Mathematical Programming Computation (2019) 11:457–501
https://doi.org/10.1007/s12532-019-00156-4

FULL LENGTH PAPER

Solving equilibrium problems using extended mathematical programming

Youngdae Kim[1] · **Michael C. Ferris**[2]

Received: 19 January 2018 / Accepted: 26 December 2018 / Published online: 2 March 2019

Abstract
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 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 shared variables and demonstrate their uses for sparse reformulation, economic equilibrium problems sharing economic states, 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 provided. Our framework has been implemented and is available within GAMS/EMP.

Keywords Equilibrium programming · Nash equilibrium problems · Quasi-variational inequalities

Mathematics Subject Classification 90C33 · 90C90 · 65K10 · 65K15

✉ Youngdae Kim
youngdae@anl.gov

Michael C. Ferris
ferris@cs.wisc.edu

[1] Mathematics and Computer Science Division, Argonne National Laboratory, 9700 Cass Avenue, Lemont, IL 60439, USA

[2] Department of Computer Sciences and Wisconsin Institute for Discovery, University of Wisconsin-Madison, 1210 West Dayton St., Madison, WI 53706, USA

1 Introduction

In this paper, 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 [2,4,13]. 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) in their variational forms. All of these problems have been used extensively in the literature [3,9,16,26], but until this work they have not been available directly within modeling systems.

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 said to be a solution to the GNEP if it satisfies

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

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) whose definition is given in Sect. 2, 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

$$\begin{aligned} x_i^* &\in \underset{x_i \in K_i(x_{-i}^*) \subset \mathbb{R}^{n_i}}{\arg\min} \quad f_i(x_i, x_{-i}^*), \quad \text{for} \quad i = 1, \ldots, N, \\ x_{N+1}^* &\in \mathrm{SOL}(K_{N+1}(x_{-(N+1)}^*), G(\cdot, x_{-(N+1)}^*)), \end{aligned} \tag{2}$$

where $\mathrm{SOL}(K, G)$ denotes the solution set of a variational inequality $\mathrm{VI}(K, G)$, assuming that for each given $x_{-(N+1)}$ $K_{N+1}(x_{-(N+1)})$ is a nonempty closed convex set and $G(\cdot, x_{-(N+1)})$ 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 these problems using modeling languages are usually obtained by transforming the problem into their equivalent (under mild assumptions described in Sect. 2) complementarity form, such as a mixed complementarity problem (MCP), and then solving the complementarity problem using a specialized solver, for example PATH [6,12]. The complementarity problem is constructed by concatenating the Karush–Kuhn–Tucker (KKT) conditions of each agent. This implies that users need to compute those conditions by hand and then manually specify the complementarity relationships within modeling languages [11,29]. Similar transformations are needed

to formulate equilibrium problems in their variational forms represented by QVIs as we show in Sect. 2.

This approach has several drawbacks. Computing derivatives by hand to form the MCP is a time-consuming and error-prone procedure. 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 (objective/VI 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.

To resolve these issues, it is desirable to allow equilibrium problems to be specified in their natural algebraic form, such as (1) and (2). However, the notion of agents' ownership of variables and equations of a model makes it challenging to achieve this. For example, agent i owns (or controls) variable x_i and equations f_i and K_i, while other agents' variable x_{-i} appearing in agent i's problem must be treated as a parameter. This is in contrast to traditional optimization or complementarity problems which have a monolithic nature in their ownership, that is, we can think of them as a single agent owning all the variables and equations of the model. As the ownership information is essential in constructing and testing correct optimality conditions of each agent, modeling languages should be able to provide constructs to capture and pass it on to the underlying solvers. Since the existing modeling languages are designed based on the monolithic ownership, no appropriate constructs exist to specify agents' ownership information for the equilibrium problems.

For more intuitive and efficient equilibrium programming, that is, formulating GNEP, MOPEC, or QVI in modeling languages, the paper [10] briefly mentioned that the EMP framework can be used to specify GNEPs and MOPECs. The goal of EMP is to enable users to focus on the problem description itself rather than spending time and checking errors on the derivation of complementarity formulations. Users define variables and equations of a model in the usual way in modeling languages. Complicated structure, such as agent's ownership information, is specified in a separate text file, called the `empinfo` file, by annotating variables and equations of the model in a natural way using the constructs provided by the EMP framework. The modeling language reads that file to identify high level structure of the problem, and the information captured is passed on to the solvers to compute a solution. 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 [10].

In this paper, 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 [9,28]. We introduce 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 economic equilibrium problems sharing the same economic states; (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 variables. 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 paper is organized as follows. In Sect. 2, we define the equilibrium formulations our framework allows. We describe conditions under which the complementarity form is equivalent to the given equilibrium problem and its variational form. Section 3 presents the underlying assumptions of the existing framework and shows how we can model equilibrium problems satisfying these assumptions. In Sects. 4 and 5, we present sophisticated structures that violate the assumptions and introduce our modifications to incorporate them into our framework. Section 4 describes shared constraints and presents a new construct to define the type of solutions, either GNEP equilibria or variational equilibria, associated with them. In Sect. 5, we introduce shared variables and various complementarity formulations for them. Section 6 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 Sects. 3, 4, 5 and 6, we provide examples from the literature that can be neatly formulated using the feature of our framework. Section 7 concludes the paper, pointing out some areas for future extensions.

2 Preliminaries

For given equilibrium problems or their variational forms, the default action of our framework converts them into MCPs and computes a solution to those complementarity problems. In this section, we describe equivalences of the equilibrium problems with quasi-variational inequalities, variational inequalities, and mixed complementarity problems.

We first 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 $K : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ where $K(x)$ is a closed convex (possibly empty) 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

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

One can easily show that x^* is a solution to the (VI) if and only if $0 \in F(x^*) + N_K(x^*)$, where $N_K(\cdot)$ is a normal cone defined over a closed convex set K such that $N_K(x) := \{y \mid \langle y, z - x \rangle \leq 0, \forall z \in K\}$ if $x \in K$ and an empty set otherwise.

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. In this case, $x^* \in B$ 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 put a bound constraint on a function explicitly when the corresponding variable has only one-sided bound and use MCP(x, F) when the feasible region of x is clear from the context.

Throughout this paper, we assume by default that equilibrium problems are of the form (2), 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 (generalized) Nash equilibrium problem. If there are no optimization agents but a single equilibrium agent, then the problem is a variational inequality. All results in this section hold in the case where either type of agent is not present.

The results described below are simple extensions of the existing results found in [16]. We first show the equivalence between the equilibrium problems and their associated QVIs.

Proposition 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 (2) if and only if it is a solution to the QVI(K, F) where*

$$\begin{aligned} K(x) &= \prod_{i=1}^{N+1} K_i(x_{-i}), \\ F(x) &= (\nabla_{x_1} f_i(x_1, x_{-1})^\top, \ldots, \nabla_{x_N} f_N(x_N, x_{-N})^\top, G(x_{N+1}, x_{-(N+1)})^\top)^\top, \end{aligned}$$

with G being the VI function of the equilibrium agent.

