$ `true`.
while `changed` is `true` **do**
    Set `changed` $\leftarrow$ `false`.
    for each $j \in N_{fr}$ **do**
        Do a ratio test on the nonbasic column j over basic variables that are not free variables.
        if the ratio is finite **then**
            Pivot in the jth column into basis.
            Update $\bar{z}^0$ and its index sets $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$. ▷ $|N_{fr}| \leftarrow |N_{fr}| - 1$
            Set `changed` $\leftarrow$ `true`.
        end if
    end for
end while
return $\bar{z}^0$ and its index sets $(B, N_l, N_u, N_{fr}, \mathcal{A}, \bar{\mathcal{A}})$

Algorithm 2 Overall computation procedure of PathAVI

Input: AVI(C, q, M)

Output: One of the following: emptiness of C, a solution z^* to the AVI(C, q, M), or a secondary ray

Construct and solve the LP problem defined in (LP) using an LP solver.
if the LP solver determines that C is empty **then**
    return C is empty
end if
Let z^0 be the basic feasible solution returned by the LP solver.
Run Algorithm 1 with z^0 and its index sets to compute an implicit extreme point $\bar{z}^0$.
Construct and solve the complementary system of equations defined in Proposition 3.23 using $\bar{z}^0$ and its index sets.
if $(\bar{z}^0, \lambda, w, v, s)$ is feasible **then**
    Set $z^* \leftarrow \bar{z}^0$.
    return z^*.
else
    Choose $r \in \mathrm{ri}(N_C(\bar{z}^0))$ by referring to the active constraint set at $\bar{z}^0$.
    Augment a column $(r, 0)^{\mathrm{T}}$ with a t variable to the complementary system of equations.
    Compute an almost complementary feasible basis by pivoting in the t variable.
    Do complementary pivoting until either we find a solution z^* or a secondary ray is generated.
    return z^* if we have found a solution or a secondary ray otherwise.
end if

Chapter 4

SELKIE: a model transformation and distributed solver for structured equilibrium problems

We introduce SELKIE, a general-purpose solver for equilibrium problems described by a set of agents. It exploits problem structures, such as block structure and cascading dependency of interacting agents, in a flexible and adaptable way to achieve a more robust and faster solution path. To accomplish this, it transforms a given model into a set of structure-exploiting sub-models via structure analysis and taking into account user's knowledge. A diagonalization method is then applied to those sub-models possibly with parallel computations for making full use of computational resources. Depending on the configurations of sub-model generations and diagonalization method to use, various decomposition schemes can be instantiated. We can choose a sub-solver to use for each

sub-model so that a highly efficient solver can be employed tailored to a certain problem type. For stronger convergence results and numerical stability, primal and dual proximal perturbations are implemented. Examples illustrating the flexibility and effectiveness of SELKIE are given. SELKIE has been implemented and is available within GAMS/EMP.

4.1 Introduction

This chapter is concerned with a general-purpose solver, called SELKIE, for structured equilibrium problems. Equilibrium problems of interest are generalized Nash equilibrium problems (GNEP) and multiple optimization problems with equilibrium constraints (MOPEC) [10, 29, 44, 65]. SELKIE is suitable for large-scale and block-structured equilibrium problems as well as equilibrium problems in general.

While the framework discussed in Chapter 2 makes it much easier to access and model equilibrium problems, its main solution method was limited to the MCP formulation. As the formulation is oblivious of the original problem in this case, it does not fully exploit its structure such as independent groups of agents and cascading dependency between interacting agents. These structures allow us to decompose a given model into smaller sub-models, possibly amenable to parallel computations, thus to find a more robust and faster solution path. This motivates us to develop SELKIE.

SELKIE distinguishes itself by its adaptability and flexibility in transforming a model, applying parallelism, and choosing a sub-solver to use. It transforms a given equilibrium model into several different structure-exploiting sub-models to solve the problem. The way a model is transformed is determined by either our automatic structure detection

mechanism or user-supplied information. Different sub-models can be easily generated by adapting user's knowledge provided in an option file `selkie.opt`. This adaptability allows SELKIE to exploit problem structure to a great extent. Parallelism can be achieved whenever the transformed models are independently solvable so that we can make full use of the underlying resources on a machine. Finally, we can choose a sub-solver to use for each sub-model of the same type such as linear, nonlinear, quadratic constrained programming, and so on. This allows us to apply a highly efficient solver tailored to certain problem types. All of these features have been implemented to work with the EMP framework so that we can use SELKIE in modeling languages as a solver for equilibrium problems.

Our main approach to transforming a model is a mixture of grouping and decomposing agents. It transforms a given initial set of agents obtained from the `empinfo` file into another set of agents by grouping and decomposing them based on structure analysis and user-supplied information. SELKIE then applies its solution method to those newly constructed agents. This approach is quite flexible as it generalizes the existing MCP approach (by grouping all the agents) while allowing various decomposition schemes to be applied.

As a solution method, we use diagonalization [17, 42, 63]. Diagonalization is an iterative method that repeatedly solves each agent while other agents' variable values are kept fixed until convergence. It not only fits well with our abstraction of a problem being a set of agents, but also can generalize many iterative methods such as Dantzig-Wolfe decomposition, Benders decomposition, dynamic programming, and so on. Five variants including nonlinear Gauss-Seidel and Jacobi are implemented. We also allow our diagonalization to work in a recursive manner: for a group of agents we can apply

diagonalization for each agent in the group while it is also applied to groups of agents. This feature is especially useful when we solve some dynamic programming applications in economics.

Finally, SELKIE provides proximal perturbations to transform the model for stronger convergence properties and to resolve some practical issues on applying diagonalization such as unbounded and infeasible solutions. It can perturb primal or dual or both parts where they correspond to the objective/VI functions and the constraints, respectively. Primal perturbation prevents from having an unbounded solution or infeasible VI function and dual perturbation is for avoiding infeasible constraints. They can help us find a more robust solution path.

The rest of the chapter is organized as follows. Section 4.2 presents the architecture of SELKIE. The overall data flow together with its three major components are described. In Section 4.3, we introduce SELKIE's solution method, diagonalization, with its five different variants. We discuss some practical issues on applying them such as unbounded and infeasible solutions. Section 4.4 introduces our two main model transformation methods, agent grouping and proximal perturbations, with examples showing their effectiveness. In Sections 4.3-4.4, we show how we can use the functions mentioned in those sections using an option file. We conclude this chapter in Section 4.5.

4.2 Architecture of SELKIE

In this section, we present the overall process of SELKIE together with its basic components to solve an equilibrium problem described using the EMP framework [49]. As SELKIE (and

the EMP framework) is currently implemented to work with GAMS, we assume that the model is written in GAMS/EMP, that is, GAMS using the EMP framework. This will be extended to support other algebraic modeling languages such as AMPL and Julia.

Figure 4.1 depicts the architecture of Selkie. Selkie is basically composed of three components: i) model I/O, ii) model transformations, and iii) solution methods. Model I/O unit takes an equilibrium model written in GAMS/EMP, parses its `empinfo` file to identify each agent's problem description such as agent's ownership of equations and variables, and generates an in-memory representation of the original model. By an in-memory representation, we mean a set of opcodes for each equation.[1] These opcodes enable us to compute derivatives of an equation (a symbolic differentiation is employed) quite efficiently in transformations unit. It also has the ability to transform opcodes into an expression tree using which we can recover a printable form of algebraic terms. This can be used to generate a new model file that corresponds to each individual agent's or a group of agents' problem.

Once we identify the structure of the problem through model I/O unit, transformations unit is subsequently called and performs transformations of the original model: depending on the option it formulates each individual agent's problem, problems for groups of agents, or perturbed version of those problems. To facilitate transformations, we have basic building blocks such as differentiating opcodes with respect to a variable, constructing the Lagrangian function, and adding proximal terms to an equation. These building blocks are applied to the equations of the original model to build new formulations we want. A list

[1] In general, algebraic modeling languages including GAMS internally generate a compact representation for each equation in the form of opcodes, similar to assembly language, instructing what operations need to be done to evaluate an equation at a given point.

