Complementarity Problems in GAMS and the PATH Solver *

Michael C. Ferris † Todd S. Munson †

September 25, 1998

Abstract

A fundamental mathematical problem is to find a solution to a square system of nonlinear equations. There are many methods to approach this problem, the most famous of which is Newton's method. In this paper, we describe a generalization of this problem, the complementarity problem. We show how such problems are modeled within the GAMS modeling language and provide details about the PATH solver, a generalization of Newton's method, for finding a solution. While the modeling format is applicable in many disciplines, we draw the examples in this paper from an economic background. Finally, some extensions of the modeling format and the solver are described.

Keywords: Complementarity problems, variational inequalities, algorithms
AMS Classification: 90C33,65K10
This paper is an extended version of a talk presented at CEFES '98 (Computation in Economics, Finance and Engineering: Economic Systems) in Cambridge, England in July 1998

*This material is based on research supported by National Science Foundation Grant CCR-9619765 and GAMS Corporation.

†Computer Sciences Department, University of Wisconsin – Madison, 1210 West Dayton Street, Madison, Wisconsin 53706, email:{ferris,tmunson}@cs.wisc.edu

1 Introduction

Modeling languages are becoming increasingly important to application developers as the problems considered become more complex. Modeling languages such as AMPL or GAMS offer an environment tailored to expressing mathematical constructs. They can efficiently manage a large volume of data and allow users to concentrate on the model rather than the solution methodology. Over time, modeling languages have evolved and adapted as new algorithms and problem classes have been explored.

A fundamental mathematical problem is to find a solution to a square system of nonlinear equations. Newton's method, perhaps the most famous solution technique, has been extensively used in practice to calculate a solution. Two generalizations of nonlinear equations are very popular with modelers, the constrained nonlinear system (that incorporates bounds on the variables), and the complementarity problem.

The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. For example, the complementarity problem can be used to model the Karush-Kuhn-Tucker (KKT) optimality conditions for nonlinear programs [25, 26], Wardropian and Walrasian equilibria [20], and bimatrix games [27]. One popular solver for these problems, PATH, is based upon a generalization of the classical Newton method. This method has achieved considerable success on practical problems.

In this paper, we study the complementarity problem from a modeling perspective with emphasis on economic examples, show how to model such problems within the GAMS modeling language, and provide details about the PATH solver. We will assume an elementary understanding of linear programming, including basic duality theory, and a working knowledge of the GAMS modeling system [3].

We begin developing the complementarity framework by looking at the KKT conditions for linear programs. We discuss the adaptations made in GAMS to support the complementarity problem class and provide some additional examples. Section 3 continues by elaborating on the PATH solver, available options, and output. Finally some extensions of the modeling format and additional uses of the solver are given.

2 Complementarity Problems

The transportation model is a simple linear program where demands for a single good must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs and the problems variables are the amounts $x_{i,j}$ to be shipped over each arc $(i,j) \in \mathcal{A}$. The linear program can be written mathematically as

$$\begin{array}{rc} \min_{x \geq 0} & \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \\ \text{subject to} & \sum_{j:(i,j)\in\mathcal{A}} x_{i,j} \leq s_i, \ \forall i \\ & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \ \forall j. \end{array} \tag{1}$$

Here $c_{i,j}$ is the unit shipment cost on the arc (i,j) connecting nodes i and j, s_i is the supply capacity at i and d_j is the demand at j.

Associated with each constraint is a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. If we label the prices on the supply constraint p^s and those on the demand constraint p^d, then intuitively, at each supply node i

$$p_i^s \geq 0, \ s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}.$$

Further, if $s_i > \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, then in a competitive marketplace, no one is willing to pay for more supply from i, thus $p_i^s = 0$. We write these conditions succinctly as:

$$0 \leq p_i^s \quad \perp \quad s_i \geq \textstyle\sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p_i^s$ or $s_i = \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$.