Proof ($\Rightarrow$) Let x^* be a solution to (2). 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 agent i has knowledge of a closed convex set X and uses this to define its 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 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 and* $K(x) = \prod_{i=1}^{N+1} K_i(x_{-i})$. *If* x^* *is a solution to the VI*(X, F) *with* F *defined in Proposition 1, then it is a solution to the QVI*(K, F), *thus it is a solution to* (2) *with the same assumptions on* $f_i(\cdot)$ *given in Proposition 1. The converse may not hold.*

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

($\nLeftarrow$) See the example in Sect. 3 of [16]. □

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 using their associated MCP and vice versa. Throughout this section, by a suitable constraint qualification we mean a constraint qualification implying the KKT conditions hold at a local optimal solution, for example the Mangasarian-Fromovitz or the Slater constraint qualification.[1] 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 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 *and* G *given in Proposition 1,* x^* *is a solution to* (2) *if and only if* (x^*, λ^*, μ^*) *is a solution to the MCP*(B, F), *assuming that a suitable constraint qualification holds at* x^* *with*

$$B = \prod_{i=1}^{N+1} [l_i, u_i] \times \mathbb{R}^v \times \mathbb{R}_-^m, \quad v = \sum_{i=1}^{N+1} v_i, \quad m = \sum_{i=1}^{N+1} m_i,$$

$$F(x, \lambda, \mu) = ((\nabla_{x_1} f_1(x) - \nabla_{x_1} h_1(x)\lambda_1 - \nabla_{x_1} g_1(x)\mu_1)^\top, \ldots,$$

[1] Under convex differentiable inequalities and no equalities, both constraint qualifications are equivalent [31].

$$
\begin{aligned}
&(\nabla_{x_N} f_N(x) - \nabla_{x_N} h_N(x)\lambda_N - \nabla_{x_N} g_N(x)\mu_N)^\top,\\
&(G(x) - \nabla_{x_{N+1}} h_{N+1}(x)\lambda_{N+1} - \nabla_{x_{N+1}} g_{N+1}(x)\mu_{N+1})^\top,\\
&h_1(x)^\top, \ldots, h_{N+1}(x)^\top,\\
&g_1(x)^\top, \ldots, g_{N+1}(x)^\top)^\top.
\end{aligned}
$$

Proof ($\Rightarrow$) Let x^* be a solution to (2). From the KKT conditions and constraint qualification at x^*, there exists (λ^*, μ^*) such that

$$
\begin{aligned}
\nabla_{x_i} f_i(x^*) - \nabla_{x_i} h_i(x^*)\lambda_i^* - \nabla_{x_i} g_i(x^*)\mu_i^* \quad &\perp \quad l_i \le x_i^* \le u_i, \quad \text{for} \quad i = 1, \ldots, N,\\
G(x^*) - \nabla_{x_i} h_i(x^*)\lambda_i^* - \nabla_{x_i} g_i(x^*)\mu_i^* \quad &\perp \quad l_i \le x_i^* \le u_i, \quad \text{for} \quad i = N+1,\\
0 = h_i(x^*) \quad &\perp \quad \lambda_i^* \text{ free}, \quad \text{for} \quad i = 1, \ldots, N+1,\\
0 \ge g_i(x^*) \quad &\perp \quad \mu_i^* \le 0, \quad \text{for} \quad i = 1, \ldots, N+1.
\end{aligned}
\tag{3}
$$

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 (3). Since the constraint qualification holds at x^*, we have $N_{K_i(x^*_{-i})}(x_i^*) = \{-\nabla_{x_i} h_i(x^*)\lambda_i - \nabla_{x_i} g_i(x^*)\mu_i \mid 0 = h_i(x^*) \perp \lambda_i, 0 \ge g_i(x^*) \perp \mu_i \le 0\} + N_{[l_i, u_i]}(x_i^*)$ 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 a stationary point to (2) is a solution to the MCP model defined in Proposition 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 4 *For a given QVI(K, F), suppose that $K(x) = \{l \le y \le u \mid h(y, x) = 0, g(y, x) \le 0\}$ where $h : \mathbb{R}^{n \times n} \to \mathbb{R}^v$ and $g : \mathbb{R}^{n \times n} \to \mathbb{R}^m$. Assuming that a suitable constraint qualification holds, 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*

$$
\begin{aligned}
B &= [l, u] \times \mathbb{R}^v \times \mathbb{R}^m_-,\\
\tilde{F}(x, \lambda, \mu) &= \begin{bmatrix} F(x) - \nabla_y h(x, x)\lambda - \nabla_y g(x, x)\mu \\ h(x, x) \\ g(x, x) \end{bmatrix}
\end{aligned}
$$

Proof By applying similar techniques used in the proof of Proposition 3, we get the desired result. □

For a given equilibrium problem or quasi-variational inequality, our framework generates the MCP model defined in Propositions 3 and 4, respectively, and solves it using PATH. If the feasible region is defined by a shared constraint, users can choose

between the VI defined in Proposition 2 and the MCP by specifying the solution type. This will be discussed in Sect. 4. Other extensions are found in Sects. 5 and 6. While this constitutes one method of solution, the ability to define the structured equilibria explicitly opens the door for new solution methods [18].

3 Modeling equilibrium problems using the existing EMP framework

We now describe how to specify equilibrium problems in modeling languages using the EMP framework. While this is implemented in GAMS syntax, the extension to other modeling systems is straightforward. We first present the underlying assumptions on the specification and discuss their limitations in Sect. 3.1. Examples from the literature are given in Sect. 3.2. In Sects. 4 and 5, we relax these assumptions to take more sophisticated structures into account.

3.1 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:

$$\begin{array}{lll} \text{find} & (x_1^*, \ldots, x_N^*) & \text{satisfying,} \\ x_i^* \in & \underset{x_i}{\arg\min} & f_i(x_i, x_{-i}^*), \\ & \text{subject to} & h_i(x_i) = 0, \\ & & g_i(x_i) \leq 0, \ \text{for } i = 1, \ldots, N. \end{array} \tag{4}$$

We need to specify each agent's variables, its objective function, and constraints. Functions and constraints (collectively equations) 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 and equations among the agents using annotations given in an `empinfo` file. For example, we may formulate and solve (4) within GAMS/EMP as follows:

Listing 1 Modeling the NEP

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

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

model nep / deff, defh, defg /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
  put 'min', obj(i), x(i), deff(i), defh(i), defg(i) /;
);
putclose;

solve nep using emp;
```

Let us explain Listing 1. Variable `obj(i)` holds the value of $f_i(x)$, `x(i)` represents variable x_i, and `deff(i)`, `defh(i)`, and `defg(i)` are the closed-form definitions of the objective function $f_i(x)$ and the constraints $h_i(x_i)$ and $g_i(x_i)$, respectively, for $i = 1, \ldots, N$. Equations listed in the model statement and variables in these 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') defh('1') defg('1')
...
min obj('N') x('N') deff('N') defh('1') 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')`, `defh('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 (4), 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 constraint h_i or g_i is defined using other agents' decisions, that is, $h_i(x_i, x_{-i}) = 0$ or $g_i(x_i, x_{-i}) \leq 0$, the equilibrium model written in Listing 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 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 1 is that the current framework does not allow *shared objective functions*, *shared constraints*, and *shared variables*. Sections 4 and 5 give examples of problems that violate Assumption 1 and provide 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:

$$
\begin{aligned}
&\text{find} \quad (x_1^*, \ldots, x_N^*, p^*) \quad \text{satisfying,} \\
&x_i^* \in \quad \underset{x_i}{\arg\min} \quad f_i(x_i, x_{-i}^*), \\
&\qquad \text{subject to} \quad h_i(x_i, x_{-i}^*) = 0, \\
&\qquad\qquad\qquad g_i(x_i, x_{-i}^*) \leq 0, \quad \text{for} \quad i = 1, \ldots, N, \\
&p^* \in \quad \text{SOL}(K(x^*), V(p, x^*)), \\
&\qquad \text{where} \quad K(x^*) := \{p \mid w(p, x^*) \leq 0\}.
\end{aligned}
\tag{5}
$$

Assuming that $p \in \mathbb{R}^r$, we can then formulate (5) within GAMS/EMP in the following way:

Listing 2 Modeling the MOPEC