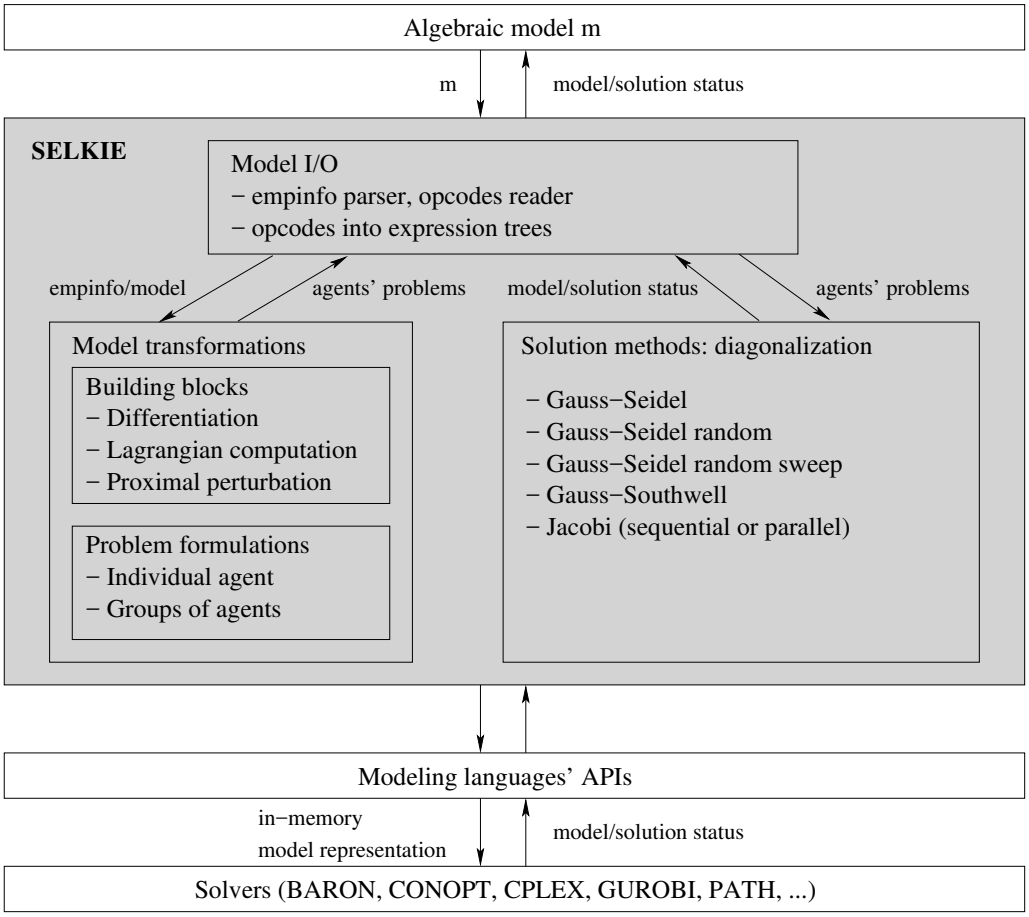


Figure 4.1: Architecture of SELKIE

of agents' problems (SELKIE creates a new agent for a group of agents) is generated as an output of transformations unit. By default, each individual agent's problem specified in the `empinfo` file is created. This can be changed using `agent_group` option as described in Section 4.4.1.

Problems in the form of a list of agents are then passed onto solution methods unit to compute a solution. SELKIE's main solution method is diagonalization which repeatedly solves each agent until convergence. It has five variants, and details are described in

Method	Next agent to solve	Other agents' variables
Gauss-Seidel	cyclic	the most recent values
Gauss-Seidel random	random	the most recent values
Gauss-Seidel random sweep	random-permutations cyclic	the most recent values
Gauss-Southwell	highest residual	the most recent values
Jacobi	cyclic	iteration starting values

Table 4.1: SELKIE's diagonalization methods and their differences

Section 4.3. A problem type is defined for each agent, and we can choose one of the appropriate solvers supported by the underlying modeling languages. For instance, we could use BARON [77], CONOPT [22] or IPOPT [79] and so on in GAMS for agents defining nonlinear programming problems. For a group of agents, the default problem type is MCP. In this case, we solve the group in one-shot by considering changes of all the agents' variable values in the group simultaneously. Other group solve method can be specified via `agent_group` option. For MCPs, PATH [19] is used.

4.3 Solution methods: diagonalization

This section introduces SELKIE's five different variants of diagonalization method. Basically, diagonalization is an iterative method: for a given list of agents it repeatedly solves each agent in the list while fixing other agents' variable values until convergence. Details about each method are described in Section 4.3.2. Section 4.3.3 presents some theoretical results on convergence. We discuss some practical issues on applying these methods such as infeasible and unbounded solutions of sub-problems in Section 4.3.4. We start with convergence criteria that are shared by all of the methods in Section 4.3.1.

4.3.1 Convergence criteria

At each major iteration, SELKIE checks its convergence criteria to see if it finds a solution. It has two criteria: i) the absolute maximum change of variable values of all the agents (in other term the absolute deviation) is less than a tolerance; ii) the residual is less than a tolerance. In general, the second criteria is much more accurate than the first one as it checks if the current point satisfies each agent's KKT conditions. If one of those criteria is satisfied, SELKIE concludes that it has found a solution and terminates the iterations.

The tolerance for the first criterion is determined by `deviation_convergence_tolerance` option. By default, it is set to 10^{-10}. The reason for somewhat such a strict value is that values larger than 10^{-10} sometimes give high residual, say larger than 10^{-6}. With such a large residual, the current point may not be a solution.

To compute the residual, we either evaluate the normal map [67] or Fischer-Burmeister type merit function [36] at the current point. The tolerance for the residual is defined by `convergence_tolerance` option. By default, it is set to 10^{-6} which is same as PATH.

Note that computing a residual is usually much more expensive than computing the absolute deviation. Sometimes this could even slow down the iteration process much. To avoid such a case, SELKIE has options to delay residual computation until deviation reaches a tolerance. For example, we can set options as follows:

```
delay_residual_check=yes
deviation_delay_convergence=1e-8
```

With those options, SELKIE will not compute residual until deviation becomes less than 10^{-8}. By default, delayed residual computation is disabled.

Algorithm 3 Gauss-Seidel method

Input: a list of N agents' problems
Output: a solution if successful or a status otherwise

1: Let $x^0 = (x_1^0, \ldots, x_N^0)$ be an initial point where x_i represents agent i's variable.
2: Let $\texttt{prob}_i(x_i, x_{-i})$ be agent i's problem.
3: **for** $k = 1$ to `iteration limit` **do**
4: **for** $i = 1$ to N **do**
5: Set `status` $\leftarrow$ solve $\texttt{prob}_i(x_1^k, \ldots, x_{i-1}^k, x_i^{k-1}, x_{i+1}^{k-1}, \ldots, x_N^{k-1})$.
6: **if** `status` is successful **then**
7: Set x_i^k to the solution of agent i's problem.
8: **else**
9: **return** an error with `status`
10: **end if**
11: **end for**
12: Set `status` $\leftarrow$ check the convergence criteria.
13: **if** `status` is converged **then**
14: **return** a solution x^k
15: **end if**
16: **end for**
17: **return** an error with status "iteration limit reached"

4.3.2 Diagonalization methods

Basic principle of diagonalization method is to repeatedly solve each agent at a time while fixing other agents' variable values appeared in that agent's problem until convergence. As an example, our Gauss-Seidel method is illustrated in Algorithm 3. At each major iteration, numbered by k, it sequentially solves each agent one at a time while other agents' variable values are kept fixed to the most recent ones. Hence in line 5 of Algorithm 3, agent i uses x_j^k for $j = 1, \ldots, i-1$ (solutions of agents that are solved before it) while it uses x_j^{k-1} for $j = i, \ldots, N$. In general, the use of the most recent values helps achieve a better convergence result than Jacobi method.

Variants of diagonalization method differ by the way they choose the next agent to solve

and how they fix other agents' variable values. Table 4.1 shows the five variants and their differences implemented in SELKIE. Note that Gauss-Southwell chooses the next agent that violates the KKT conditions the most. Compared to other Gauss type methods, Jacobi uses starting values of each major iteration when it needs to fix them so it can be run in parallel. To enable parallelism, we need to set `parallel_jacobi` to `yes`.

