SANDIA REPORT

SAND2015-XXXX
Unlimited Release
Printed June 2015

Modeling Mathematical Programs with Equilibrium Constraints in Pyomo

William E. Hart, John D. Siirola

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2015-XXXX
Unlimited Release
Printed June 2015

Modeling Mathematical Programs with Equilibrium Constraints in Pyomo

William E. Hart, John D. Siirola
Sandia National Laboratories
Discrete Math and Optimization
PO Box 5800, MS 1318
Albuquerque, NM 87185
{wehart,jdsiiro}@sandia.gov

Abstract

We describe new capabilities for modeling MPEC problems within the Pyomo modeling software. These capabilities include new modeling components that represent complementarity conditions, modeling transformations for re-expressing models with complementarity conditions in other forms, and meta-solvers that apply transformations and numeric optimization solvers to optimize MPEC problems. We illustrate the breadth of Pyomo's modeling capabilities for MPEC problems, and we describe how Pyomo's meta-solvers can perform local and global optimization of MPEC problems.

Acknowledgment

We thank Carl Laird for helpful discussion regarding the design of complementarity conditions in Pyomo. We thank Michael Ferris and Todd Munson for their assistance with the PATH solver. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Contents

1 Introduction **7**

2 Modeling Equilibrium Conditions **9**
2.1 Complementarity Conditions 9
2.2 Complementarity Expressions 9
2.3 Modeling Mixed-Complementarity Conditions 10

3 MPEC Transformations **15**
3.1 Standard Form 15
3.2 Simple Nonlinear 16
3.3 Simple Disjunction 16
3.4 AMPL Solver Interface 17

4 Solver Interfaces and Meta-Solvers **19**
4.1 Nonlinear Reformulations 19
4.2 Disjunctive Reformulations 19
4.3 PATH and the ASL Solver Interface 20

5 Discussion **23**

References **24**

1 Introduction

Mathematical Programs with Equilibrium Constraint (MPEC) problems arise in a large number of applications in engineering and economic systems [6, 21, 25]. An MPEC is an optimization problem that includes equilibrium contraints in the form of complementarity conditions. Equilibrium constraints naturally arise as the solution to an optimization subproblem (e.g. for bilevel programs), variational inequalities, and complementarity problems [13].

Since MPEC problems frequently arise in practice, many algebraic modeling languages (AML) have integrated capabilities for expressing complementarity conditions [24], including AMLs like AIMMS [1], AMPL [2, 11], GAMS [12], MATLAB [23] and YALMIP [20]. AMLs are high-level programming languages for describing and solving mathematical problems, particularly optimization-related problems [18]. AMLs provide a mechanism for defining variables and generating constraints with a concise mathematical representation, which is essential for large-scale, real-world problems that involve thousands or millions of constraints and variables.

In this paper, we describe new functionality in Pyomo 4.1 for expressing and optimizing MPEC models in the Pyomo modeling environment. Pyomo is an open-source software package that supports the definition and solution of optimization applications using the Python language [27, 26, 14, 15]. Python is a powerful programming language that has a clear, readable syntax and intuitive object orientation. Pyomo uses an object-oriented approach for defining models that contain decision variables, objectives, and constraints. MPEC models can be easily expressed with Pyomo modeling components for complementarity conditions. Further, Pyomo's object-oriented design naturally supports the ability to automate the reformulation of MPEC models into other forms (e.g. disjunctive programs). We describe Pyomo meta-solvers that transform MPECs as MIP or NLP problems, which are then optimized with standard solvers. Further, we describe interfaces to specialized mixed complementarity problem solvers, which solve MPEC problems expressed without an optimization objective.

The remainder of this paper is organized as follows. Section 2 describes how Pyomo supports modeling of equilibrium constraints as mixed-complementarity conditions. Section 3 describes transformation capabilities that automate the reformulation of MPEC models. Section 4 describes meta-solvers in Pyomo that leverage these transformations to support global and local optimization of MPEC problems. Section 5 describes future work that is planned for Pyomo.