```
variables obj(i), x(i), p(j);
equations deff(i), defh(i), defg(i), defV(j), defw;

model mopec / deff, defh, defg, defV, defw /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
  put 'min', obj(i), x(i), deff(i), defh(i), defg(i) /;
);
put 'vi defV p defw' /;
putclose empinfo;
```

In addition to optimization agents, we now have an equilibrium agent defined with the `'vi'` keyword in Listing 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., `defV(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 `defV`, its variable is `p`, and `defw` is a constraint. The functions f_i, h_i, and g_i, defined in `deff(i)`, `defh(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 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 construct a VI agent from the KKT conditions of $\max_{y,z} y^2$ subject to $y^2 + z^2 \leq 1$, then the VI function has variable y only, and variable z appears only in the constraint. We could specify `'vi z Fy y cons'` in this case, where $\texttt{Fy}(y, z) \equiv 2y$ and `cons` corresponds to $\{(y, z) \mid y^2 + z^2 \leq 1\}$, and z becomes a preceding variable. Our

interface then automatically creates an artificial function `Fz` defined by $\texttt{Fz}(z, y) \equiv 0$ and matches it with variable `z`.

3.2 Examples

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

3.2.1 NEP

We consider the following oligopolistic market equilibrium problem [14,23]:

$$\begin{aligned}
&\text{find} \quad (q_1^*, \ldots, q_5^*) \quad \text{satisfying,} \\
&q_i^* \in \underset{q_i \geq 0}{\arg\max} \quad q_i\, p\left(\sum_{j=1, j\neq i}^{5} q_j^* + q_i\right) - f_i(q_i), \\
&\qquad \text{where} \quad p(Q) := 5000^{1/1.1}(Q)^{-1/1.1}, \\
&\qquad\qquad f_i(q_i) := c_i q_i + \frac{\beta_i}{\beta_i + 1} K_i^{-1/\beta_i} q_i^{(\beta_i+1)/\beta_i}, \\
&\qquad\qquad (c_i, K_i, \beta_i) \text{ is problem data, for } \quad i = 1, \ldots, 5.
\end{aligned} \tag{6}$$

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 (6) is a NEP.

Listing 3 shows an implementation of (6) within GAMS/EMP.[2] As we see, the `empinfo` file is a natural translation of the algebraic form of (6). Using the same starting value as in [14,23], our GAMS/EMP implementation computed a solution $q^* = (36.933, 41.818, 43.707, 42.659, 39.179)^\top$ that is consistent with the one reported in those papers.

[2] The resulting MCP model can be obtained by either running the GAMS with `keep=1` or storing it in a file named using the option `filename` in an option file `jams.opt`. In the former case, the MCP model is stored in a file `emp.dat` in the scratch directory, e.g., `225a`. In the latter case, refer to http://www.gams.com/latest/docs/S_JAMS.html for more details.

Listing 3 Implementation of the NEP (6) within GAMS/EMP

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

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

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

equations objdef(i);

objdef(i)..
    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)));

model nep / objdef /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
    put 'max', obj(i), q(i), objdef(i) /;
);
putclose empinfo;

q.l(i) = 10;
solve nep using emp;
```

3.2.2 GNEP

We use the following GNEP example derived from the QVI example of [25, p. 14]:

$$\begin{array}{lll} \text{find} & (x_1^*, x_2^*) & \text{satisfying,} \\ x_1^* \in & \underset{0 \le x_1 \le 11}{\arg\min} & x_1^2 + \frac{8}{3} x_1 x_2^* - \frac{100}{3} x_1, \\ & \text{subject to} & x_1 + x_2^* \le 15, \\ x_2^* \in & \underset{0 \le x_2 \le 11}{\arg\min} & x_2^2 + \frac{5}{4} x_1^* x_2 - 22.5 x_2, \\ & \text{subject to} & x_1^* + x_2 \le 20. \end{array} \tag{7}$$

In (7), 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 4 shows an implementation of (7) within GAMS/EMP. Our model has computed a solution $(x_1^*, x_2^*) = (10, 5)$ that is consistent with the one reported in [25]. In Sect. 6.2, we show that (7) can be equivalently formulated as a QVI using our extension to the EMP framework.

Listing 4 Implementation of the GNEP (7) within GAMS/EMP

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

variable obj(i);
positive variable x(i);

equation defobj(i), cons(i);

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

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

x.up(i) = 11;

model gnep / defobj, cons /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
    put 'min', obj(i), x(i), defobj(i), cons(i) /;
);
putclose empinfo;

solve gnep using emp;
```

3.2.3 MOPEC

We present a general equilibrium example in economics [22, Sect. 3] and model it as a MOPEC. While [22] 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 [22]. 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:

$$
\begin{aligned}
&\text{No positive profit for each activity} && -A^\top p^* \geq 0,\\
&\text{No excess demand} && b + Ay^* - x^* \geq 0,\\
&\text{Nonnegativity} && p^* \geq 0,\ y^* \geq 0,\\
&\text{No activity for earning negative profit} && (-A^\top p^*)^\top y^* = 0,\\
&\text{and positive activity implies balanced profit,} && \\
&\text{Zero price for excess supply} && p^{*\top}(b + Ay^* - x^*) = 0,\\
&\text{and market clearance for positive price,} && \\
&\text{Utility maximizing demand} && x^* \in \arg\max_x \ \text{utility}(x),\\
& && \text{subject to}\ \ p^{*\top} x \leq p^{*\top} b.
\end{aligned}
\tag{8}
$$

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 $y^* = 3, x^* = (3, 2, 0)^\top, p^* = (6, 1, 5)^\top$ for $\alpha = 0.9$ that is consistent with the one in [22].[3] Note that in Listing 5 we kept the price of the second commodity fixed to 1 as only relative prices are determined in a general equilibrium: any positive scalar multiplication of equilibrium prices is also equilibrium prices [22, see p. 3].

Listing 5 Implementation of the MOPEC within GAMS/EMP

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

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

variable u    utility of the consumer;
positive variables y         activity of the producer,
                   x(i)      Marshallian demand of the consumer,
                   p(i)      prices;

equations mkt(i)    constraint on excess demand,
          profit    profit of activity,
          udef      Cobb-Douglas utility function,
          budget    budget constraint;

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

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

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

budget..
```

[3] $x_3^* = 0$ as its budget share s_3 is zero.

```
27      sum(i, p(i)*x(i)) =L= sum(i, p(i)*b(i));

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

31  file empinfo / '%emp.info%' /;
32  put empinfo 'equilibrium' /;
33  put 'max', u, 'x', udef, budget /;
34  * We have mkt perp p and profit perp y, the fourth and fifth
        conditions of (6).
35  put 'vi mkt p profit y' /;
36  putclose empinfo;

38  * The second commodity is used as a numeraire.
39  p.fx('2') = 1;
40  x.l(i) = 1;

42  solve mopec using emp;
```

4 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.

4.1 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 1 In equilibrium problems, if the same constraint, given explicitly as an equality or an inequality, appears multiple times in different agents' problem definitions, then it is a shared constraint.

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

Example 1 Find $(x_1^*, \ldots, x_N^*)$ satisfying

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

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 [8,9], where each agent's feasible region is defined by the shared constraint only: in that situation 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 1, 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 [24, Sect. 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 [17,19], 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 can be found in Sect. 4.3.