We can choose which method to use using `diagonalization_method` option by specifying one of the following:

```
gauss_seidel
gauss_seidel_random
gauss_seidel_random_sweep
gauss_southwell
jacobi
```

4.3.3 Convergence theory

A diagonalization method is more like a computational method without strong convergence properties. Convergence results of Gauss-Seidel or Jacobi method for equilibrium problems is either usually not known or very restrictive [29, see page 198]. For Nash equilibrium problems, when Jacobi method is applied with primal proximal perturbation described in Section 4.4.2, convergence results including existence and uniqueness of a solution [31, see page 481] exist under *P*-matrix property of the Jacobian matrix. For GNEPs, the authors [32] defined a generalized potential game and showed that under appropriate assumptions if there exists an accumulation point generated by Gauss-Seidel method with

primal proximal perturbation, then it is a solution.

4.3.4 Practical issues on applying diagonalization

To apply diagonalization, each sub-problem (agent's problem) needs to be well-defined: a solution is found at each iteration. However, in practice sub-problems could be infeasible or unbounded, which prohibits us from proceeding the iteration. For example, if we run Gauss-Seidel over the pure trade market problem (4.4) in Section 4.4.1, the clearing agent (the VI) that couples all the other agents' variables will have an infeasible solution at the first iteration as demand becomes larger than supply. This occurs because each utility maximizing agent decides its demand subject to the budget constraint, not including the market clearing conditions that are seen only by the clearing agent. In other cases, if the clearing agent chooses to set the price of a good to zero because of an excess supply, then it could result in an unbounded solution of some utility maximizing agent's problem.

The main reason for such difficulties with diagonalization is that information, represented by functions and constraints, is not shared by all agents; only partial information is available to each agent when it makes a decision. We could make it shared by formulating an MCP, in which all the variables are endogenous, but this will restrict us to work on only a relatively large formulation compared to smaller sub-problems. In some favorable case where the feasible region of each agent is represented by the same shared constraint, we can guarantee at least feasibility of each sub-problem as shown in [32]. However, in general it is not easy to have such guarantee.

To overcome these difficulties, SELKIE provides two approaches. The first is grouping of agents as described in Section 4.4.1. It automatically detects independent groups of agents

via Jacobian analysis. Agents that have interactions with each other are grouped together, and there is no interaction between groups. Diagonalization is then applied to those groups, possibly in parallel. By considering all the inter-related agents together, we can avoid the issues mentioned above while achieving a faster computation time. SELKIE also offers an option for users to form their own groups of agents. This could be helpful if they know how to group agents to avoid difficulties.

The second approach is primal and dual proximal perturbation strategies introduced in Section 4.4.2. By perturbing objective and VI functions, called primal proximal perturbation, we can prevent unbounded and infeasible solutions. Here the infeasibility is due to the VI function. Under some assumptions, the sequence generated by the perturbed problems will converge to a solution to the original problem. For infeasible constraints, we expand their feasible region by perturbing those constraints using dual variables, called dual proximal perturbation. We present each strategy and examples showing their effectiveness in Sections 4.4.2-4.4.2, respectively.

4.4 Model transformations

This section introduces SELKIE's two main model transformation features, agent grouping and proximal perturbation, in Sections 4.4.1-4.4.2, respectively. These features can be used to find a more robust and faster solution path compared to the naive approach, either diagonalization over given agents in the `empinfo` file or its MCP formulation. Agent grouping allows us to create groups of agents considering interactions between them and to apply diagonalization to those groups. Proximal perturbations are for avoiding logical

or numerical issues occurring during iterations. They can also be used to expand problem classes amenable to diagonalization by strengthening their convergence properties.

4.4.1 Agent grouping

SELKIE either can automatically detect structurally independent groups of agents or allows users to provide group information via `agent_group` option. We detail each in subsequent sections with examples showing their effectiveness.

Detecting independent groups of agents via Jacobian analysis

Some equilibrium problems can be partitioned into independent groups of agents so that we can find a solution by solving those groups in parallel. By independent groups of agents, we mean that agents belonging to different groups do not affect each other in computing a solution. For example, [25] showed that some pure trade markets with Cobb-Douglas utilities can be decomposed into submarkets each of which defines an independent group of agents. Another example is the friction contact problems of capsules in [50] where independent groups can be formed by grouping capsules that have contacts each other either directly or indirectly through other capsules.

Computationally, independent groups can provide a faster and more robust solution path: i) we can achieve a faster computation time by decomposing the problem into smaller ones and solving them in parallel; ii) we can avoid some numerical or infeasibility issues arising from solving the problem as a whole or decomposing it without considering feasibility, respectively. We demonstrate these two points using the pure trade market example in [25] at the end of this section.

Selkie has two options for detecting and exploiting independent groups. The first option is to let users explicitly specify those groups in its option file. Users describe information about the groups in `agent_group` option. Selkie then reads it, constructs a list of agents each of which represents each group, and solves these agents using one of its solution methods. This will be detailed in Section 4.4.1. The second option is for Selkie to perform an analysis of the nonzero pattern of the Jacobian matrix to automatically identify those groups, where the Jacobian is from the MCP function of the problem.

The nonzero pattern of the Jacobian matrix contains dependency information between agents; by analyzing it we can identify what agents would be affected by the decisions made by some other agent. Proposition 4.1 presents our theoretical background for this. It shows that the pattern can be used to identify independent groups of functions and variables inside a function.

Proposition 4.1. *A function $F(x) : \mathbb{R}^n \to \mathbb{R}^m$ can be partitioned into k independent functions and variables such that $F(x) = P\left(F_{r_1}(x_{c_1}), \ldots, F_{r_k}(x_{c_k})\right)^\top$, where P is a permutation matrix of size m, $x_{c_i} \in \mathbb{R}^{n_i}, F_{r_i} : \mathbb{R}^{n_i} \to \mathbb{R}^{m_i}$ for $i = 1, \ldots, k$ and $\sum_{i=1}^{k} m_i = m, \sum_{i=1}^{k} n_i = n, r_i \cap r_j = c_i \cap c_j = \emptyset$ for each $i \neq j$, if and only if the Jacobian entry $JF(x)_{pq} = 0$ for all x for each pair (p, q) with $p \in r_i$ and $q \in c_j$ where $i \neq j$. Here r_i and c_i are ordered index sets for each $i = 1, \ldots, k$.*

Proof. ($\Rightarrow$) Suppose that F can be partitioned into k independent functions and variables as stated in the proposition. Let (p, q) be given with $p \in r_i$ and $q \in c_j$ where $i \neq j$. Then $JF(x)_{pq} = \left(\partial F_p / \partial x_q\right)(x) = 0$ for all x as $x_q \notin x_{c_i}$ and F_p is a function of x_{c_i} only.

($\Leftarrow$) Assume that we are given a partition pair (R, C) with $R = \{r_1, \ldots, r_k\}$ and $C = \{c_1, \ldots, c_k\}$ for functions and variables, respectively, such that $JF(x)_{pq} = 0$ for all x for

each pair (p, q) with $p \in r_i$ and $q \in c_j$ where $i \neq j$. We show that F_{r_i} is a function of x_{c_i} for each $i = 1, \ldots, k$ by contradiction. Suppose that we have $i \in \{1, \ldots, k\}$ where F_{r_i} is not a function of x_{c_i}. There exists $q, q \notin c_i$ such that x_q contributes to the function value of F_p for some $p \in r_i$. Then we must have $\left(\partial F_p / \partial x_q\right)(x) \neq 0$ for some x, which contradicts our assumption. □

To apply Proposition 4.1 to our case where we want to detect independent sub-MCPs inside an MCP, we need to consider not only the function values but also complementarity relationship between functions and variables. We first define what we mean by independent sub-MCPs.

Definition 4.2. *An* MCP(x, F) *has k independent sub-*MCP*s if there exist a permutation matrix P and a partition* $C = \{c_1, \ldots, c_k\}$ *of variable indices with each* c_i *being an ordered index set such that* $F(x) = P\left(F_{c_1}(x_{c_1}), \ldots, F_{c_k}(x_{c_k})\right)^\top$.

Note that we have used the same index sets for functions and variables in Definition 4.2. We apply a symmetric permutation to the functions and variables. Complementarity relationship is then preserved at a solution so that solving an MCP is equivalent to solving sub-MCPs independently and vice versa as shown in Proposition 4.3.