2 Modeling Equilibrium Conditions

2.1 Complementarity Conditions

Ferris et al. [7] note that there are a few fundamental forms that account for a wide range of complementarity conditions that arise in practice. Consider a variable x and function $g(x)$. The classical form of complementarity condition can be expressed as

$$x \geq 0 \perp g(x) \geq 0,$$

which expresses the complementarity restriction that at least one of these must hold with equality. When the variable x is bounded such that $x \in [l, u]$, then a *mixed complementarity condition* can be expressed as

$$l \leq x \leq u \perp g(x),$$

which expresses the complementarity restriction that at least one of the following must hold:

$$\begin{array}{rlll} & x = l & \text{and} & g(x) \geq 0, \\ & x = u & \text{and} & g(x) \leq 0, \\ \text{or} & l < x < u & \text{and} & g(x) = 0. \end{array}$$

These forms can be generalized by substituting a function $f(x)$ for the variable x. Thus, a *generalized mixed complementarity condition* can be expressed as

$$l \leq f(x) \leq u \perp g(x),$$

which expresses the complementarity restriction that at least one of the following must hold:

$$\begin{array}{rlll} & f(x) = l & \text{and} & g(x) \geq 0, \\ & f(x) = u & \text{and} & g(x) \leq 0, \\ \text{or} & l < f(x) < u & \text{and} & g(x) = 0. \end{array}$$

For completeness, note that the complementarity condition

$$f(x) \perp g(x) = 0$$

is a special case where the function $f(x)$ is unbounded.

2.2 Complementarity Expressions

The design of complementarity conditions in Pyomo relies on the specification of Pyomo constraint expressions. A Pyomo constraint expression defines an equality, a simple inequality, or a pair of inequalities. For example:

$$\begin{array}{l} expr_1 = expr_2 \\ expr_1 <= expr_2 \\ const_1 <= expr_2 <= const_2 \end{array}$$

where $const_i$ are constant arithmetic expressions that may only contain variables that are fixed, and $expr_i$ are arithmetic expressions that contain unfixed variables.

A complementarity condition is defined with a pair of constraint expressions

$$l_1 <= expr_1 <= u_1 \ \perp \ l_2 <= expr_2 <= u_2,$$

where exactly two of the constant bounds l_1, u_1, l_2 and u_2 are finite. The non-finite bounds values are omitted in practice, so this condition directly describes a classical or mixed complementarity condition. Additionally, a complementarity condition can be expressed with a simple inequality:

$$l_1 <= expr_1 <= u_1 \ \perp \ expr_2 <= expr_3.$$

This complementarity condition is implicitly transformed to a form with constant bounds:

$$l_1 <= expr_1 <= u_1 \ \perp \ expr_2 - expr_3 <= 0.$$

2.3 Modeling Mixed-Complementarity Conditions

Pyomo employs an object-oriented strategy for representing models. A Pyomo model object contains modeling components that define standard elements of algebraic models (e.g. parameters, sets, variables, constraints, and objectives). This allows Pyomo to automatically manage the naming of AML components, and multiple Pyomo models can be simultaneously defined.

Additionally, Pyomo's modeling capabilities can be extended by simply defining new modeling components. Pyomo's `pyomo.mpec` package defines the `Complementarity` component that is used to declare complementarity conditions.

For example, consider the `ralph1` problem in MacMPEC [22]:

$$\begin{array}{ll} \min & 2x - y \\ & 0 \leq y \perp y \geq x \\ & x, y \geq 0 \end{array}$$

The following script defines a Pyomo model for `ralph1`:

```
# file ralph1.py

from pyomo.environ import *
from pyomo.mpec import *

model = ConcreteModel()

model.x = Var( within=NonNegativeReals )
model.y = Var( within=NonNegativeReals )

model.f1 = Objective( expr=2*model.x - model.y )

model.compl = Complementarity(
                expr=complements(0 <= model.y, model.y >= model.x) )
```

The first lines in this script import Pyomo packages:

```
from pyomo.environ import *
from pyomo.mpec import *
```

The first line imports `pyomo.environ` to initialize Pyomo's environment, and it imports the core modeling components from Pyomo. The second line imports modeling components for complementarity conditions. The subsequent lines in this script create a model, declare variables `x` and `y`, declare an objective `f1`, and declare a complementarity condition `compl`.

The complementarity condition is declared with the `Complementarity` component. In the simplest case, this Python class takes a keyword argument `expr` that contains the value of the `complements` function. This function accepts two Pyomo constraint expressions that are used to declare a complementarity condition.