There are two types of solutions when shared constraints are present. Assume a suitable constraint qualification holds for each solution x^* of Example 1. 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* [9,28]. 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 by Proposition 2, where $X = \{x \in \mathbb{R}^n \mid h(x) \leq 0\}$ and h is a convex function. 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 1, we may model it within GAMS/EMP as follows:

Listing 6 Modeling the GNEP equilibrium via replications

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

model gnep_shared / deff, defg, defh /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
  put 'min', obj(i), x(i), deff(i), defg(i), defh(i) /;
);
putclose empinfo;
```

In Listing 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 specialized solution methods, such as the one in [30], for 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 using the existing EMP framework.

4.2 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 1 as follows:

Assumption 2 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 these agents.
- Each variable is owned by a single agent.

Using Assumption 2, we define shared constraints by placing the same constraint in multiple agents' problems. For example, we can model Example 1 without replications by changing lines 2 and 8–10 of Listing 6 into the following:

Listing 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 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. 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 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 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 8 Computing a variational equilibrium

```
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 1, the following MCP(z, F) is generated:

$$\begin{aligned} F(z) &= ((F_i(z)^\top)_{i=1}^N)^\top, & z &= ((z_i^\top)_{i=1}^N)^\top, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x)\lambda_i - \nabla_{x_i} h(x)\mu_i \\ g_i(x) \\ h(x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \lambda_i \le 0 \\ \mu_i \le 0 \end{bmatrix}, \\ & \text{for} \quad i = 1, \ldots, N. \end{aligned} \tag{9}$$

Note that the same equation $h(\cdot)$ is replicated, and a separate multiplier μ_i is assigned in (9) 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 1:

$$\begin{aligned} F(z) &= ((F_i(z)^\top)_{i=1}^N, F_h(z)^\top)^\top, & z &= ((z_i^\top)_{i=1}^N, z_h^\top)^\top, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x)\lambda_i - \nabla_{x_i} h(x)\mu \\ g_i(x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \lambda_i \le 0 \end{bmatrix}, \\ & \text{for} \quad i = 1, \ldots, N, \\ F_h(z) &= \big[h(x)\big], & z_h &= \big[\mu \le 0\big]. \end{aligned} \tag{10}$$

In (10), 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 2 with the set X defined as $X := \{x \mid h(x) \le 0\}$.

4.3 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.

4.3.1 GNEP with a shared constraint: tragedy of the commons

We consider the tragedy of the commons example [24, Sect. 1.1.2]:

$$\begin{aligned} &\text{find} && (x_1^*, \ldots, x_N^*) && \text{satisfying,} \\ &x_i^* \in && \underset{0 \le x_i \le 1}{\arg\max} && x_i \left(1 - \left(x_i + \sum_{j=1, j\neq i}^{N} x_j^*\right)\right), \\ &&& \text{subject to} && x_i + \sum_{j=1, j\neq i}^{N} x_j^* \le 1. \end{aligned} \tag{11}$$

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, (11) 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 [24], 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 (11) within GAMS/EMP in Listing 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 9 Implementation of the GNEP (11) within GAMS/EMP

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

set i / 1*%N% /;
alias(i,j);

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

equations defobj(i), cap;

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

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

model m / defobj, cap /;

file info / '%emp.info%' /;
put info 'equilibrium' /;
loop(i,
    put 'max', obj(i), x(i), defobj(i), cap /;
);
putclose;
```

```
26 x.up(i) = 1;

28 * Specify SharedEqu option in the jams.opt file to allow
     shared constraints.
29 $echo SharedEqu > jams.opt
30 m.optfile = 1;

32 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 right after line 20 in Listing 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.

4.3.2 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 [17,19] described below:

$$
\begin{array}{lll}
\text{find} & (x_1^*, x_2^*, x_3^*) & \text{satisfying,} \\
x_i^* \in & \underset{x_i \geq 0}{\arg\min} & (c_{1i} + c_{2i}x_i)x_i - \left(d_1 - d_2 \left(\sum_{j=1, j\neq i}^{3} x_j^* + x_i \right) \right) x_i, \\
& \text{subject to} & \sum_{j=1, j\neq i}^{3} \left(u_{jm} e_j x_j^* \right) + u_{im} e_i x_i \leq K_m, \\
& & \text{for} \quad m = 1, 2, \; i = 1, 2, 3, \\
& \text{where} & (c, d, e, u, K) \text{ is problem data.}
\end{array}
\tag{12}
$$

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 $(d_1 - d_2(\sum_{j=1, j\neq i}^{3} x_j^* + x_i))x_i$ is the revenue. Each agent can throw pollutant in the river, but its amount is limited by the two shared constraints in (12).

Listing 10 shows an implementation of (12) 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.[4] Note that a different variational equilibrium (or also called a normalized equilibrium [28]) can be computed by changing the

[4] We used the vector form for the constraints when we declare the equation `cons` for each agent in the `empinfo` file so that we do not have to loop through the set `m`.

scale factors of the objective variables. If we uncomment the code on lines 50-51, then the scale factor $1/i$ for $i = 1, \ldots, 3$ is multiplied to agent i's objective function value. In this case, we obtain a different variational equilibrium $x^* = (26.650, 10.709, 0)$ with multipliers $\mu^*_{\text{cons1}} = -0.531$ and $\mu^*_{\text{cons2}} = 0$.

If we compute a GNEP equilibrium by deleting line 40 in Listing 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:

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

Listing 10 Implementation of (12) within GAMS/EMP

```
1  sets i / 1*3 /
2       m / 1*2 /;
3  alias(i,j);

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

11 table c(m,i)
12         1       2       3
13   1   0.1    0.12   0.15
14   2   0.01   0.05   0.01;

16 table u(i,m)
17         1       2
18   1   6.5     4.583
19   2   5.0     6.250
20   3   5.5     3.750;

22 variables obj(i);
23 positive variables x(i);

25 equations
26     objdef(i)
27     cons(m);

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

32 cons(m)..
33     sum(i, u(i,m)*e(i)*x(i)) =L= K(m);

35 model m_shared / objdef, cons /;

37 file empinfo / '%emp.info%' /;
38 put empinfo 'equilibrium' /;
39 * Comment out the following line to compute a GNEP
       equilibrium.
```

```
40 put 'visol cons' /;
41 loop(i,
42     put 'min', obj(i), x(i), objdef(i), 'cons' /;
43 );
44 putclose empinfo;

46 $echo SharedEqu > jams.opt
47 m_shared.optfile = 1;

49 * Uncomment the code below if we want to compute a
     normalized equilibrium.
50 * obj.scale(i) = ord(i);
51 * m_shared.scaleopt = 1;

53 solve m_shared using emp;

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

Note that since we only have a single constraint `cons` in the modeling system, the lines 51–53 show how to recover a multiplier value for each agent owning the shared constraint.

5 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.[5] For example, state variables controlled by multiple agents in economics, but that need to have the same values across the problem, 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 Sect. 5.1, implicit variables and shared variables are defined. Sect. 5.2 presents various MCP formulations for them. Finally, in Sect. 5.3, we present examples of using shared variables and experimental results comparing various MCP formulations.

5.1 Implicit variables and shared variables

Definition 2 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.

[5] A similar concept was introduced as common decision variables in multi-leader-common-follower games [20]. However, it is the responsibility of solution methods that guarantees the same values at an equilibrium in their setting. In contrast, our definition of shared variables ensures that they will have the same values at all equilibria regardless of solution methods.

Note that Definition 2 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. They also need to be of the same size. Constraints including bounds on variable y can be introduced by explicitly defining them in additional equations. This is for allowing different solution types discussed in Sect. 4 to be associated with them.

Based on Definition 2, we define a shared variable.

Definition 3 In equilibrium problems, variables y_i's are shared variables if there is a set X such that

- The feasible region of agent i is given by

$$K_i(x_{-i}) := \{(y_i, x_i) \in \mathbb{R}^{n_y \times n_i} \mid (y_i, x_i) \in X_i(x_{-i}), (y_i, x_i, x_{-i}) \in X\}, \text{ for } i=1, \ldots, N. \quad (13)$$

- 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's share their values. An extension to the case where they are shared by some subset of agents is straightforward.

An equilibrium where shared variables y_i's are present is defined as follows:

$$\begin{array}{lll} \text{find} & (y^*, x_1^*, \ldots, x_N^*, x_{N+1}^*) & \text{satisfying,} \\ (y^*, x_i^*) & \in \underset{(y,x_i) \in K_i(x_{-i}^*)}{\arg\min} & f_i(y, x_i, x_{-i}^*), \quad \text{for} \quad i = 1, \ldots, N, \\ x_{N+1}^* & \in \mathrm{SOL}(K_{N+1}(x_{-(N+1)}^*), F(\cdot, x_{-(N+1)}^*)). & \end{array} \quad (14)$$

Example 2 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 The variable y is a shared variable of the following equilibrium problem:

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

Listing 11 presents GAMS code to model Example 2. 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 11 Modeling a shared variable

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

model shared_implicit / deff, defH /;

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'implicit y defH' /;
loop(i,
 put 'min', obj(i), x(i), y, deff(i) /;
);
putclose empinfo;
```