Proposition 4.3. *Assume that an* MCP(x, F) *with* $F : \mathbb{R}^n \to \mathbb{R}^n$ *has k independent sub-*MCP*s with a partition* $C = \{c_1, \ldots, c_k\}$. *Then* x^* *is a solution to the* MCP(x, F) *if and only if* $x^*_{c_i}$ *is a solution to the* MCP(x_{c_i}, F_{c_i}) *for each* $i = 1, \ldots, k$.

Proof. ($\Rightarrow$) Suppose that x^* is a solution to the MCP(x, F). For $c_i \in C$ and $j \in c_i$, we have $F_j(x^*) \perp x^*_j$. As $F_j(x^*) \equiv F_j(x^*_{c_i})$ and c_i is an ordered set, $x^*_{c_i}$ is a solution to the MCP(x_{c_i}, F_{c_i}).

($\Leftarrow$) Assume that $x^*_{c_i}$ is a solution to the MCP(x_{c_i}, F_{c_i}) for each $i = 1, \ldots, k$. As c_i is an ordered set, the result follows. □

We now prove the relationship between sub-MCPs and the nonzero pattern of the Jacobian matrix.

Proposition 4.4. *An* MCP(x, F) *with* $F : \mathbb{R}^n \to \mathbb{R}^n$ *has* k *independent sub-*MCP*s with a partition* $C = \{c_1, \ldots, c_k\}$ *if and only if* $JF(x)_{pq} = 0$ *for all* x *for each pair* (p, q) *with* $p \in c_i$ *and* $q \in c_j$ *where* $i \neq j$.

Proof. The result follows from Proposition 4.1. □

As an example of using the nonzero pattern analysis, consider the following having two optimization agents:

$$\begin{aligned} &\text{find} \quad (x_1^*, x_2^*) \quad \text{satisfying,} \\ &x_i^* \in \quad \underset{z \geq 0}{\arg\min} \quad \frac{1}{2}(z-1)^2, \quad \text{for } i = 1, 2. \end{aligned} \tag{4.1}$$

It is easy to see that (4.1) has two independent agents; agent 1's decision does not affect agent 2's decision. In this case, we can compute a solution by solving each agent's problem independently. The corresponding MCP($\mathbb{R}^2_+, F(x)$) and the nonzero pattern of the Jacobian matrix JF are then

$$F(x) = \begin{bmatrix} x_1 - 1 \\ x_2 - 1 \end{bmatrix}, \quad \text{nnz}(JF(x)) = \begin{bmatrix} \mathtt{x} & 0 \\ 0 & \mathtt{x} \end{bmatrix}, \tag{4.2}$$

where we have marked `x` if the corresponding entry in JF could be nonzero at some point. By setting $C = \{\{1\}, \{2\}\}$, agents 1 and 2 are independent of each other by Proposition 4.4.

Algorithm 4 Detecting independent sub-MCPs

Input: a nonzero pattern of a square Jacobian matrix J of size n
Output: a set of independent sub-MCPs

1: Construct a bipartite graph $\mathcal{G}(J)$ with the pair (V, E) such that $V = R \cup C, R = (r_1, \ldots, r_n), C = (c_1, \ldots, c_n)$ and $(r_i, c_j) \in E$ if $r_i \in R, c_j \in C$, and J_{ij} could be nonzero, where r_i and c_j represent row i and column j, respectively. Add $(r_j, c_j) \in E$ for each $j = 1, \ldots, n$ if it is not already in E. This is to consider complementarity relationship.
2: Allocate an array `colgid` of size n and initialize its elements with value -1.
3: Set `gid` $\leftarrow 0$.
4: **for** $j = 1$ to n **do**
5: **if** `colgid[j]` not equal to -1 **then**
6: continue
7: **end if**
8: Set `gid` $\leftarrow$ `gid` + 1.
9: $S \leftarrow$ connected component with node c_j.
10: **for** each c_k in S **do**
11: Set `colgid[k]` $\leftarrow$ `gid`.
12: **end for**
13: **end for**
14: **return** `colgid`.

As the pattern also contains dependency of variable values of each agent's problem, it could be further used to decompose an agent into a set of independent smaller sub-agents, which was not specified explicitly in the `empinfo` file. For instance, suppose that we have the following agent:

$$\underset{x:=(x_1,x_2)\geq 0}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{2}(x_i - 1)^2. \tag{4.3}$$

The corresponding MCP and the nonzero pattern of its Jacobian matrix will be the same as in (4.2). Therefore we can identify that the agent actually has two independent sub-agents inside.

We are now left with two issues: i) how to obtain the nonzero pattern of the Jacobian matrix of a given problem; ii) how to identify a partition C in Proposition 4.4 that corresponds

to the independent groups from the nonzero pattern. For i), we formulate an MCP and use modeling languages' APIs for the Jacobian structure identification. For example, GAMS has `gmoGetRowJacInfoOne()` function which provides information about the nonzero Jacobian entries of an equation.

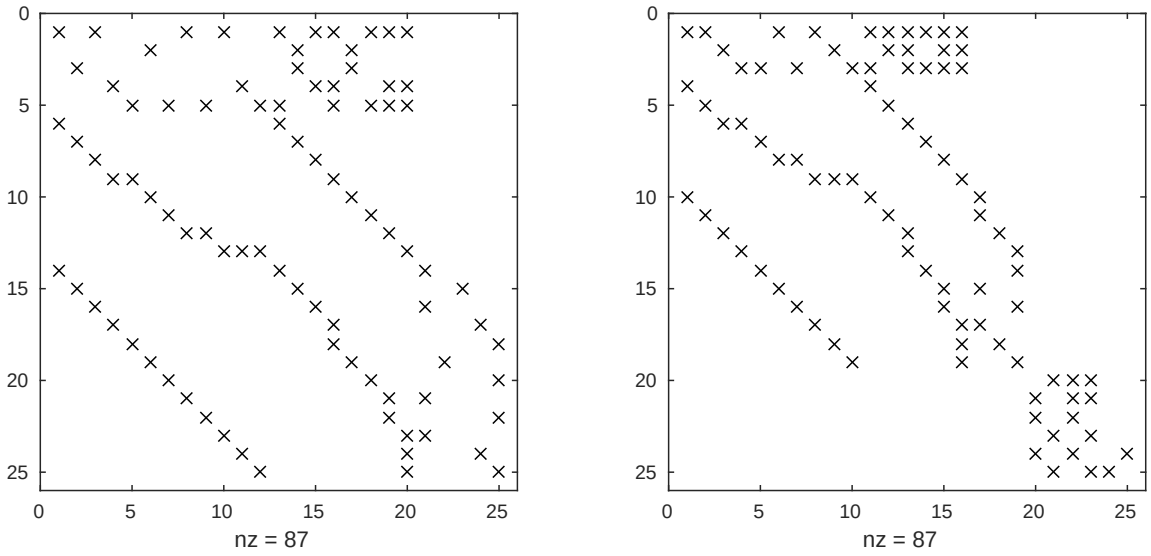
For ii), we construct a bipartite graph similar to [35, 40] that corresponds to the nonzero pattern and search for the connected components of it as described in Algorithm 4. We create two types of nodes for rows and columns of the Jacobian matrix J, denoted by r_i and c_j, respectively. An edge (r_i, c_j) is created if J_{ij} could be nonzero. We also create an edge (r_j, c_j) for $j = 1, \ldots, n$ if it is not already added as we need to consider complementarity. By computing the connected components of the graph, for example using BFS (breadth-first search) algorithm, we can easily identify independent sub-MCPs.