Pyomo also supports indexed components, where a set of components are initialized over an index set using a construction rule. Thus, the `Complementarity` component can be declared with an index set. For example, consider the following model, `indexed`:

$$\begin{array}{ll} \min & \sum_{i=1}^{n} i(x_i - 1)^2 \\ & 0 \leq x_i \perp 0 \leq x_{i+1} \qquad i = 1, \ldots, n-1 \end{array}$$

The following script defines a Pyomo model for `indexed` with $n = 5$:

```
# file ex1a.py

from pyomo.environ import *
from pyomo.mpec import *

n = 5

model = ConcreteModel()

model.x = Var( range(1,n+1) )

model.f = Objective( expr=sum(i*(model.x[i]-1)**2 for i in range(1,n+1)) )

def compl_(model, i):
    return complements(model.x[i] >= 0, model.x[i+1] >= 0)
model.compl = Complementarity( range(1,n), rule=compl_ )
```

The complementarity conditions are defined with a single `Complementarity` component that is indexed over the set $1, \ldots, n-1$ and initialized with a construction rule `compl_`. This rule is a function that accepts a model instance and an index, and returns the i-th complementarity condition.

The declared set of indexes may be a superset of the indices that define complementarity conditions. For example, if the construction rule returns `Complementarity.Skip`, then the corresponding index is skipped. For example:

```
# file ex1d.py

from pyomo.environ import *
from pyomo.mpec import *
```

```
n = 5

model = ConcreteModel()

model.x = Var( range(1,n+1) )

model.f = Objective( expr=sum(i*(model.x[i]-1)**2 for i in range(1,n+1)) )

def compl_(model, i):
    if i == n:
        return Complementarity.Skip
    return complements(model.x[i] >= 0, model.x[i+1] >= 0)
model.compl = Complementarity( range(1,n+1), rule=compl_ )
```

This example can also be expressed with the `ComplementarityList` component:

```
# file ex1b.py

from pyomo.environ import *
from pyomo.mpec import *

n = 5

model = ConcreteModel()

model.x = Var( range(1,n+1) )

model.f = Objective( expr=sum(i*(model.x[i]-1)**2 for i in range(1,n+1)) )

model.compl = ComplementarityList()
model.compl.add( complements(model.x[1] >= 0, model.x[2] >= 0) )
model.compl.add( complements(model.x[2] >= 0, model.x[3] >= 0) )
model.compl.add( complements(model.x[3] >= 0, model.x[4] >= 0) )
model.compl.add( complements(model.x[4] >= 0, model.x[5] >= 0) )
```

This component defines a list of complementarity conditions. The list index can be used in Pyomo, but this component simplifies the declaration of models for which the index values are not important. The `ComplementarityList` component can also be defined with a rule that iteratively returns complementarity conditions:

```
# file ex1c.py

from pyomo.environ import *
from pyomo.mpec import *

n = 5

model = ConcreteModel()

model.x = Var( range(1,n+1) )

model.f = Objective( expr=sum(i*(model.x[i]-1)**2 for i in range(1,n+1)) )
```

```
def compl_(model):
    yield complements(model.x[1] >= 0, model.x[2] >= 0)
    yield complements(model.x[2] >= 0, model.x[3] >= 0)
    yield complements(model.x[3] >= 0, model.x[4] >= 0)
    yield complements(model.x[4] >= 0, model.x[5] >= 0)
model.compl = ComplementarityList( rule=compl_ )
```

Similarly, the construction rule may be a list expression that generates a sequence of complementarity conditions:

```
# file ex1e.py

from pyomo.environ import *
from pyomo.mpec import *

n = 5

model = ConcreteModel()

model.x = Var( range(1,n+1) )

model.f = Objective( expr=sum(i*(model.x[i]-1)**2 for i in range(1,n+1)) )

model.compl = ComplementarityList(
    rule=(complements(model.x[i] >= 0, model.x[i+1] >= 0) for i in range(1,n)) )
```

3 MPEC Transformations

Pyomo's object-oriented design supports the structured transformation of models. Pyomo can iterate through model components as well as nested model blocks. Thus, model components can be easily transformed locally, and global data can be collected to support global transformations. Further, Pyomo components and blocks can be activated and deactivated, which facilitates *in place* transformations that do not require the creation of a separate copy of the original model.

Pyomo's `pyomo.mpec` package defines several model transformations that can be easily applied. For example, if `model` defines an MPEC model (as in our previous examples), then the following example illustrates how to apply a model transformation:

```
transformed = model.transform("mpec.simple_nonlinear")
```

In this case, the `mpec.simple_nonlinear` transformation is applied. The following sections describe the transformations currently supported in `pyomo.mpec`.