Similarly, for each demand node j

$$p_i^d \geq 0, \quad \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j$$

and $p_i^d = 0$ if there is excess supply, that is $\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} > d_j$. Clearly the supply price at i plus the transportation cost $c_{i,j}$ from i to j must exceed the market price at j, that is $p_i^s + c_{i,j} \geq p_j^d$. Furthermore, if the cost of

delivery strictly exceeds the market price, that is $p_i^s + c_{i,j} > p_j^d$, then nothing is shipped from i to j, so that $x_{i,j} = 0$. An equivalent, but more succinct notation for all the above conditions is

$$\begin{array}{rcll} 0 \le p_i^s & \perp & s_i \ge \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, & \forall i \\ 0 \le p_i^d & \perp & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \ge d_j, & \forall j \\ 0 \le x_{i,j} & \perp & p_i^s + c_{i,j} \ge p_j^d, & \forall (i,j) \in \mathcal{A}. \end{array} \tag{2}$$

The conditions (2) define a linear complementarity problem that is easily recognized as the complementary slackness conditions of the linear program (1). For linear programs the complementary slackness conditions are both necessary and sufficient for x to be an optimal solution of the problem (1). Furthermore, the conditions (2) are also the necessary and sufficient optimality conditions for a related problem in the variables (p^s, p^d)

$$\begin{array}{ll} \max_{p^s,p^d\ge 0} & \sum_j d_j p_j^d - \sum_i s_i p_i^s \\ \text{subject to} & c_{i,j} \ge p_j^d - p_i^s,\ \forall (i,j) \in \mathcal{A} \end{array}$$

termed the dual linear program (hence the nomenclature "dual variables").

Looking at these conditions a bit more closely we can gain further insight into complementarity problems. A solution of (2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However, the key to the modeling power of complementarity is that it chooses which of the inequalities in (2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active.

The GAMS code for the complementarity version of the transportation problem is given in Figure 1; the actual data for the model is assumed to be given in the file `transmcp.dat`. Note that the model written corresponds very closely to (2). In GAMS, the $\perp$ sign is replaced in the `model` statement with a ".". It is precisely at this point that the pairing of variables and equations shown in (2) occurs in the GAMS code, so that in the example, the function defined by `rational` is complementary to the variable `x`. Further information on the GAMS syntax can be found in [35]; more examples of problems written using the complementarity problem are found in MCPLIB [7].

```
sets   i   canning plants,
       j   markets ;

parameter
       s(i)    capacity of plant i in cases,
       d(j)    demand at market j in cases,
       c(i,j)  transport cost in thousands of dollars per case ;

positive variables
       x(i,j)           shipment quantities in cases
       p_demand(j)      price at market j
       p_supply(i)      price at plant i;

equations
       supply(i)      observe supply limit at plant i
       demand(j)      satisfy demand at market j
       rational(i,j);

supply(i) ..     s(i) =g= sum(j, x(i,j)) ;

demand(j) ..     sum(i, x(i,j))  =g=  d(j) ;

rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;

model transport / rational.x, demand.p_demand, supply.p_supply /;

$include transmcp.dat

solve transport using mcp;
```

Figure 1: A simple MCP model in GAMS, `transmcp.gms`

While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1 is independent of the prices p. Since the prices p are variables in the complementarity problem (2), we can easily replace the constant demand d with a function $d(p)$. Clearly, any algebraic function of p that can be expressed in GAMS can now be added to the model given in Figure 1. For example, a linear demand function could be expressed using

```
demand(j) ..     sum(i, x(i,j))  =g=  d(j)*(1-p_demand(j)) ;
```

Note that the demand is rather strange if $p(j)$ exceeds 1. Other more reasonable examples for $d(p)$ are easily derived from Cobb-Douglas or CES utilities. For those examples, the resulting complementarity problem becomes nonlinear in the variables p. Details of complementarity for more general transportation models can be found in [12, 17].

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (2) is replaced by

$$p_i^s(1+t_i) + c_{i,j} \geq p_j^d, \ \forall (i,j) \in \mathcal{A}$$

The taxes can be made endogenous to the model, details are found in [35]. The key point to make is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In fact, in many cases, there is no optimization problem corresponding to the complementarity conditions.

We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast in the following form:

(NCP) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that

$$0 \leq z \perp F(z) \geq 0.$$

Recall that the $\perp$ sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, $z_i F_i(z) = 0$. We frequently refer to this

property as z_i is "complementary" to F_i. A special case of the NCP problem that has received much attention is when F is a linear function, the linear complementarity problem [6].