Bounds on the shared variables can be introduced by explicitly defining them in additional equations. These bounds could change feasible region hence solutions of the problem. For example, the following two-agent problem, each of which minimizes its total cost, has bounds on the shared variable y, and its solution changes as the value of the upper bound b varies. An implementation of (15) is available at [7]. Our framework computes $(x_1^*, x_2^*) = (b/2, b/2)$ for $b \leq 12$ and $(x_1^*, x_2^*) = (6, 6)$ for $b > 12$.

$$\begin{array}{lll} \text{find} & (y^*, x_1^*, x_2^*) & \text{satisfying,} \\ (y^*, x_i^*) \in & \underset{x_i \geq 0, y}{\arg\min} & x_i - x_i(10 - 0.5 * y), \\ & \text{subject to} & y = x_i + x_{-i}^*, \\ & & 0 \leq y \leq b, \ \text{for } i = 1, 2. \end{array} \tag{15}$$

As we now allow shared variables, Assumption 2 needs to be modified as follows:

Assumption 3 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 these 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 these 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.

Note that in Assumption 3 we allow missing ownership for an implicit variable as its value is well-defined via its defining constraint once the values of other variables are set. When the ownership is not specified for an implicit variable, our framework creates a VI agent that owns the variable and its defining constraint: H becomes a VI function, and y is its matching variable in Example 2. This turns out to be especially useful to model mixed behavior as described in Sect. 5.3.3.

5.2 Various MCP formulations for shared variables

This section describes various MCP formulations for shared variables. For clarity, we will use Example 2 to demonstrate our formulations throughout this section. Each formulation in Sects. 5.2.1, 5.2.2 and 5.2.3 shares the same GAMS code of Listing 11. Different formulations can be obtained by specifying an appropriate value for the option `ImplVarModel` in the file `jams.opt`. In Sect. 5.3, we present experimental results comparing the sizes and performance of these formulations.

5.2.1 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, 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 these KKT conditions:

$$F(z) = \left[(F_i(z)^\top)_{i=1}^N\right]^\top, \qquad z = \left[(z_i^\top)_{i=1}^N\right]^\top,$$
$$F_i(z) = \begin{bmatrix} \nabla_{x_i} f_i(x,y) - (\nabla_{x_i} H(y,x))\mu_i \\ \nabla_{y_i} f_i(x,y) - (\nabla_{y_i} H(y,x))\mu_i \\ H(y_i, x) \end{bmatrix}, \quad z_i = \begin{bmatrix} x_i \\ y_i \\ \mu_i \end{bmatrix}. \tag{16}$$

The size of (16) 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. Table 1 summarizes the sizes of the MCP formulations depending on the strategy. (16) can be obtained by specifying an option `ImplVarModel=Replication` in `jams.opt`.

5.2.2 Switching shared variables with multipliers

We introduce a switching strategy that does not require replications. 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.[6] For example, if an MCP is given by

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

[6] Note that this could affect the behavior of the underlying solution methods that are sensitive to the row or column order of the given problem.

Table 1 The size of the MCPs containing shared variables of Example 2

Strategy	Size of the MCP
Replication	$(n + 2mN)$
Switching	$(n + mN + m)$
Substitution (implicit)	$(n + nm + m)$
Substitution (explicit)	$(n + m)$

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 defining constraints. Thus we can reduce the size of the resultant MCP.

The MCP(z, F) obtained by applying the switching technique to Example 2 is as follows:

$$\begin{aligned} F(z) &= \left[(F_i(z)^\top)_{i=1}^N, F_h(z)^\top\right]^\top, & z &= \left[(z_i^\top)_{i=1}^N, z_h^\top\right]^\top, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x, y) - (\nabla_{x_i} H(y, x))\mu_i \\ \nabla_y f_i(x, y) - (\nabla_y H(y, x))\mu_i \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \mu_i \end{bmatrix}, \\ F_h(z) &= \left[H(y, x)\right], & z_h &= \left[y\right]. \end{aligned} \tag{17}$$

The size of (17) 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 currently the default value for `ImplVarModel`.

5.2.3 Substituting out multipliers

We can apply our last strategy when 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 unique continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}^m$ that maps into some neighborhood of $\bar{y}$ such that $H(h(x), x) = 0$ for all x in some neighborhood of $\bar{x}$.

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 [13, 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^*) of 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 can be represented as follows:

$$\begin{array}{lcll} 0 = \nabla_{x_i} f_i - \nabla_{x_i} H (\nabla_y H)^{-1} \nabla_y f_i & \perp & x_i & \text{free}, \quad \text{for} \quad i = 1, \ldots, N, \\ 0 = H(y, x) & \perp & y & \text{free}, \end{array} \tag{18}$$

where we also applied the switching technique in Sect. 5.2.2.

We can derive the same formulation (18) 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

$$\begin{aligned} 0 &= \nabla_y f_i(x_i^*, x_{-i}^*, y^*) - (\nabla_y H(y^*, x^*)) \mu_i^* \Longrightarrow \mu_i^* \\ &= (\nabla_y H(y^*, x^*))^{-1} \nabla_y f_i(x_i^*, x_{-i}^*, y^*). \end{aligned} \tag{19}$$

We can then substitute out every occurrence of μ_i by the right-hand side of (19) and remove the left-hand side from consideration. The result is the formulation (18).

A critical issue with applying the formulation (18) 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 could 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 (18) there exists Λ_i^* satisfying the following and vice versa:

$$\begin{array}{lcll} 0 = \nabla_{x_i} f_i - \Lambda_i \nabla_y f_i & \perp & x_i & \text{free}, \\ 0 = \Lambda_i \nabla_y H - \nabla_{x_i} H & \perp & \Lambda_i & \text{free}, \quad \text{for} \quad i = 1, \ldots, N \\ 0 = H(y, x) & \perp & y & \text{free}. \end{array} \tag{20}$$

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

$$\begin{aligned} F(z) &= \left[(F_i(z)^\top)_{i=1}^N, F_h(z)^\top\right]^\top, & z &= \left[(z_i^\top)_{i=1}^N, z_h^\top\right]^\top, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x,y) - \Lambda_i \nabla_y f_i(x,y) \\ \Lambda_i \nabla_y H(y,x) - \nabla_{x_i} H(y,x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \Lambda_i \end{bmatrix}, \\ F_h(z) &= \left[H(y,x)\right], & z_h &= \left[y\right]. \end{aligned} \tag{21}$$

The size of (21) 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. As we know the explicit algebraic formulation of $\nabla_{x_i} H$, the following MCP is formulated:

$$\begin{aligned} F(z) &= \left[(F_i(z)^\top)_{i=1}^N, F_h(z)^\top\right]^\top, & z &= \left[(z_i^\top)_{i=1}^N, z_h^\top\right]^\top, \\ F_i(z) &= \left[\nabla_{x_i} f_i(x,y) - \nabla_{x_i} H(y,x) \nabla_y f_i(x,y)\right], & z_i &= \left[x_i\right], \\ F_h(z) &= \left[H(y,x)\right], & z_h &= \left[y\right]. \end{aligned} \tag{22}$$

Note that the size of (22) 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, (21) is formulated. The formulation can be obtained by specifying an option `ImplVarModel=Substitution` in `jams.opt`.

5.3 Examples

In this section, we introduce three models that use shared variables. Section 5.3.1 describes an example where we can reduce its density significantly by introducing a shared variable. This enables the problem, previously known as computationally intractable, to be efficiently solved. Section 5.3.2 presents an economic equilibrium 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 agents. We can then use the constructs for shared variables to define the state variables. In Sect. 5.3.3, we present an example of modeling mixed pricing behavior of agents. More examples on using shared variables, for example modeling shared objective functions, can be found at [7]. 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 [5] as its basis computation engine.