For the rest of this section, we demonstrate benefits of our independent group detection in terms of its efficiency and robustness using the pure trade market example [25]. We briefly introduce the problem first. The pure trade market consists of five agents and eight goods. Each agent goes to the market with its initially endowed goods and sells them all. With the budget at hand obtaining from selling the goods, the agent then tries to buy goods that maximize its utility. Mathematically, the problem can be formulated in the following way:

$$
\begin{aligned}
\text{find} \quad & (x^{1,*}, \ldots, x^{5,*}, \pi^*) \quad \text{satisfying,} \\
x^{t,*} \in \quad & \underset{x^t \geq 0}{\arg\max} \quad \prod_{i=1}^{8} (x_i^t)^{A_i^t}, \\
& \text{subject to} \quad \sum_{i=1}^{8} \pi_i^* x_i^t \leq \sum_{i=1}^{8} \pi_i^* W_i^t, \ \text{for } t = 1, \ldots, 5, \\
\pi^* \in \quad & \text{SOL}(\mathbb{R}_+^8, F(\pi; x^{t,*})), \\
& \text{where} \quad F_i(\pi; x^{t,*}) = \sum_{t=1}^{5} W_i^t - \sum_{t=1}^{5} x_i^{t,*}, \ \text{for } i = 1, \ldots, 8, \\
& \qquad (A_i^t, W_i^t) \text{ is data.}
\end{aligned}
\tag{4.4}
$$

Note that $A_i^t \geq 0$ represents how much agent t desires good i, and $W_i^t \geq 0$ denotes how many agent t possesses good i prior to trade. Zero values of these parameters represent no desire and no possession prior to trade, respectively. Each agent t tries to maximize its utility subject to its budget constraint. The VI agent plays the role of a perfect competitive market. At a solution to the problem, the market is cleared as $F_i(\pi^*; x^{t,*}) \geq 0$ for each $i = 1, \ldots, 8$, that is, supply of a good should be larger than or equal to demand. Also by complementarity, if there is an excess supply, then the corresponding price should be free. Formulation of (4.4) using GAMS/EMP is available on [27]. Note that we take the log of the utility maximizing objective functions and do not include variables whose A_i^t's are zero in the formulation.

We detect independent groups of (4.4) using Selkie. As noted in [25], the model has two submarkets. The first submarket consists of agents $(1, 4, 5)$ with goods $(1, 3, 4, 6, 7, 8)$, and the second one has agents $(2, 3)$ with goods $(2, 5)$. As independent group detection is turned off by default, we turn it on by setting the following in the option file:



(a) Original (b) Permuted

Figure 4.2: Nonzero patterns of the Jacobian matrix of the pure trade market (4.4)

```
detect_independent_groups=yes
```

Figures 4.2(a) and 4.2(b) show a nonzero pattern of the Jacobian matrix of (4.4) and its symmetrically permuted version based on the groups, respectively. Two sub-MCPs were detected that correspond to the two submarkets described before. Furthermore, our independent group detection found that the VI agent is composed of 8 sub-VI agents. This can be easily verified by the fact that F is independent of the price π for a fixed x^t.

Computationally, independent group detection enables us to find a solution in a more robust and faster way. A direct application of diagonalization without grouping will give us an infeasible solution of the VI agent because of an excess demand thus we cannot proceed the iteration. On the other hand, formulating an MCP by grouping the entire agents will help us avoid the infeasible solution, but it takes more iterations, 92 major iterations of

PATH, than solving those two groups independently, 36 and 22 major iterations of PATH, respectively.

Static grouping

This section introduces how we can specify groups of agents along with their solution methods in `agent_group` option and present an example in which we use the group information to obtain improved performance. Note that once the groups are defined, they stay fixed during the solution process, that is, group membership does not change. In this sense, we call it static grouping. We plan to allow dynamic grouping in the future.

We first describe how to specify the group information in `agent_group` option. Assume that we have five agents described in the `empinfo` file. We would like to form two groups, the first group with the first two agents and the second with the rest of them. Each agent in the `empinfo` file is assigned a unique ID of a positive integer starting from 1. IDs follow the order in which agents are specified in the `empinfo` file. Thus a unique number in $\{1, \ldots, 5\}$ is assigned to each agent according to its order in this case.

Using those IDs, the group information can be specified in `agent_group` option as follows:

```
agent_group={{1,2},{3,4,5}} or agent_group={{1,2},{3..5}}
```

We use the set notation to define each group of agents. Thus $\{1, 2\}$ defines the first group consisting of the first two agents, and $\{3, 4, 5\}$ the second group with the rest of them. As a list of groups is another set, we enclose it with braces again. The second format {3..5} is for preventing a tedious listing of group members. In general, $\{a..b\}$ means all the agents with IDs between a and b inclusive. In this case, SELKIE will create two groups, and

diagonalization will be applied to these groups. The order in which the groups are specified determines the execution order of diagonalization on them. For example, if we use the Gauss-Seidel or Jacobi method, group $\{1, 2\}$ is solved first and then the second one at each major iteration. Note that no spaces are allowed in the group definition unless the entire group definition is enclosed by double quotation marks.

The option also allows each group to have a specialized solution method. By default, an MCP is formulated for a group containing more than two agents, and PATH is used to solve it. This default behavior can be changed by defining a solution method after each group's definition. Using the same example above, we can change the second group's solution method to Jacobi by specifying in the following way:

```
agent_group={{1,2},{3,4,5}:jacobi} or agent_group={{1,2},{3..5}:jacobi}
```

SELKIE will now use PATH to solve the first group and Jacobi for the second. It creates two agents corresponding to the groups, and they are solved using Gauss-Seidel. For the second group, it is like SELKIE applies diagonalization recursively to it: each agent in the group is solved in the Jacobi way at each major iteration until convergence. Basically, any of the five variants of diagonalization in addition to the default MCP can be used as a group's solution method. Table 4.2 shows a list of those methods and their abbreviations that can be specified in `agent_group` option.

We conclude this section with a dynamic programming (DP) example in economics that shows a significant performance improvement via our static grouping. For clarity, we omit detailed explanation of the mathematical terms of the problem. Refer to [12] for more details.

The example consists of one fitting agent and 625 Bellman optimization agents. The

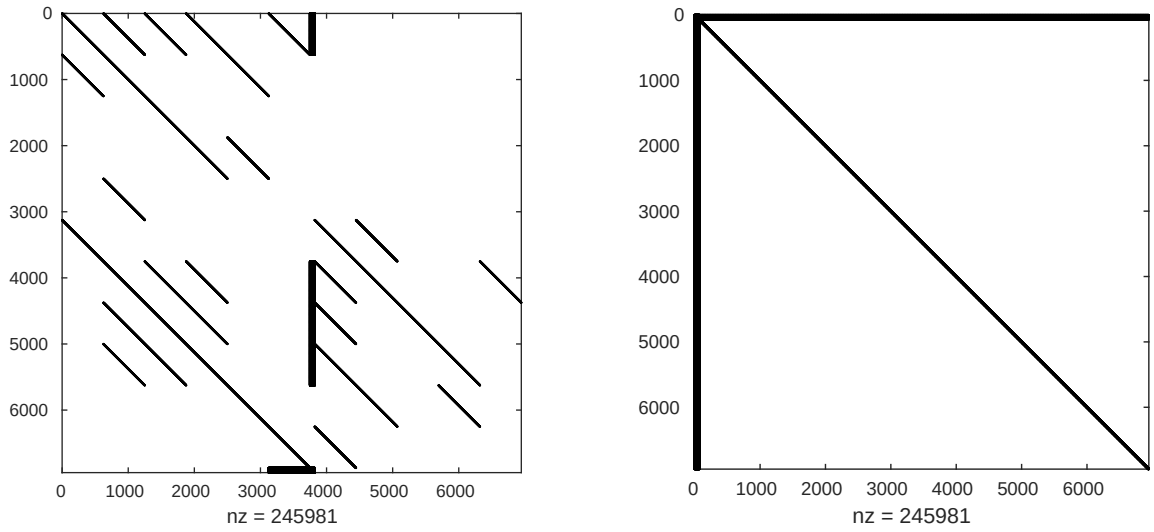
Group solve method name	Abbreviation
Gauss-Seidel	gs
Gauss-Seidel random	gsr
Gauss-Seidel random sweep	gsrs
Gauss-Southwell	gsw
Jacobi	jacobi
MCP	mcp

Table 4.2: Group solve method and its abbreviation for use in `agent_group` option

fitting problem is relatively much easier than Bellman optimization problems, thus a significant time is needed to solve Bellman agents. However, it has a special structure that enables a parallel computation: Bellman agents are independent of each other once fitting parameters are fixed by the fitting agent. In Figure 4.3(b), we permuted rows and columns of the Jacobian matrix by agents. The fitting agent comes first and Bellman agents follow. The arrowhead shape pattern indicates that Bellman agents have interactions only with the fitting agent. In this case, we can achieve parallelism in SELKIE by specifying two groups of agents according to their type and setting a solution method for the Bellman group to use parallel Jacobi as follows:

```
agent_group={{1},{2..626}:jacobi}
parallel_jacobi=yes
```

Computational results show that parallel Jacobi with 625 threads achieved two times faster computation time: it took about 10 minutes whereas 20 minutes were needed if we use sequential version of Jacobi for the Bellman group on Intel Core i5-3340M@2.70 GHz machine having 2 physical cores (logically it has 4 cores, two for each physical core). We expect that we would have achieved a much better computation time if we experimented on



(a) Original (b) Permuted

Figure 4.3: Nonzero patterns of the Jacobian matrix of the DP example

a machine with many more cores.