To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable $z(i)$):

```
z.lo(i) = 0;
```

or

```
positive variable z;
```

Note that GAMS requires the modeler to write `F(z) =g= 0` whenever the variable is lower bounded, and does not allow the alternative form `0 =l= F(z)`.

A Walrasian equilibrium can also be formulated as a complementarity problem. In this case, we want to find a price $p \in \mathbf{R}^m$ and an activity level $y \in \mathbf{R}^n$ such that

$$\begin{array}{rcl} 0 \leq y & \perp & L(p) := -A^T p \geq 0 \\ 0 \leq p & \perp & S(p,y) := b + Ay - d(p) \geq 0. \end{array} \tag{3}$$

Here, $S(p, y)$ represents the excess supply function, and $L(p)$ represents the loss function. Complementarity allows us to choose the activities y_i to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras' law which states that supply equals demand if we assume all prices p will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter. A GAMS implementation of (3) is given in Figure 2.

Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [23].

In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function $d(p)$ in the Walrasian equilibrium (3). The syntax for carrying this out is shown in Figure 3. We use

```
$include walras.dat

positive variables p(i), y(j);
equations S(i), L(j);

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, b(k)*p(k)) / p(i)
         =g= 0;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0;

model walras / S.p, L.y /;
solve walras using mcp;
```

Figure 2: Walrasian equilibrium as an NCP, `walras1.gms`

```
$include walras.dat

positive variables p(i), y(j);
variables d(i);
equations S(i), L(j), demand(i);

demand(i)..
         d(i) =e= c(i)*sum(k, b(k)*p(k)) / p(i) ;

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;
```

Figure 3: Walrasian equilibrium as an MCP, `walras2.gms`

the variables `d` to store the demand function referred to in the excess supply equation. The model `walras` now contains a mixture of equations and complementarity constraints. Since constructs like the above are prevalent in many practical models, the GAMS syntax allows such formulations.

Note that positive variables are paired with inequalities, while free variables are paired with equations. Of course, as a special case, we can take a square system of nonlinear equations and formulate it as a complementarity problem.

(NE) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$ find $z \in \mathbf{R}^n$ such that

$$F(z) = 0.$$

A crucial point misunderstood by many experienced modelers is that *the bounds on the variable determine the relationships satisfied by the function* F. Thus, an MCP is specified by three pieces of data, namely the lower bounds ℓ, the upper bounds u and the function F.

(MCP) Given lower bounds $\ell \in \{\mathbf{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbf{R} \cup \{\infty\}\}^n$ and a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that *precisely* one of the following holds for each $i \in \{1, \ldots, n\}$:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

These relationships define a general MCP (sometimes termed a rectangular variational inequality [22]). Both NCP and NE are special cases of MCP. For example, to formulate an NCP in the MCP format we set

```
z.lo(I) = 0;
```

or declare

```
positive variable z;
```

and to formulate a square system of nonlinear equations in the MCP format we declare

```
free variable z;
```

In both cases, we *must not* modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

A simplification is allowed to the model statement in Figure 3. In many cases, it is not significant to match free variables explicitly to equations; we only require that there are the same number of free variables as equations. Thus, in the example of Figure 3, the model statement could be replaced by

```
model walras / demand, S.p, L.y /;
```

GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equations equals the number of remaining free variables. All variables that are not free and all inequalities must be explicitly matched. This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged.

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of F. If a variable z_i is fixed, then $F_i(z)$ is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when $z_i = \ell_i = u_i$. Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time.

Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

$$F(z) = 0,\ \ell \leq z \leq u,$$

where typically, the bounds on z are inactive at the solution. This is *not* an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function F. If a finite bound is active at a solution, the corresponding component of F is only constrained to be nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do not satisfy $F(z) = 0$. Only if z^* is a solution of MCP with $\ell < z^* < u$ is it guaranteed that $F(z^*) = 0$.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u.$$

The first order optimality conditions of this problem are precisely MCP($\nabla f(x)$, $[\ell, u]$). We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so $\nabla f(x) \geq 0$. Conversely, if x is at its upper bound, then the function must be increasing as x decreases, so that $\nabla f(x) \leq 0$. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in Figure 3 it may be necessary to have an upper bound on the activity level y. In this case, we simply add an upper bound to y in the model statement, and replace the loss equation with the following definition:

```
y.up(j) = 10;
L(j)..   -sum(i, p(i)*A(i,j)) =e= 0 ;
```

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less than inequality or greater than inequality, since this determination depends on the values of the solution variables. We adopt the convention that all bounded variables are paired to equations. Further details on this point are given in Section 3.1. However, let us interpret the relationships that the above change generates. If $y_j = 0$, the loss function can be positive since we are not producing in the jth sector. If y_j is strictly between its bounds, then the loss function must be zero by complementarity; this is the competitive assumption. However, if y_j is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [7, 19, 20].