3.1 Standard Form

In Pyomo, a complementarity condition is expressed as a pair of constraint expressions

$$l_1 <= expr_1 <= u_1 \ \perp \ l_2 <= expr_2 <= u_2,$$

where exactly two of the constant bounds l_1, u_1, l_2 and u_2 are finite. The non-finite bounds are typically omitted, but the value `None` can be used to express infinite bounds. Additionally, each constraint expression can be expressed with a simple inequality of the form

$$expr_1 <= expr_2.$$

The `mpec.simple_nonlinear` transformation reformulates each complementarity condition in a model into a standard form:

$$l_1 <= expr <= u_1 \ \perp \ l_2 <= var <= u_2,$$

where exactly two of the constant bounds l_1, u_1, l_2 and u_2 are finite, and either l_2 is zero or both l_2 or u_2 are finite.

Note that this transformation creates new variables and constraints as part of this transformation. For example, the complementarity condition

$$1 \leq x+y \ \perp \ 1 \leq 2x-y,$$

get re-expressed as the following:

$$\begin{array}{l} 1 \leq x+y \\ v = 2x-y-1 \\ v \in \mathbb{R}, v \geq 0 \end{array}$$

For each complementary condition object, the new variable and constraints are added as additional components within the complementarity object. Thus, the overall structure of the MPEC model is not changed by this transformation.

3.2 Simple Nonlinear

The `mpec.simple_nonlinear` transformation begins by applying the `mpec.standard_form` transformation. Subsequently, a nonlinear constraint is created that defines the complementarity condition. This is a simple nonlinear transformation adapted from Ferris et al. [8], which can be described by three different cases:

- If l_1 is finite, then the following constraint is defined:

$$(expr - l_1) * v \leq \varepsilon$$

- If u_1 is finite, then the following constraint is defined:

$$(u_1 - expr) * v \leq \varepsilon$$

- If l_2 and u_2 are both finite, then the following constraints are defined:

$$(var - l_2) * expr \leq \varepsilon$$
$$(var - u_2) * expr \leq \varepsilon$$

Each of these cases ensure that the complementarity condition is met when ε is zero. For example, in the first case, we know that $0 \leq v$ and $0 \leq expr - l_1$. When ε is zero, this constraint ensures that either v is zero or $expr - l_1$ is zero.

This transformation uses the parameter `mpec_bound`, which defines the value for ε for every complementarity condition. This allows for the specification of a relaxed nonlinear problem, which may be easier to optimize with some nonlinear programming solvers. The default value of `mpec_bound` is zero.

3.3 Simple Disjunction

The `mpec.simple_disjunction` transformation expresses a complementarity condition as a disjunctive program. We are given a complementarity condition defined with a pair of constraint expressions

$$l_1 <= expr_1 <= u_1 \ \perp \ l_2 <= expr_2 <= u_2,$$

where exactly two of the constant bounds l_1, u_1, l_2 and u_2 are finite. Without loss of generality, we assume that either l_1 or u_1 is finite.

This transformation can be described by three different cases:

- If the first constraint is an equality, then the complementarity condition is trivially replaced by that equality constraint.

- If both bounds on the first constraint are finite but different, then the disjunction has the form:

$$\begin{bmatrix} Y_1 \\ l_1 = expr_1 \\ expr_2 \geq 0 \end{bmatrix} \bigvee \begin{bmatrix} Y_2 \\ expr_1 = u_1 \\ expr_2 \leq 0 \end{bmatrix} \bigvee \begin{bmatrix} Y_3 \\ l_1 \leq expr_1 \leq u_1 \\ expr_2 = 0 \end{bmatrix}$$

$$Y_1 \veebar Y_2 \veebar Y_3 = True$$

$$Y_1, Y_2, Y_3 \in \{True, False\}$$

- Otherwise, each constraint is a simple inequality. The complementarity condition is reformulated as

$$0 <= \overline{expr}_1 \ \perp \ 0 <= \overline{expr}_2,$$

  and the disjunction has the form:

$$\begin{bmatrix} Y \\ 0 = \overline{expr}_1 \\ 0 \leq \overline{expr}_2 \end{bmatrix} \bigvee \begin{bmatrix} \neg Y \\ 0 \leq \overline{expr}_1 \\ 0 = \overline{expr}_2 \end{bmatrix}$$

$$Y \in \{True, False\}$$

This transformation makes use of modeling components and transformations from Pyomo's `pyomo.gdp` package [29]. The transformation expresses each of the disjunctive terms explicitly using `Disjunct` components and the *select exactly one* logical condition using the `Disjunction` component. The transformation adds the `Disjunct` and `Disjunction` components within the objects that represent the complementarity conditions. It then recasts the modified complementarity components into simple `Block` components. This localizes all changes to the model to the individual complementarity components. Subsequent transformation of the disjunctive expressions to algebraic constraints can be effected through either Big-M (`gdp.bigm`) or Convex Hull (`gdp.chull`) transformations.