4.4.2 Proximal perturbations

As discussed in Section 4.3.4, diagonalization could suffer some practical issues such as infeasibility or unboundedness of sub-problems during iterations. To overcome these difficulties, we introduce primal and dual proximal perturbation strategies that perturb the sub-problems in a controlled fashion so that we can find a way to a solution. Primal proximal strategy is for preventing unbounded optimal values or infeasible VI problems, and dual is for making infeasible constraints feasible by expanding their feasible region.

Our proximal strategy is based on the proximal point algorithm for (pseudo) monotone

operators [7, 8, 73]. Suppose that we have the following convex problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x), \\ \text{subject to} \quad & c(x) \geq 0 \end{aligned} \tag{4.5}$$

where $f : \mathbb{R}^n \to \mathbb{R}^1$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are differentiable convex and concave functions, respectively. With appropriate constraint qualifications, the KKT conditions of (4.5) will give us a monotone operator $F(x, y)$ over $y \geq 0$:

$$F(x, y) := \begin{bmatrix} \nabla f(x) - \nabla c(x) y \\ c(x) \end{bmatrix}, \tag{4.6}$$

Note that an operator T defined by $T(x, y) := F(x, y) + N_{\mathbb{R}^n \times \mathbb{R}^m_+}(x, y)$ for $y \geq 0$ and $T(x, y) := \emptyset$ otherwise is a maximal monotone operator [72], and $0 \in T(x^*, y^*)$ implies that x^* is a solution to (4.5) with y^* being the multiplier of $c(x^*)$.

Proximal perturbation strategy perturbs $F(x, y)$ by adding a proximal term as follows:

$$\tilde{F}(x, y; \tilde{x}, \tilde{y}) = \begin{bmatrix} \nabla f(x) - \nabla c(x) y \\ c(x) \end{bmatrix} + \gamma \begin{bmatrix} x - \tilde{x} \\ y - \tilde{y} \end{bmatrix}, \text{ for some } \gamma > 0. \tag{4.7}$$

In the context of equilibrium problems, the operator F constructed by concatenating the KKT conditions and the VI functions of agents could be non-monotone thus theoretical results may not be applicable. However, in practice we still find the strategy useful. Refer to [29, 31, 32, 64] for convergence results of the proximal perturbation strategy in the context of equilibrium problems.

Our strategy allows different parameter values to be used for the primal and dual variables such that $\gamma = (\mu, \lambda)$ where μ and λ are proximal parameters for the primal and dual proximal terms $(x - \tilde{x})$ and $(y - \tilde{y})$, respectively. This is similar to [56] where different regularization parameters are applied to the primal and dual parts. Each strategy is described in detail in Sections 4.4.2-4.4.2, respectively, with examples illustrating their effectiveness.

Selkie allows agent-wise control of these perturbation strategies: i) we can decide whether and which strategy (either primal or dual or primal-dual) to use for an agent; ii) we can have different initial values for the primal and dual proximal parameters for each agent. These are controlled by `proximal_use` option. We start with how to use it in Section 4.4.2. By default, no proximal perturbations are activated.

Enabling proximal perturbation

Proximal perturbation strategies can be enabled by `proximal_use` option. It takes a list of groups[2] of agents where agents in the same group will have the same strategy with the same initial parameter value. By default, the strategy is set to use primal proximal perturbation with perturbation parameter value of 0. Like `agent_group` option, this default behavior can be changed by annotating the strategy and the initial values to use for each group.

For example, suppose that we have five agents. We want to use primal proximal perturbation for the first three agents with initial parameter value of 50 and primal-dual perturbations for the fourth agent with initial values of 50 and 10, respectively. The fifth one has no perturbations. This can be specified as follows:

[2]The group here is to denote the set of agents sharing the same proximal strategies, not the same group as described in Section 4.4.1.

```
proximal_use={{1,2,3}:primal(50),{4}:primaldual(50,10)} or
proximal_use={{1..3}:primal(50),{4}:primaldual(50,10)}
```

For a proximal strategy, one of the following can be specified:

```
primal
dual
primaldual
fixedprimal
fixeddual
fixedprimaldual
```

Note that for the first three strategies proximal parameter values could be changed during iterations. Depending on the sub-problem status, SELKIE chooses to either increase or decrease them in a controlled fashion. For the last three, the parameter values are fixed throughout the iterations, that is, we use $\mu = \mu^k$ and $\lambda = \lambda^k$ for all k.

Primal proximal perturbation

For an optimization problem, we implement primal proximal perturbation strategy by penalizing a large departure from a proximal point, say $\tilde{x}^k$. For example, if f is a minimizing (or a maximizing) objective function, then the strategy perturbs it by adding (or subtracting) a quadratic term:

$$f(x) + \frac{\mu^k}{2}\|x - \tilde{x}^k\|_2^2, \tag{4.8}$$

where $\mu^k > 0$ is a perturbation parameter. Taking a derivative will give us the desired term $(\nabla f(x) + \mu^k(x - \tilde{x}^k))$. The proximal point $\tilde{x}^k$ is usually set to the last solution of the perturbed problem. We used superscript k to emphasize that values could be changed during iterations.

For a convex function f, the strategy guarantees resulting function is strongly convex. The problem will then have a unique solution so that unboundedness cannot occur. This makes each convex sub-problem well-defined, assuming that it is feasible. Theoretical results on convergence and its relationship with methods of multipliers are described in [5, 73].

For a VI function F, we perturb it by adding a proximal term:

$$F(x) + \mu^k(x - \tilde{x}^k). \tag{4.9}$$

As in the case of a convex f, the strategy turns a monotone F into a strongly monotone operator so that we have a unique solution.

We demonstrate the effectiveness of the strategy using the pure trade market example (4.4) in Section 4.4.1. If we run Gauss-Seidel over it, then the VI agent will be infeasible at the first iteration: supply of goods 4, 5, and 8 was less than demand. An intuitive approach in this case is that we increase the price for those goods so that agents would demand less at the next iteration. In other case where there is an excess supply, we decrease the price so that agents would demand more. The approach is similar to the way a perfect competitive market works.

This basic intuition can be implemented using primal proximal perturbation strategy. By perturbing the VI function like (4.9), price would be increased (or decreased) when there is

an excess demand (or an excess supply). Here the VI function F is a constant representing a net supply of each iteration in this case. If it is negative, the case where an excess demand occurs, then the price needs to be larger than the current one $\tilde{x}^k$ so that the perturbed value is nonnegative. Otherwise, it needs to be less than $\tilde{x}^k$. With `proximal_use` option below perturbing the VI agent only, we were able to solve the problem without difficulties.

```
proximal_use={{6}:fixedprimal(50)}
```

Other examples of using primal proximal perturbation can be found on [27].

Dual proximal perturbation

Compared to the primal proximal perturbation strategy where objective and VI functions are perturbed, dual proximal perturbation perturbs constraints so that it can make infeasible constraints feasible. For example, suppose that we have a constraint $c(x) \geq 0$ and $y \geq 0$ is its dual variable. We perturb $c(x)$ as follows:

$$c(x) + \lambda^k(y - \tilde{y}^k) \geq 0, \tag{4.10}$$

where $\lambda^k > 0$ is a perturbation parameter. We can easily see that for some infeasible x, i.e., $c(x) < 0$, we can make it feasible by choosing a large enough $y \geq 0$.

For an optimization problem, we have two different ways of implementing dual perturbation. The first is to perturb the original problem directly by introducing an auxiliary

variable y. For example, we perturb in the following way:

$$\begin{aligned} \underset{y \geq 0, x}{\text{minimize}} \quad & f(x) + \frac{\lambda^k}{2}\|y\|_2^2, \\ \text{subject to} \quad & c(x) + \lambda^k(y - \tilde{y}^k) \geq 0. \end{aligned} \tag{4.11}$$

By checking the KKT conditions, it can be easily verified that variable y implicitly plays the role of a dual variable: it will have the same value with the multiplier at a solution, while allowing us to remedy some infeasible solutions during iterations.