3 The Path Solver

The PATH solver [8] for mixed complementarity problems (MCPs) was introduced in 1995 and has since become the standard against which new MCP

solvers are compared [1]. The core algorithm is a nonsmooth Newton method [34] to find a zero of the normal map [33]

$$F(\pi(x)) + x - \pi(x)$$

associated with the MCP. Here $\pi(x)$ represents the projection of x onto $[\ell, u]$ in the Euclidean norm. If x is a zero of the normal map, then $\pi(x)$ solves the MCP. A non-monotone pathsearch [9, 16] using the residual of the normal map

$$\|F(\pi(x)) + x - \pi(x)\|$$

as a merit function can be used to improve robustness. Rather than repeat mathematical results from elsewhere, we note that proof of convergence and rate of convergence results can be found in [32], while technical details of the implementation of the code and interfaces to other software packages are given in [18]. The remainder of this section attempts to describe the algorithm features and enhancements from the perspective of a modeler.

To this end, we will assume for the remainder of this paper that the modeler has created a file named `transmcp.gms` which defines an MCP model `transport` using the GAMS syntax described in Section 2. Furthermore, we will assume the modeler is using PATH for solving the MCP at hand.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the `transmcp.gms` file prior to the `solve` statement

   ```
   option mcp = path;
   ```

   PATH will then be used instead of the default solver provided.

2. Rerun the `gamsinst` program from the GAMS system directory and choose PATH as the default solver for MCP.

To solve the problem, the modeler executes the following command:

```
gams transmcp
```

Here `transmcp` can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to [3] for further details.

Code	String	Meaning
1	Normal completion	PATH returned to GAMS without an error
2	Iteration interrupt	PATH used too many iterations
3	Resource interrupt	PATH took too much time
4	Terminated by solver	PATH encountered difficulty and was unable to continue
8	User interrupt	The user interrupted the solution process

Table 1: Solution Status Codes

Code	String	Meaning
1	Optimal	PATH found a solution of the problem
6	Intermediate infeasible	PATH failed to solve the problem

Table 2: Model Status Codes

At this stage, a log (file) is generated that provides details of what GAMS and the PATH solver are doing as time elapses. In Section 3.2, we describe the output generated in this log file for a typical execution of PATH. After GAMS terminates, a listing file is also generated. We now describe the output in the listing file specifically related to the complementarity problem and PATH.

3.1 The Listing File

The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output for the PATH solver, an example of which is given in Figure 4. We use "..." to indicate where we have omitted continuing similar output.

After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. Tables 1 and 2 display the relevant codes that are returned by PATH under different circumstances. A modeler can access these codes within the `transmcp.gms` file using `transport.solstat` and `transport.modelstat` respectively.

After this, a listing of the time and iterations used is given, along with

```
               S O L V E      S U M M A R Y

     MODEL   TRANSPORT
     TYPE    MCP
     SOLVER  PATH                FROM LINE  45

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

 RESOURCE USAGE, LIMIT          0.057     1000.000
 ITERATION COUNT, LIMIT        31        10000
 EVALUATION ERRORS              0            0

 Work space allocated           --    0.06 Mb

PATH 4.0.0: GAMS Link ver007-033, solaris


---- EQU RATIONAL

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york    -0.225    -0.225     +INF     50.000
seattle  .chicago     -0.153    -0.153     +INF    300.000
seattle  .topeka      -0.162    -0.126     +INF        .

...

---- VAR X           shipment quantities in cases

                      LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york       .       50.000     +INF        .
seattle  .chicago        .      300.000     +INF        .

...

**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
                             0  REDEFINED
                             0     ERRORS
```