3.4 AMPL Solver Interface

Solvers like PATH [5] have been tailored to work with the AMPL Solver Library (ASL). AMPL uses `nl` files to communicate with solvers, which read `nl` files with the ASL. Pyomo can also create `nl` files, and the `mpec.nl` transformation processes `Complementarity` components into a canonical form that is suitable for this format [7].

4 Solver Interfaces and Meta-Solvers

Pyomo supports interfaces to third-party solvers as well as meta-solvers that apply transformations and third-party solvers, perhaps in an iterative manner. The `pyomo.mpec` package includes an interface to the `PATH` solver, as well as several meta-solvers. These are described in this section, and examples are provided that employ the `pyomo` command-line interface.

4.1 Nonlinear Reformulations

The `mpec.simple_nonlinear` transformation provides a generic way for transforming an MPEC into a nonlinear program. When the MPEC only has continuous decision variables, the resulting model can be optimized by a wide range of solvers.

For example, the `pyomo` command-line interface allows the user to specify a nonlinear solver and a model transformation that is applied to a model:

```
pyomo solve --solver=ipopt --transform=mpec.simple_nonlinear ex1a.py
```

This example illustrates the use of the `ipopt` interior-point solver with the `mpec.simple_nonlinear` transformation. When a transformation is used directly like this, the results that are returned to the user include decision variables for the transformed model. Pyomo does not have general capabilites for mapping a solution back into the space from the original model. In this example, the results object includes values for the x variables as well as the variables v introduced when applying the transformation to the standard form (see above).

Pyomo includes a meta-solver, `mpec_nlp` that applies the nonlinear transformation, performs optimization, and then returns results for the original decision variables. For example, `mpec_nlp` executes the same logic as the previous `pyomo` example:

```
pyomo solve --solver=mpec_nlp ex1a.py
```

Additionally, this meta-solver can also manipulate the ε values in the model, starting with larger values and iteratively tightening them to generate a more accurate model.

```
pyomo solve --solver=mpec_nlp \
            --solver-options="epsilon_initial=1e-1 epsilon_final=1e-7" ex1a.py
```

This approach may be useful when using a nonlinear solver that has difficulty optimizing with equality constraints.

4.2 Disjunctive Reformulations

The `mpec.simple_disjunction` transformation provides a generic way for transforming an MPEC into a disjunctive program. The `mpec_minlp` solver applies this transformation to create a nonlinear disjunctive program, and then further reformulates the disjunctive model using a "Big-M" transformation that is provided by the `pyomo.gdp` package. The resulting transformation is similar the

reformulation of bilevel models described by Fortuny-Amat and McCarl [10]. If the original model was nonlinear, then the resulting model is a mixed-integer nonlinear program (MINLP). Pyomo includes interfaces to solvers that use the AMPL Solver Library (ASL), so `mpec_minlp` can optimize nonlinear MPECs with a solver like Couenne [3].

If the original model was a linear MPEC, then the resulting model is a mixed-integer linear program that can be globally optimized (e.g. see Hu et al. [16], Júdice [17]). For example, the `pyomo` command can be used to execute the `mpec_minlp` solver using a specified MIP solver:

```
pyomo solve --solver=mpec_minlp --solver-options="solver=glpk" ralph1.py
```

Note that Pyomo includes interfaces to a variety of commonly used MIP solvers, including CPLEX, Gurobi, CBC, and GLPK.

4.3 PATH and the ASL Solver Interface

Pyomo's solver interface for the AMPL Solver Library (ASL) applies the `mpec.nl` transformation, writes an AMPL `.nl` file, executes an ASL solver, and then loads the solution into the original model. Pyomo provides a custom interface to the PATH solver [5], which simply allows the solver to be specified as `path` while the solver executable is named `pathamp`.