The second option is to convert the optimization problem into its MCP($\mathbb{R}^n \times \mathbb{R}^m_+, F$) formulation and perturb the dual part of F. Hence we will have the following in this case:

$$\tilde{F}(x, y) = \begin{bmatrix} \nabla f(x) - \nabla c(x) y \\ c(x) \end{bmatrix} + \begin{bmatrix} 0 \\ \lambda^k(y - \tilde{y}^k) \end{bmatrix} \tag{4.12}$$

Currently, Selkie supports only the second option. Note that for a VI agent the second option is automatically applied.

We present an example where the dual proximal strategy is effective. Suppose that we are given the following equilibrium problem having two optimization agents:

$$\begin{aligned} \underset{x_1}{\text{minimize}} \quad & \frac{1}{2}x_1^2 - x_1 x_2 - 4x_1, \\ \text{subject to} \quad & x_1 + x_2 = 1, \\ \underset{x_2}{\text{minimize}} \quad & \frac{1}{2}x_2^2 - x_1 x_2 - 3x_2. \end{aligned} \tag{4.13}$$

If we start at $(x_1, x_2) = (0, 0)$, then all variants of diagonalization fail to converge. It generates a cyclic sequence that does not contain a solution. In fact, the associated MCP

function F is non-monotone. This can be verified that $(1,1,-1)J(1,1,-1)^\top = -1$ where J is the Jacobian of F defined below.

$$F(x_1,x_2,y) = \begin{bmatrix} x_1 - x_2 - y - 4 \\ -x_1 + x_2 - 3 \\ x_1 + x_2 - 1 \end{bmatrix}, \quad J = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \tag{4.14}$$

In this case, we recover strong monotonicity using primal-dual proximal perturbation. If we perturb J using primal-dual proximal perturbation, then we will have

$$\begin{aligned} \begin{bmatrix} x_1 & x_2 & y \end{bmatrix} \begin{bmatrix} 1+\mu_1 & -1 & -1 \\ -1 & 1+\mu_2 & 0 \\ 1 & 1 & \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ y \end{bmatrix} &= (1+\mu_1)x_1^2 - 2x_1x_2 + (1+\mu_2)x_2^2 + x_2y + \lambda y^2, \\ &= x_1^2 + x_2^2 + y^2 + (x_1 - x_2)^2 + (x_2 + \frac{1}{2}y)^2 + \frac{3}{4}y^2, \\ &\geq \|(x_1,x_2,y)\|_2^2, \quad \text{for } (\mu_1,\mu_2,\lambda) = (1,2,2). \end{aligned} \tag{4.15}$$

Thus the perturbed Jacobian is strongly monotone. This cannot be achieved if we just applied primal proximal perturbation: if $\lambda = 0$, we can choose some y that makes the result negative. With strong monotonicity, it is guaranteed that we have a unique solution for each proximal point.

With the option below, we found a solution $(x_1^*, x_2^*, y^*) = (-1, 2, -7)$.

```
proximal_use={{1}:fixedprimaldual(1,2),{2}:fixedprimal(2)}
```

4.5 Conclusions

SELKIE is a general-purpose solver for equilibrium problems that exploits block-structure and cascading dependency of the problem. It transforms a given equilibrium problem into a set of structure-exploiting sub-problems via its automatic structure detection mechanism and using user's knowledge. It then applies diagonalization on those sub-problems to find a solution. Sub-problems can be arranged to be solved in a user-defined order so that we can make full use of cascading dependency between them. Parallelism can be achieved on sub-problems when they are block-structured, that is, independently solvable. A sub-solver for each sub-problem of the same type can be chosen so that we can employ a highly efficient solver tailored to the problem. Using proximal perturbations, we can achieve stronger convergence results and avoid unbounded and infeasible solutions of diagonalization. These flexibility and adaptability of SELKIE allows us to find a more robust and faster solution path and to easily extend it to accommodate a new decomposition scheme.

There is potential for future work. For very large-scale problems such as [12], there could be a massive number of sub-problems, and solving them on a personal computer will take a huge amount of time. It may be better to use a grid computing facility such as [78] in this case. We plan to extend SELKIE to be able to run its sub-problems on a computational grid without requiring users to implement APIs for accessing such a facility. This allows users to focus on modeling itself thus to improve their productivity. Such a large problem could also incur a significant amount of model generation time. We plan to study a new construct in modeling languages, possibly combined with the EMP framework, for parallel model generations similar to [15]. We intend to extend SELKIE to take bilevel programming, equilibrium problems with equilibrium constraints, and stochastic programming, all with

consideration of the EMP framework.

Bibliography

[1] Acary, V., M. Brémond, T. Koziara, and F. Pérignon. 2014. FCLIB: a collection of discrete 3D Frictional Contact problems. Technical Report RT-0444, INRIA.

[2] Acary, V., and B. Brogliato. 2008. *Numerical methods for nonsmooth dynamical systems: Applications in mechanics and electronics*, vol. 35 of *Lecture Notes in Applied and Computational Mechanics*. Springer Berlin Heidelberg.

[3] Acary, V., and F. Pérignon. 2007. An introduction to Siconos. Rapport Technique RT-0340, INRIA.

[4] Aguiar, A., B. Narayanan, and R. Mcdougall. 2016. An overview of the GTAP 9 data base. *Journal of Global Economic Analysis* 1(1):181–208.

[5] Bertsekas, D. P., and J. N. Tsitsiklis. 1997. *Parallel and distributed computation: numerical methods*. Belmont, MA: Athena Scientific.

[6] Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* 59(1):65–98.

[7] Billups, S. C. 2000. Improving the robustness of descent-based methods for semismooth equations using proximal perturbations. *Mathematical Programming* 87(1): 153–175.

[8] Billups, S. C., and M. C. Ferris. 1997. Qpcomp: A quadratic programming based solver for mixed complementarity problems. *Mathematical Programming* 76(3): 533–562.

[9] Bixby, R. E. 1992. Implementing the simplex method: The initial basis. *ORSA Journal on Computing* 4(3):267–284.

[10] Britz, W., M. C. Ferris, and A. Kuhn. 2013. Modeling water allocating institutions based on multiple optimization problems with equilibrium constraints. *Environmental Modeling & Software* 46:196–207.

[11] Brook, A., D. Kendrick, and A. Meeraus. 1988. *Gams: A user's guide*. South San Francisco, CA: The Scientific Press.

[12] Cai, Y., K. L. Judd, G. Thain, and S. J. Wright. 2015. Solving dynamic programming problems on a computational grid. *Computational Economics* 45(2):261–284.

[13] Cao, M., and M. C. Ferris. 1995. Lineality removal for copositive-plus normal maps. *Communications on Applied Nonlinear Analysis* 2:1–10.

[14] ———. 1996. A pivotal method for affine variational inequalities. *Mathematics of Operations Research* 21(1):44–64.

[15] Colombo, M., A. Grothey, J. Hogg, K. Woodsend, and J. Gondzio. 2009. A structure-conveying modeling language for mathematical and stochastic programming. *Mathematical Programming Computation* 1(4):223–247.

[16] Cottle, R.W., J.-S. Pang, and R.E. Stone. 2009. *The linear complementarity problem*. Classics in Applied Mathematics 60, Society for Industrial Mathematics.

[17] Dafermos, S. 1983. An iterative scheme for variational inequalities. *Mathematical Programming* 26(1):40–47.

[18] Davis, T. A. 2007. Umfpack. `http://faculty.cse.tamu.edu/davis/suitesparse.html`. [Online; accessed 19-August-2015].

[19] Dirkse, S. P., and M. C. Ferris. 1995. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software* 5(2):123–156.

[20] ———. 1996. A pathsearch damped Newton method for computing general equilibria. *Annals of Operations Research* 68(2):211–232.

[21] ———. 1997. Crash techniques for large-scale complementarity problems. In *Complementarity and variational problems: State of the art*. Philadelphia, PA: SIAM.

[22] Drud, A. S. 1994. CONOPT - a large-scale GRG code. *ORSA Journal on Computing* 6(2):207–216.

[23] Dubois, F., M. Jean, M. Renouf, R. Mozul, A. Martin, and M. Bagneris. 2011. LMGC90. In *10e colloque national en calcul des structures*. Giens, France.

[24] Eaves, B. C. 1976. A short course in solving equations with pl homotopies. *Nonlinear Programming. SIAM-AMS Proceedings* 9:73–143.