Figure 4: Listing File from PATH for solving `transmcp.gms`

a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "****". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal. GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle  .chicago     -0.153    -2.000     +INF    300.000 INFES
```

The "marginal" column on an equation returns the level value of the variable that was paired with this equation. If the modeler did not pair a particular equation with a variable, the value returned here corresponds to the variable that GAMS paired with the equation.

Unfortunately, this is not the end of the story. Some equations may have the following form:

```
             LOWER      LEVEL       UPPER     MARGINAL

new-york   325.000    350.000     325.000    0.225 REDEF
```

This should be construed as a warning from GAMS, as opposed to an error. The `REDEF` only occurs if the paired variable to the constraint had a finite lower and upper bound and the variable is at one of those bounds, since at the solution of the complementarity problem the "equation" may not be satisfied. The problem occurs because of a limitation in the GAMS syntax for complementarity problems. The GAMS equations are used to define the function F. The bounds on the function F are derived from the bounds on the associated variable. Before solving the problem, for finite bounded variables, we do not know if the associated function will be positive, negative or zero at the solution. Thus, we do not know whether to define the equation as "=e=", "=l=" or "=g=". GAMS therefore allows any of these, and informs the modeler via the "REDEF" label that internally GAMS has redefined the

bounds so that the solver processes the correct problem, but that the solution given by the solver does not satisfy the original bounds. *Note that this is not an error, just a warning. The solver has solved the complementarity problem specified by this equation.* GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the bounds, not from the equation definition.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of equ.l - equ.lower and equ.l - equ.upper, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. Figure 4 is a good case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

3.2 The Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically displayed when using the code. An example of the log for a particular run is given as Figure 5.

The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If it is directed to read an option file (see Section 3.4 below), then the following output is generated after the PATH banner.

```
Reading options file PATH.OPT
 > output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.
```

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts

```
--- Starting compilation
--- trnsmcp.gms(0) 138 Kb
--- Starting execution
--- trnsmcp.gms(27) 134 Kb
--- Generating model TRANSPORT
--- trnsmcp.gms(41) 134 Kb
---    11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
 Work space allocated           --    0.06 Mb
 Reading the matrix.
 Reading the dictionary.
PATH 4.0.0: GAMS Link ver007-033, solaris
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.0 (Tue Sep 15 16:21:37 1998)
Written by Todd Munson, Steven Dirkse, Michael Ferris, and Christian Kanzow

Crash Log
major  func  diff  size  residual    step       prox   (label)
    0     0             1.0416e+03             0.0e+00 (DEMAND.new-york)
    1     1     3     3 1.0039e+03  1.0e+00    1.0e+01 (DEMAND.new-york)
pn_search terminated: no basis change.

Major Iteration Log
major minor  func  grad  residual    step  type prox    inorm  (label)
    0     0     2     2 1.0039e+03           I 9.4e+00 6.2e+02 (DEMAND.new-york)
    1     1     3     3 8.3923e+02  1.0e+00 SO 3.7e+00 4.6e+02 (DEMAND.new-york)
...
   15     2    17    17 1.0367e-08  1.0e+00 SO 6.2e-06 9.8e-09 (DEMAND.chicago)