5.3.1 Improving sparsity using a shared variable

We consider an oligopolistic energy market equilibrium example [21, Sect. 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 [21], solvable. The example is defined as follows:

$$
\begin{aligned}
&\text{find} && (q_0^*, q_1^*, \ldots, q_5^*) && \text{satisfying,}\\
&q_0^* \in && \underset{0\le q_0\le U_0}{\arg\max} && p\left(\sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^*\right)\left(\sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^*\right) - \sum_{i=1}^{5} c_i(q_i^*) - Pq_0,\\
&&& \text{subject to} && q_0 + \sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}^* = d,\\
&q_i^* \in && \underset{0\le q_i\le U_i}{\arg\max} && p\left(\sum_{j=1, j\neq i}^{5}\sum_{k=1}^{n_j} q_{jk}^* + \sum_{k=1}^{n_i} q_{ik}\right)\left(\sum_{k=1}^{n_i} q_{ik}\right) - c_i(q_i),\\
&&& \text{subject to} && q_0^* + \sum_{j=1, j\neq i}^{5}\sum_{k=1}^{n_j} q_{jk}^* + \sum_{k=1}^{n_i} q_{ik} = d,\\
&&& \text{where} && c_i(q_i) = \frac{1}{2} q_i^\top M_i q_i + b_i^\top q_i,\\
&&&&& p(Q) := \left(\frac{-P}{(1.5d)^2} Q^2 + P\right),\\
&&&&& (P, d, M_i, b_i, U_i, n_i) \text{ is problem data,} \quad \text{for } i = 1, \ldots, 5.
\end{aligned}
\tag{23}
$$

Let us briefly describe (23). 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 [21], 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 $n = 2500$, and it was not able to solve problems of sizes $n = 5000$ 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 $z := \sum_{i=1}^{5}\sum_{k=1}^{n_i} q_{ik}$. Mathematically, the problem is defined as follows:

$$
\begin{array}{lll}
\text{find} & (z^*, q_0^*, q_1^*, \ldots, q_5^*) & \text{satisfying,} \\
q_0^* \in & \underset{0\le q_0\le U_0}{\arg\max} & p(z^*)z^* - \displaystyle\sum_{i=1}^{5} c_i(q_i^*) - Pq_0, \\
& \text{subject to} & q_0 + z^* = d, \\
(z^*, q_i^*) \in & \underset{z, 0\le q_i\le U_i}{\arg\max} & p(z)\left(\displaystyle\sum_{k=1}^{n_i} q_{ik}\right) - c_i(q_i), \\
& \text{subject to} & q_0^* + z = d, \\
& & z = \displaystyle\sum_{j=1, j\neq i}^{5}\sum_{k=1}^{n_j} q_{jk}^* + \sum_{k=1}^{n_i} q_{ik}.
\end{array}
\tag{24}
$$

Listing 12 implements (24). We 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 12 Implementation of (24) 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);
z.l = sum((i,k), q.l(i,k));

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

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

demand..
     q0 + z =E= d;

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

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

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'implicit z defz' /;
put 'visol demand' /;
put 'min', iso_obj, q0, iso_defobj, demand /;
loop(i,
     put 'min', agent_obj(i);
     loop(k, put q(i,k););
     put z, agent_defobj(i), demand /;
);
putclose empinfo;

$echo SharedEqu > jams.opt
m_oligop.optfile = 1;

solve m_oligop using emp;
```

Table 2 describes the statistics and performance of (23) over various sizes of plants and agents.[7] The '-' symbol represents that we were not able to obtain the results because of memory issue. In Tables 2(a) and 2(b), we used the same setup as in [21]. 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 similar issue: its high density generated computationally intractable models for $n = 25{,}000$ and $50{,}000$. This was due to the

[7] The replication strategy is not allowed in this case as it is ambiguous what variable to replicate for the ISO agent: the agent uses shared variable z, but it does not own it. Our solver automatically detects this case and generates an error.

Table 2 Model statistics and performance comparison of (23) using PATH

n	Original		Switching		Substitution	
	Size	Density (%)	Size	Density (%)	Size	Density (%)
(a) MCP model statistics when we have 1 ISO agent and 5 energy-producing agents						
2500	2502	99.92	2508	0.20	2503	20.07
5000	5002	99.96	5008	0.10	5003	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	–	–

n	Original		Switching		Substitution	
	(Major, Minor)	Time (s)	(Major, Minor)	Time (s)	(Major, Minor)	Time (s)
(b) Performance comparison when we have 1 ISO agent and 5 energy-producing agents						
2500	(2, 2639)	57.78	(1, 2630)	1.30	(1, 2630)	13.18
5000	(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	–	–

n	Switching		Substitution	
	Size	Density (%)	Size	Density (%)
(c) MCP model statistics when we have 1 ISO agent and $n/2$ energy-producing agents				
2500	3753	0.12	2503	0.20
5000	7503	0.06	5003	0.10

Table 2 continued

n	Switching		Substitution	
	Size	Density (%)	Size	Density (%)
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

n	Switching		Substitution	
	(Major, Minor)	Time (s)	(Major, Minor)	Time (s)
(d) Performance comparison when we have 1 ISO agent and $n/2$ energy-producing agents				
2500	(1, 2650)	1.43	(1, 2650)	0.88
5000	(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

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(c) and 2(d) report 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 1). This made the strategy two times slower than the substitution strategy.

5.3.2 Modeling equilibrium problems with equilibrium constraints

We construct an economic equilibrium model[8] where data was taken from the GTAP (Global Trade Analysis Project) 9 database [1]. 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:

$$\begin{array}{lll} \text{find} & (w^*, z^*, t^*) & \text{satisfying,} \\ (w^*, z^*, t_i^*) \in & \underset{w,z,t_i \in T_i}{\arg\max} & w_i, \\ & \text{subject to} & H(w, z, t) = 0, \\ & & \text{for} \quad i = 1, \ldots, 23, \end{array} \tag{25}$$

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 determines 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. 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 3, 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

[8] The original model was written by Thomas Rutherford, and was solved by applying the diagonalization method (Gauss-Seidel) to the nonlinear problem (25) 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.

Table 3 MCP model statistics and performance comparison of the economic equilibrium model (25)

# Agents	Replication		Switching		Substitution	
	Size	Density (%)	Size	Density (%)	Size	Density (%)
5	570	1.66	350	3.34	1230	0.77
10	2290	0.72	1300	1.70	10,210	0.14
15	5160	0.50	2850	1.28	35,190	0.06
20	9180	0.40	5000	1.10	84,420	0.03
23	12,144	0.37	6578	1.03	129,030	0.02

# Agents	Replication		Switching		Substitution	
	(Major, Minor)	Time (s)	(Major, Minor)	Time (s)	(Major, Minor)	Time (s)
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

the MCP models. We do not describe the implementation within GAMS/EMP as the number of lines is long. Refer to [7] for the implementation.

In all settings, the switching strategy was of the smallest MCP size as it did not replicate or create any variables and equations. 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[9] than the substitution strategy. Although it performed more number of iterations on the problem having 10 agents, its time was still faster than 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 1. Although the density of it was the smallest, the number of nonzero entries was the largest. Hence linear algebra computation became much slower.

5.3.3 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 Sect. 3.2.1. 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.

[9] 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.

Listing 13 implements our mixed behavior model. We introduce an implicit variable z that represents the price $p(Q)$ defined in (6). 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 32 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 13 Implementation of mixed behavior of agents within GAMS/EMP

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

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

8  variables obj(i), z;
9  positive variables q(i);

11 equations defobj(i), defz;

13 defobj(i)..
14     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)));

16 defz..
17     z =e= 5000**(1.0/1.1)*sum(i, q(i))**(-1.0/1.1);

19 model mixed / defobj, defz /;

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

29 q.l(i) = 10;
30 z.l = sum(i, q.l(i));

32 solve mixed using emp;

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

36 objval(i,'competitive') = obj.l(i);
37 qval(i,'competitive') = q.l(i);
38 pval('competitive') = z.l;
39 totalobjval('competitive') = sum(i, objval(i,'competitive'
       ));
40 socialwelfare('competitive') = (5000**(1.0/1.1)*11*sum(i,
       q.l(i))**(0.1/1.1)
41       - z.l*sum(i, q.l(i))) + totalobjval('competitive');
```