[25] ———. 1985. Finite solution of pure trade markets with cobb-douglas. *Mathematical Programming Study* 23:226–239.

[26] Eldersveld, S. K., and M. A. Saunders. 1992. A block-lu update for large-scale linear programming. *SIAM Journal on Matrix Analysis and Applications* 13(1):191–201.

[27] EMP repository. 2017. An EMP framework for equilibrium problems. `http://pages.cs.wisc.edu/~youngdae/emp`. 2017-10-10.

[28] Facchinei, F., A. Fischer, and V. Piccialli. 2007. On generalized Nash games and variational inequalities. *Operations Research Letters* 35(2):159–164.

[29] Facchinei, F., and C. Kanzow. 2010. Generalized Nash equilibrium problems. *Annals of Operations Research* 175(1):177–211.

[30] Facchinei, F., and J.-S. Pang. 2003. *Finite-dimensional variational inequalities and complementarity problems*, vol. I and II. New York: Springer.

[31] ———. 2010. Nash equilibria: the variational approach. In *Convex optimization in signal processing and communications*, ed. Daniel P. Palomar and Yonina C. Eldar. Cambridge: Cambridge University Press.

[32] Facchinei, F., V. Piccialli, and M. Sciandrone. 2011. Decomposition algorithms for generalized potential games. *Computational Optimization and Applications* 50(2): 237–262.

[33] Ferris, M. C., S. P. Dirkse, J.-H. Jagla, and A. Meeraus. 2009. An extended mathematical programming framework. *Computers & Chemical Engineering* 33(12): 1973–1982.

[34] Ferris, M. C., R. Fourer, and D. M. Gay. 1999. Expressing complementarity problems in an algebraic modeling language and communicating them to solvers. *SIAM Journal on Optimization* 9(4):991–1009.

[35] Ferris, M. C., and J. D. Horn. 1998. Partitioning mathematical programs for parallel solution. *Mathematical Programming* 80(1):35–61.

[36] Ferris, M. C., C. Kanzow, and T. S Munson. 1999. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming* 86:475–497.

[37] Ferris, M. C., and T. S. Munson. 2013. Path 4.7. http://www.gams.com/dd/docs/solvers/path. [Online; accessed 19-August-2015].

[38] Fischer, A. 1992. A special newton-type optimization method. *Optimization* 24: 269–284.

[39] Fourer, R., D. M. Gay, and B. W. Kernighan. 2002. *Ampl: A modeling language for mathematical programming*. 2nd ed. Cengage Learning.

[40] Gilbert, J. R., and NG. E. G. 1993. Predicting structure in nonsymmetric sparse matrix factorizations. In *Graph theory and sparse matrix computation*. Springer Berlin.

[41] Guignard, M. 1969. Generalized kuhn-tucker conditions for mathematical programming problems in a banach space. *SIAM Journal on Control* 7(2):232–241.

[42] Harker, P. T. 1984. A variational inequality approach for the determination of oligopolistic market equilibrium. *Mathematical Programming* 30:105–111.

[43] ———. 1988. Multiple equilibrium behaviors on networks. *Transportation Science* 22(1):39–46.

[44] ———. 1991. Generalized Nash games and quasi-variational inequalities. *European Journal of Operational Research* 54:81–94.

[45] Haurie, A., and J. B. Krawczyk. 1997. Optimal charges on river effluent from lumped and distributed sources. *Environmental Modeling and Assessment* 2(3):177–189.

[46] Huber, O., and M. C. Ferris. 2016. Friction Contact Problems from a Variational Inequality perspective. In preparation.

[47] Josephy, N. H. 1979. Newton's method for generalized equations and the pies energy model. Ph.D. thesis, Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI.

[48] Kim, Y., and M. C. Ferris. 2017. SELKIE: a model transformation and distributed solver for structured equilibrium problems. Tech. Rep., University of Wisconsin-Madison, Department of Computer Sciences.

[49] Kim, Y., and M. C. Ferrs. 2017. Solving equilibrium problems using extended mathematical programming. Tech. Rep., University of Wisconsin-Madison, Department of Computer Sciences.

[50] Kim, Y., O. Huber, and M. C. Ferris. 2017. A structure-preserving pivotal method for affine variational inequalities. *Mathematical Programming (online first)*.

[51] Klarbring, A., and J.-S. Pang. 1998. Existence of solutions to discrete semicoercive frictional contact problems. *SIAM Journal on Optimization* 8(2):414–442.

[52] Krawczyk, J. B., and S. Uryasev. 2000. Relaxation algorithms to find Nash equilibria with economic applications. *Environmental Modeling and Assessment* 5(1):63–73.

[53] Lemke, C. E. 1965. Bimatrix equilibrium points and mathematical programming. *Management Science* 11(7):681–689.

[54] Liu, J. 1995. Strong stability in variational inequalities. *SIAM Journal on Control and Optimization* 33(3):725–749.

[55] Luna, Juan Pablo, Claudia Sagastizabal, and Mikhail Solodov. 2014. A class of dantzig-wolfe type decomposition methods for variational inequalty problems. *Mathematical Programming* 143(1):177–209.

[56] Maes, C. M. 2010. A regularized active-set method for sparse convex quadratic programming. Ph.D. thesis, Institute for Computational and Mathematical Engineering, Standford University, Stanford, CA.

[57] Maros, I. 1998. Qp examples. `http://www.doc.ic.ac.uk/~im/00README.QP`.

[58] Mathiesen, L. 1987. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: an example. *Mathematical Programming* 37(1):1–18.

[59] Murphy, F. H., H. D. Sherali, and A. L. Soyster. 1982. A mathematical programming approach for determining oligopolistic market equilibrium. *Mathematical Programming* 24(1):92–106.

[60] Murty, K. G. 1976. *Linear and combinatorial programming*. NY: John Wiley & Sons.

[61] Nisan, N., T. Roughgarden, E. Tardos, and V. V. Vazirani. 2007. *Algorithmic game theory*. New York: Cambridge University Press.

[62] Outrata, J. V., and J. Zowe. 1995. A Newton method for a class of quasi-variational inequalities. *Computational Optimization and Applications* 4(1):5–21.

[63] Pang, J.-S., and D. Chan. 1982. Iterative methods for variational and complementarity problems. *Mathematical Programming* 24(1):284–313.

[64] Pang, J.-S., G. Scutari, F. Facchinei, and C. Wang. 2008. Distributed power allocation with rate constraints in Gaussian parallel interference channels. *IEEE Transactions on Information Theory* 54(8):3471–3489.

[65] Philpott, A., M. C. Ferris, and R. Wets. 2016. Equilibrium, uncertainty and risk in hydro-thermal electricity systems. *Mathematical Programming* 157(2):483–513.

[66] Robinson, S. M. 1979. Generalized equations and their solutions, Part I: Basic theory. *Mathematical Programming Study* 10:128–141.

[67] ———. 1992. Normal maps induced by linear transformations. *Mathematics of Operations Research* 17(3):691–714.

[68] ———. 1994. Newton's method for a class of nonsmooth functions. *Set-Valued Analysis* 2:291–305.

[69] ———. 2013. Equations on monotone graphs. *Mathematical Programming* 141(1): 49–101.

[70] ———. 2015. Convexity in finite-dimensional spaces. Unpublished manuscript.

[71] Rockafellar, R. T. 1970. *Convex analysis*. Princeton, NJ: Princeton University Press.

[72] ———. 1970. On the maximality of sums of nonlinear monotone operators. *Transactions of the Americal Mathematical Society* 149:75–88.

[73] ———. 1976. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization* 14(5):877–898.

[74] Rutherford, T. F. 1995. Extension of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control* 19(8): 1299–1324.

[75] Saunders, M. 2008. Lusol. `http://web.stanford.edu/group/SOL/software/lusol/`. [Online; accessed 19-August-2015].

[76] Schiro, D. A., J.-S. Pang, and U. V. Shanbhag. 2013. On the solution of affine generalized Nash equilibrium problems with shared constraints by Lemke's method. *Mathematical Programming* 142(1):1–46.

[77] Tawarmalani, M., and N. V. Sahinidis. 2005. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming* 103:225–249.

[78] Thain, D., T. Tannenbaum, and M. Livny. 2005. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience* 17(2–4):323–356.

[79] Wachter, A., and L. T. Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming* 106(1):25–57.