 ** EXIT - solution found.
Major Iterations. . . . 15
Minor Iterations. . . . 31
Restarts. . . . . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . 0.010000
Residual. . . . . 1.036687e-08
--- Restarting execution
--- trnsmcp.gms(45) 134 Kb
--- Reading solution for model TRANSPORT
*** Status: Normal completion
```

Figure 5: Log File from PATH for solving `transmcp.gms`

solving the problem.

The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in [10]. The first column of the crash log is just a label indicating the current iteration number, the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See Section 3.6 for further information. The column "step" corresponds to the steplength taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section 3.4 gives mechanisms to affect the behavior of these steps.

After the crash is completed, the main algorithm starts shown by the "Major Iteration Log" flag. The columns that have the same labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [27]. The number of pivots performed `per` major iteration is given in the "minor" column.

If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. In particular, the number of problem variables z, slack variables corresponding to variables at lower bound w and at upper bound v are noted. Artificial variables are also noted in this minor log, see [18] for further details. Again, the option file can be used to affect the frequency of such output.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be

Code	Meaning
C	A cycle was detected.
E	An error occurred in the linear solve.
I	The minor iteration limit was reached.
N	The basis became singular.
R	An unbounded ray was encountered.
S	The linear subproblem was solved.
T	Failed to remain within tolerance after factorization was performed.

Table 3: Linear Solver Codes

taken. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. Table 3 explains the return codes for the linear solver and Table 4 explains the meaning of each step type.

At the end of the log file, summary information regarding the algorithm's performance is given. The string "** EXIT - solution found." is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what `modelstat` and `solstat` will be returned to GAMS. After this, the "Restarting execution" flag indicates that GAMS has been restarted and is processing the results passed back by PATH. Currently, features to detect ill-posed, poorly scaled, or singular models are being incorporated into PATH.

3.3 The Status File

If for some reason the PATH solver exits without writing a solution, or the `sysout` flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but provides more detailed information. The modeler is typically not interested in this output.

3.4 Options

The default options of PATH should be sufficient for most models; the techniques for changing these options are now described. To change the default

Code	Meaning
B	A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.
D	The step was accepted because the Distance between the current iterate and the Newton point was small.
G	A gradient step was performed.
I	Initial information concerning the problem is displayed.
M	The step was accepted because the Merit function value is smaller than the non-monotone reference value.
O	A step that satisfies both the distance and merit function tests.
R	A Restart was carried out.
W	A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the non-monotone reference value (M, O, or B step), regenerated the Newton point, and performed a backtracking search.

Table 4: Step Type Codes

options on the model `transport`, the modeler is required to write a file `path.opt` in the same directory as the model resides and either add a line

```
transport.optfile = 1;
```

before the `solve` statement in the file `transmcp.gms`, or use the command-line option

```
gams transmcp optfile=1
```

We give a list of the available options along with their defaults and meaning in Tables 5 and 6. Note that only the first three characters of every word are significant.

GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the solve statement

```
option iterlim = 2000;
transport.iterlim = 2000;
```

Similarly if the solver runs out of memory, then the workspace allocated can be changed using

```
transport.workspace = 20;
```

The above example would allocate 20MB of workspace for solving the model.

Problems with a singular basis matrix can be overcome by using the `proximal_perturbation` option [2], and linearly dependent columns can be output with the `output_factorization_singularities` option.

In particular, PATH can emulate Lemke's method [5, 27] for LCP with the following options:

```
crash_method none;
major_iteration_limit 1;
lemke_start first;
nms no;
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [24, 30, 29] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

Option	Default	Explanation
convergence_tolerance	1e-6	Stopping criterion
crash_iteration_limit	50	Maximum iterations allowed in crash
crash_method	pnewton	pnewton or none
crash_minimum_dimension	1	Minimum problem dimension to perform crash
crash_nbchange_limit	1	Number of changes to the basis allowed
crash_searchtype	arc	Searchtype to use in the crash method.
cumulative_iteration_limit	10000	Maximum minor iterations allowed
gradient_searchtype	arc	Searchtype to use when a gradient step is taken
gradient_step_limit	5	Maximum number of gradient step allowed before restarting
lemke_start	automatic	Frequency of lemke starts (automatic, first, always)
major_iteration_limit	500	Maximum major iterations allowed
merit_function	fischer	Merit function to use (normal or fischer)
minor_iteration_limit	1000	Maximum minor iterations allowed in each major iteration
nms	yes	Allow line/path searching, watchdoging, and non-monotone descent
nms_initial_reference_factor	20	Controls size of initial reference value
nms_memory_size	10	Number of reference values kept
nms_mstep_frequency	10	Frequency at which m steps are performed

Table 5: PATH Options

Option	Default	Explanation
nms_searchtype	arc	Search type to use (path, line, or arc)
output	yes	Turn output on or off. If output is turned off, selected parts can be turned back on.
output_crash_iterations	yes	Output information on crash iterations
output_crash_iterations_frequency	1	Frequency at which crash iteration log is printed
output_errors	yes	Output error messages
output_factorization_singularities	yes	Output linearly dependent columns determined by factorization
output_initial_point	no	Output initial point given to PATH
output_linear_model	no	Output linear model each major iteration
output_major_iterations	yes	Output information on major iterations
output_major_iterations_frequency	1	Frequency at which major iteration log is printed
output_minor_iterations	yes	Output information on minor iterations
output_minor_iterations_frequency	500	Frequency at which minor iteration log is printed
output_options	no	Output all options and their values
output_warnings	no	Output warning messages
proximal_perturbation	0	Initial perturbation
restart_limit	3	Maximum number of restarts (0 - 3)
time_limit	3600	Maximum number of seconds algorithm is allowed to run