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

loop(kind,
     put empinfo 'equilibrium' /;
     put 'implicit', z, defz /;
     loop(i,
          if (i.val le ord(kind),
               put 'max ', obj(i), q(i), z, defobj(i) /;
          else
               put 'max ', obj(i), q(i), defobj(i) /;
          );
     );
     putclose empinfo;

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

     solve mixed using emp;

     objval(i,kind) = obj.l(i);
     qval(i,kind) = q.l(i);
     pval(kind) = z.l;
     totalobjval(kind) = sum(i, objval(i,kind));
     socialwelfare(kind) = (5000**(1.0/1.1)*11*sum(i, q.l(i
         ))**(0.1/1.1)
          - z.l*sum(i, q.l(i))) + totalobjval(kind);
);
```

Table 4 presents firms' profits 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 index 1, 2, or 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. A similar observation was made in [15] and was explained as an externality effect.

6 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.

Table 4 Profits of the firms and social welfare of various mixed models of Listing 13

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	39,063.824	39,050.191	39,034.577	39,022.469	39,016.373	39,015.125

6.1 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 15 shows a 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 14 Modeling the QVI

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

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

model qvi / defF, defh, defg /;

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

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 Sect. 3.1 except that additional variables could follow right after each function-variable pair. In this case, these 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, however, in this case an explicit symbol `0` should be specified to represent a zero function to avoid ambiguity.) Therefore, in Listing 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 the above case, the feasible region is a fixed set, $K(x) := K$, specified by `defh` and `defg`.

6.2 Example

We consider the following QVI(K, F) example in [25, p. 14]:

$$F(y) = \begin{bmatrix} -\frac{100}{3} + 2y_1 + \frac{8}{3}y_2 \\ -22.5 + \frac{5}{4}y_1 + 2y_2 \end{bmatrix}, \qquad (26)$$
$$K(x) = \{0 \le y \le 11 \mid y_1 + x_2 \le 15, x_1 + y_2 \le 20\}$$

Listing 15 describes an implementation of (26). As in (26), 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 [25].

Listing 15 Implementation of (26) within GAMS/EMP

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

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

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

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

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

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

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

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

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 (26) is equivalent to the GNEP (7) in Sect. 3.2.2 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.

However, the information provided to our framework could be different depending on the formulations. The GNEP formulation (7) gives us each agent's information: 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 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.

7 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 by hand. 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 economic equilibrium problems sharing states, price-taking and price-making agents, shared objective functions, and so on. 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 to be defined using nonsmooth equations [27]. We plan to extend our framework to incorporate equilibrium problems including agents solving stochastic programs, bilevel programming, equilibrium problems with equilibrium constraints, all with con-

sideration of shared constraints and shared variables, and to implement the EMP in other modeling systems such as AMPL and Julia.

Acknowledgements This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under contract number DE-AC02-06CH11357.

Appendix

In this Appendix, we present a brief description of our source code and introduce our recently developed alternative EMP interface that allows set notation to be specified in the `empinfo` file. The former may be useful for interested readers to understand the internal mechanism of our framework, and the latter may provide a concise and compact way of describing agent information.

Internal mechanism of the EMP equilibrium framework

Basically, our framework performs a model transformation of an EMP model into an MCP model. Key components to implement the transformation are (i) a dictionary, (ii) an EMP parser, and (iii) expression graphs and their derivative computation routines.

The dictionary contains mapping information between variables and equations of a given model and its flattened (or scalar) version. Internally, numeric indices are used to refer to variables and equations, and vector notation (surrounding the name in single quotes) is flattened. In Table 5, we present an example of the mapping information of Listing 5. For clarity, we attached characters `x` and `e` for variables and equations. This information can be obtained by specifying `Dict` option in the option file `jams.opt`.

The EMP parser reads the `empinfo` file and builds a table of agent ownership information of variables and equations using the numeric indices by referring to the dictionary. The table consists of two arrays of sizes n and m, where they correspond to the sizes of the variables and equations of the flattened model, respectively. Whenever the parser encounters the name of a variable or an equation, it identifies its numeric

Table 5 Mapping of variables and equations between the original model and its flattened model

Original variable and equation name	Flattened index
`u`	`x1`
`y`	`x2`
`x('1'), x('2'), x('3')`	`x3, x4, x5`
`p('1'), p('2'), p('3')`	`x6, x7, x8`
`mkt('1'), mkt('2'), mkt('3')`	`e1, e2, e3`
`profit`	`e4`
`udef`	`e5`
`budget`	`e6`

Fig. 1 An example of the agent ownership table

x1	x2	x3	x4	x5	x6	x7	x8
1	2	1	1	1	2	2	2

e1	e2	e3	e4	e5	e6
2	2	2	2	1	1

index using the dictionary and marks the ID of the agent owning it in the corresponding array. Figure 1 shows an example of the agent ownership table of Listing 5. It is assumed that agent ID starts with 1, and its order follows the order of the agents specified in the `empinfo` file. For shared constraints and shared variables, special agent IDs are stored using which we can identify a list of the agents sharing them.

Finally, using the table, the corresponding MCP function is constructed by computing the KKT conditions of each agent's problem. This is performed using expression graphs and their derivative computation routines.

Expression graphs represent nonlinear equations of a model. They take the form of a sequence of operations. For example, in GAMS, $(x + y)^3$ is internally represented by a sequence of `PushV 1`, `AddV 2`, `PushI 3.0`, and `CallArg2 21`, where x and y are assumed to be mapped to numeric indices 1 and 2, respectively. Operands are pushed onto the stack first via `PushV` or `PushI` before an operation is performed on them, and its result (if there is any) is pushed back onto the stack. Hence, in this case, operand x is pushed first, and an addition, which also takes its second operand y, is performed on them. The result of the addition and a constant 3.0 are pushed onto the stack, and operation 21, which corresponds to the power, is applied to them. If we take a derivative of $(x + y)^3$ with respect to x, then a new sequence of `PushV 1`, `AddV 2`, `CallArg1 9`, and `MulI 3.0` is generated, where operation 9 represents a square $(\cdot)^2$ operation. One can verify that the sequence represents $3(x + y)^2$. Two different expression graphs can be easily added (or subtracted) to form an addition (or a subtraction) of nonlinear equations. Linear equations are handled directly by storing their coefficients, and their derivative computations are straightforward. In this way, we can construct the MCP functions described in this paper.

An alternative interface allowing set notation in the `empinfo` file

So far, we have instantiated sets or used vector notation when we specify agent information in the `empinfo` file. For example, on lines 20–21 in Listing 3 each element of set `i` was instantiated by using the `loop` and `put` statements, and on lines 33 and 35 in Listing 5 vector notation was used to represent the entire elements of `x`, `mkt`, and `p`. The main reason we specify in this way is that the current implementation does not allow a direct specification of sets in the `empinfo` file.

This could potentially prevent us from taking advantage of using a set notation, such as a compact representation of a set of variables and equations sharing similar structure. For example, in Listing 12 we had to take an additional loop over set `k` to slice variable `q(i,k)` among the agents. In the cases where there are a large number of such variables or equations, they may involve many tedious `loop` and `put` statements and result in a less concise representation of agents' problems.

Table 6 Application of the EMP/Set syntax to examples in the paper

Examples	Contents of the `empinfo` file using EMP/Set syntax
Listing 3	`equilibrium` `max obj(i) s.t. q(i), objdef(i)`

Listing 5	`equilibrium`
	`max u s.t. x(i), udef, budget`
	`vi mkt(i), p(i), profit, y`

Listing 10	`equilibrium`
	`visol cons(m)`
	`max obj(i) s.t. x(i), objdef(i), cons(m)`

Listing 12	`equilibrium`
	`implicit z, defz`
	`visol demand`
	`min iso_obj s.t. q0, iso_defobj, demand`
	`min agent_obj(i) s.t. q(i,k), z, agent_defobj(i),demand`

Listing 15	`qvi F(i), y(i), x(i), g(i)`

To resolve these issues, we provide an alternative interface, called EMP/Set, which allows sets to be specified in an `empinfo` file. The EMP/Set interprets a specification of a set as enumerating all of its elements. For example, in the aforementioned case we may put `q(i,k)` directly in the `empinfo` file without instantiating each element in set `k`.

Table 6 shows the contents of the `empinfo` file obtained by applying the EMP/Set syntax to some representative examples of this paper.[10] Basically, when a set is specified,[11] all of its elements are enumerated by our interface. Thus, specifying `cons(m)` using the EMP/Set syntax is equivalent to writing `loop(m, put cons(m););`. If a set appears in the objective variable, for example set `i` in Table 6, then our interface additionally assumes that a set of optimization agents is specified, where each agent is indexed by an element of set `i`. They are assumed to have the same direction to optimize. This provides a compact way of representing a set of agents that share the same structure. In this case, set `i` plays a role as a slicing set.

Other than the set specification, the EMP/Set syntax requires a new keyword `s.t.` (implying subject to) for optimization agents and a separator comma between items (variables and equations). Vector notation is no longer supported as sets can be specified. All other syntactic requirements are the same as the original syntax, for example variables come first and equations follow them in the case of an optimization agent.

Below we show how we may use the EMP/Set interface for Listing 3. To write an EMP specification in a file, we use a pair of macros `$onecho` and `$offecho`. It writes the lines between them in a file, `empinfo.txt` in this case. Finally, to parse

[10] The source code for the examples can be found at [7].

[11] Note that aliased sets are not currently supported. For example, if we have `alias(i,j)`, where set `i` is the original set, then we do not allow set `j` to be specified.

the file we include the `empmodel.gms` file using `$libinclude` macro and pass the file name as its argument.

Listing 16 An example of using EMP/Set interface