Table 6: PATH Options (cont)

```
crash_method none;
lemke_start always;
nms_initial_reference_factor 1;
nms_memory size 1;
nms_mstep_frequency 1;
nms_searchtype line;
merit_function normal;
```

Note that `nms_memory_size 1` and `nms_initial_reference_factor 1` turn off the non-monotone linesearch, while `nms_mstep_frequency 1` turns off watchdoging [4]. `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, as opposed to the default arcsearch. `merit_function normal` tell PATH to use the normal map for calculating the residual.

3.5 Restarts

The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When we determine that no progress is being made, we restart the problem from the initial point supplied in the GAMS file with a different set of options. The three sets of option choices used during restarts are given in Table 7.

These restarts give us the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

3.6 New Merit Functions

The residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. Currently, PATH can consider an alternative nonsmooth system [21], $\Phi(x) = 0$, where $\Phi_i(x) = \phi(x_i, F_i(x))$ and $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$. The merit function, $\|\Phi(x)\|^2$, in this case is differentiable. When the subproblem solver fails, a projected gradient (of this merit function) is taken. It

Restart Number	Option Values
1	crash_method none nms_initial_reference_factor 2 proximal_perturbation 1e-2*initial_residual
2	crash_method none proximal_perturbation 0
3	crash_method pnewton crash_nbchange_limit 10 nms_initial_reference_factor 2 nms_searchtype line

Table 7: Restart Definitions

is shown in [15] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP are provided under appropriate conditions.

4 Extensions

4.1 MPSGE

The main users of PATH continues to be economists using the MPSGE preprocessor [36]. The MPSGE preprocessor adds syntax to GAMS for defining general equilibrium models. For MPSGE models, the default settings of PATH are slightly different due to the fact that many of these models are basically shocks applied to a calibrated base system with a known solution. If a modeler needs to determine the precise options in use when solving the model, the option `output_options yes` can be used.

4.2 NLP2MCP

NLP2MCP [14] is a tool that automatically converts a nonlinear program into the corresponding KKT conditions[25, 26] which form an MCP. One particular example that can benefit from this conversion is the matrix balancing problem as described in [37].

It is likely that PATH will find a solution to the KKT system. However, this solution is only guaranteed to be a stationary point for the original

problem. It could be a local maximizer as opposed to a minimizer; current work is under way to steer PATH towards local minimizers. Of course, in the convex case, it is easy to prove that every KKT point corresponds to a global solution of the nonlinear program.

4.3 MPEC

A Mathematical Program with Equilibrium Constraint (MPEC) is a nonlinear program with a complementarity problem as one of the constraints. We formally define it as:

$$\begin{array}{rc} \min_{x,y} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0, a \leq x \leq b \\ & y \text{ solves MCP}(F(x,\cdot),[\ell,u]) \end{array}$$

where x are the design variables, y are the state variables, and g is the joint feasibility constraint. There are many applications in economics of the MPEC, including the Stackelberg game [39] and optimal taxation. See [28] for more applications and relevant theory. It is possible to formulate an MPEC in the GAMS modeling language[11].

Currently, only an implementation of the bundle method [31] for solving such problems is available as a GAMS solver. We note that the joint feasibility constraint, g, is not allowed when using the bundle solver. The bundle method reformulates the MPEC as a nondifferentiable program. A nondifferentiable optimization package [38] is then used to solve the problem. We begin by defining the following:

$$Y(x) := \text{SOL}(\text{MCP}(F(x,\cdot),[\ell,u])) \tag{4}$$

We note that for the algorithm to work, the MCP must have a unique solution at each trial value x. The (nonsmooth) nonlinear program then solved is:

$$\min_{a \leq x \leq b} \quad f(x, Y(x))$$

We calculate the required subdifferential using PATH 4.0 and the technique given in [31].