```
$onecho > empinfo.txt
equilibrium
max obj(i) s.t. q(i), objdef(i)
$offecho

$libinclude empmodel empinfo.txt
```

The EMP/Set interface (`empmodel.gms`) works on top of the existing interface. For a given file written in the EMP/Set syntax, it will automatically transform the set specifications into equivalent `loop` and `put` statements and generate another file `empinfo.gms` in which an `empinfo` file is written using the original syntax. Then the resulting file `empinfo.gms` will be included, and its `empinfo` specification will be parsed by the GAMS/JAMS solver.

We believe the combination of the simplified EMP/Set interface and the more complex and flexible original syntax will allow users to easily specify simple structures and also build more complex EMP models. The implementations are consistent and can be easily updated to incorporate new features as they are developed.

References

1. Aguiar, A., Narayanan, B., Mcdougall, R.: An overview of the GTAP 9 data base. J. Glob. Econ. Anal. **1**(1), 181–208 (2016)
2. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: a fresh approach to numerical computing. SIAM Rev. **59**(1), 65–98 (2017)
3. Britz, W., Ferris, M., Kuhn, A.: Modeling water allocating institutions based on multiple optimization problems with equilibrium constraints. Environ. Model. Softw. **46**, 196–207 (2013)
4. Brook, A., Kendrick, D., Meeraus, A.: GAMS: A User's Guide. The Scientific Press, South San Francisco (1988)
5. Davis, T.A.: UMFPACK (2007). http://faculty.cse.tamu.edu/davis/suitesparse.html. Accessed 14 Dec 2017
6. Dirkse, S.P., Ferris, M.C.: The PATH solver: a non-monotone stabilization scheme for mixed complementarity problems. Optim. Methods Softw. **5**(2), 123–156 (1995)
7. An EMP framework for equilibrium problems (2019). http://pages.cs.wisc.edu/~youngdae/emp. Accessed 16 Feb 2019
8. Facchinei, F., Fischer, A., Piccialli, V.: On generalized Nash games and variational inequalities. Oper. Res. Lett. **35**(2), 159–164 (2007)
9. Facchinei, F., Kanzow, C.: Generalized Nash equilibrium problems. Annals of Operations Research **175**(1), 177–211 (2010)
10. Ferris, M.C., Dirkse, S.P., Jagla, J.H., Meeraus, A.: An extended mathematical programming framework. Comput. Chem. Eng. **33**(12), 1973–1982 (2009)
11. Ferris, M.C., Fourer, R., Gay, D.M.: Expressing complementarity problems in an algebraic modeling language and communicating them to solvers. SIAM J. Optim. **9**(4), 991–1009 (1999)
12. Ferris, M.C., Munson, T.S.: Interfaces to PATH 3.0: design, implementation and usage. Comput. Optim. Appl. **12**(1), 207–227 (1999)
13. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Cengage Learning, Boston (2002)
14. Harker, P.T.: A variational inequality approach for the determination of oligopolistic market equilibrium. Math. Program. **30**, 105–111 (1984)

15. Harker, P.T.: Multiple equilibrium behaviors on networks. Transp. Sci. **22**(1), 39–46 (1988)
16. Harker, P.T.: Generalized Nash games and quasi-variational inequalities. Eur. J. Oper. Res. **54**, 81–94 (1991)
17. Haurie, A., Krawczyk, J.B.: Optimal charges on river effluent from lumped and distributed sources. Environ. Model. Assess. **2**(3), 177–189 (1997)
18. Kim, Y., Ferris, M.C.: SELKIE: a model transformation and distributed solver for structured equilibrium problems. Technical Report, University of Wisconsin-Madison, Department of Computer Sciences (2017)
19. Krawczyk, J.B., Uryasev, S.: Relaxation algorithms to find Nash equilibria with economic applications. Environ. Model. Assess. **5**(1), 63–73 (2000)
20. Leyffer, S., Munson, T.: Solving multi-leader-common-follower games. Optim. Methods Softw. **25**(4), 601–623 (2010)
21. Luna, J.P., Sagastizabal, C., Solodov, M.: A class of Dantzig–Wolfe type decomposition methods for variational inequalty problems. Math. Program. **143**(1), 177–209 (2014)
22. Mathiesen, L.: An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: an example. Math. Program. **37**(1), 1–18 (1987)
23. Murphy, F.H., Sherali, H.D., Soyster, A.L.: A mathematical programming approach for determining oligopolistic market equilibrium. Math. Program. **24**(1), 92–106 (1982)
24. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, New York (2007)
25. Outrata, J.V., Zowe, J.: A Newton method for a class of quasi-variational inequalities. Comput. Optim. Appl. **4**(1), 5–21 (1995)
26. Philpott, A., Ferris, M., Wets, R.: Equilibrium, uncertainty and risk in hydro-thermal electricity systems. Math. Program. **157**(2), 483–513 (2016)
27. Robinson, S.M.: Equations on monotone graphs. Math. Program. **141**(1), 49–101 (2013)
28. Rosen, J.B.: Existence and uniqueness of equilibrium points for concave N-person games. Econometrica **33**(3), 520–534 (1965)
29. Rutherford, T.F.: Extension of GAMS for complementarity problems arising in applied economic analysis. J. Econ. Dyn. Control **19**(8), 1299–1324 (1995)
30. Schiro, D.A., Pang, J.S., Shanbhag, U.V.: On the solution of affine generalized Nash equilibrium problems with shared constraints by Lemke's method. Math. Program. **142**(1), 1–46 (2013)
31. Solodov, M.: Constraint qualifications. In: Cochran, J.J., et al. (ed.), Wiley Encyclopedia of Operations Research and Management Science. Wiley, Inc. (2010)