Developing new algorithms for solving MPEC problems continues to be an active research area.

Acknowledgements

We are grateful to many people who have contributed to this work over the past few years. Tom Rutherford implemented the link code between GAMS and a generic solver [13], thus making PATH available to a much larger audience; he continues to challenge the solver and the authors to improve performance. Danny Ralph described the initial ideas behind the PATH solver and proved its convergence. Steven Dirkse implemented the first version of the code and extended the algorithm to significantly improve its performance on large scale problems.

References

[1] S. C. Billups, S. P. Dirkse, and M. C. Ferris. A comparison of large scale mixed complementarity problem solvers. *Computational Optimization and Applications*, 7:3–25, 1997.

[2] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76:533–562, 1997.

[3] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[4] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Study*, 16:1–17, 1982.

[5] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[6] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, Boston, 1992.

[7] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[8] S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.

[9] S. P. Dirkse and M. C. Ferris. A pathsearch damped Newton method for computing general equilibria. *Annals of Operations Research*, 1996.

[10] S. P. Dirkse and M. C. Ferris. Crash techniques for large-scale complementarity problems. In Ferris and Pang [19].

[11] S. P. Dirkse and M. C. Ferris. Modeling and solution environments for MPEC: GAMS & MATLAB. In M. Fukushima and L. Qi, editors, *Reformulation – Nonsmooth, Piecewise Smooth, Semismooth and Smoothing Methods*, pages 127–148. Kluwer Academic Publishers, forthcoming, 1998.

[12] S. P. Dirkse and M. C. Ferris. Traffic modeling and variational inequalities using GAMS. In Ph. L. Toint, M. Labbe, K. Tanczos, and G. Laporte, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, NATO ASI Series F. Springer-Verlag, 1998.

[13] S. P. Dirkse, M. C. Ferris, P. V. Preckel, and T. Rutherford. The GAMS callable program library for variational and complementarity solvers. Mathematical Programming Technical Report 94-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994.

[14] M. C. Ferris and J. D. Horn. NLP2MCP: Automatic conversion of nonlinear programs into mixed complementarity problems. Technical report, Computer Sciences Department, University of Wisconsin, 1998. In preparation.

[15] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. Mathematical Programming Technical Report 98-04, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1998.

[16] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81:53–71, 1994.

[17] M. C. Ferris, A. Meeraus, and T. F. Rutherford. Computing Wardropian equilibrium in a complementarity framework. *Optimization Methods and Software, forthcoming*, 1998.

[18] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications, forthcoming*, 1998.

[19] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*, Philadelphia, Pennsylvania, 1997. SIAM Publications.

[20] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39:669–713, 1997.

[21] A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.

[22] P. T. Harker and J. S. Pang. Finite–dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48:161–220, 1990.

[23] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *Economic Journal*, 107, 1997.

[24] N. H. Josephy. Newton's method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.

[25] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[26] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492. University of California Press, Berkeley and Los Angeles, 1951.

[27] C. E. Lemke and J. T. Howson. Equilibrium points of bimatrix games. *SIAM Journal on Applied Mathematics*, 12:413–423, 1964.

[28] Z.-Q. Luo, J. S. Pang, and D. Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, 1996.

[29] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.

[30] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37:1–18, 1987.

[31] J. V. Outrata and J. Zowe. A numerical approach to optimization problems with variational inequality constraints. *Mathematical Programming*, 68:105–130, 1995.

[32] D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 19:352–389, 1994.

[33] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17:691–714, 1992.

[34] S. M. Robinson. Newton's method for a class of nonsmooth functions. *Set Valued Analysis*, 2:291–305, 1994.

[35] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19:1299–1324, 1995.

[36] T. F. Rutherford. Applied general equilibrium modeling with MPSGE as a GAMS subsystem: An overview of the modeling framework and syntax. *Computational Economics, forthcoming*, 1998.

[37] M. H. Schneider and S. A. Zenios. A comparative study of algorithms for matrix balancing. *Operations Research*, 38:439–455, 1990.

[38] H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM Journal on Optimization*, 2:121–152, 1992.

[39] H. Van Stackelberg. *The Theory of Market Economy*. Oxford University Press, 1952.