The `pyomo` command can execute the PATH solver by simply specifying the `path` solver name. For example, consider the `munson1` problem from MCPLIB:

```
# file munson1.py

from pyomo.environ import *
from pyomo.mpec import *


model = ConcreteModel()

model.x1 = Var()
model.x2 = Var()
model.x3 = Var()

model.f1 = Complementarity(expr=
                complements(model.x1 >= 0,
                            model.x1 + 2*model.x2 + 3*model.x3 >= 1))

model.f2 = Complementarity(expr=
                complements(model.x2 >= 0,
                            model.x2 - model.x3 >= -1))

model.f3 = Complementarity(expr=
                complements(model.x3 >= 0,
                            model.x1 + model.x2 >= -1))
```

This problem can be solved with the following command:

```
pyomo solve --solver=path munson1.py
```

5 Discussion

Pyomo supports the ability to model complementarity conditions in a manner that is similar to other AMLs. For example, Pyomo's `pyomo.data` package [28] includes Pyomo formulations for many of the MacMPEC [22] and MCPLIB [4] models, which were originally formulated in GAMS and AMPL. However, Pyomo does not currently support related modeling capabilities for equilibrium models, variational inequalities and embedded models, which are supported by the GAMS extended mathematical programming framework [9].

The tranformations and meta-solvers currently included in Pyomo illustrate how Pyomo's MPEC modeling capability can be leveraged. We expect these capabilities to mature and expand in response to application needs. For example, the `mpec.simple_nonlinear` transformation could be expanded to support reformulations that are well-suited for sequential quadratic programming solvers [19]. Similarly, current meta-solvers could be extended to directly support the communication of suffix information from the solver back to the original model.

References

[1] AIMMS. AIMMS home page. http://www.aimms.com, 2008.

[2] AMPL. AMPL home page. http://www.ampl.com/, 2008.

[3] COUENNE. Couenne home page. http://www.coin-or.org/Couenne/, 2015.

[4] Steven P. Dirkse and Michael C. Ferris. MCPLIB: A collection of nonlinear mixed-complementarity problems. *Optimization Methods and Software*, 5(4):319–345, 1995.

[5] Michael C. Ferris and Todd S. Munson. Complementarity problems in GAMS and the path solver. *Journal of Economic Dynamics and Control*, 24(2):165–188, 2000.

[6] Michael C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39(4):669–713, 1997.

[7] Michael C. Ferris, Robert Fourer, and David M. Gay. Expressing complementarity problems in an algebraic modeling language and communicating them to solvers. *SIAM J. Optimization*, 9(4):991–1009, 1999.

[8] Michael C. Ferris, Steven P. Dirkse, and A. Meeraus. Mathematical programs with equilibrium constraints: Automatic reformulation and solution via constrained optimization. In T. J. Kehoe, T. N. Srinivasan, and J. Whalley, editors, *Frontiers in Applied General Equilibrium Modeling*, pages 67–93. Cambridge University Press, 2005.

[9] Michael C. Ferris, Steven P. Dirkse, Jan-H. Jagla, and Alexander Meeraus. An extended mathematical programming framework. *Computers and Chemical Engineering*, 33(12): 19731982, 2009.

[10] José Fortuny-Amat and Bruce McCarl. A representation and economic interpretation of a two-level programming problem. *The Journal of the Operations Research Society*, 32(9): 783–792, 1981.

[11] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming, 2nd Ed.* Brooks/Cole–Thomson Learning, Pacific Grove, CA, 2003.

[12] GAMS. GAMS home page. http://www.gams.com, 2008.

[13] 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.

[14] William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3), 2011.

[15] William E. Hart, Carl Laird, J.-P. Watson, and David L. Woodruff. *Pyomo: Optimization Modeling in Python*. Springer, 2012.

[16] Jing Hu, John E. Mitchell, Jong-Shi Pang, Kristin P. Bennett, and Gautam Kunapuli. On the global solution of linear programs with linear complementarity constraints. *SIAM J. Optimization*, 19(1):445–471, 2008.

[17] Joaquim J. Júdice. Algorithms for linear programming with linear complementarity constraints. *TOP*, 20(1):4–25, 2011.

[18] Josef Kallrath. *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, 2004.

[19] Sven Leyffer. Complementarity constraints as nonlinear equations: Theory and numerical experience. In S. Dempe and V. Kalishnikov, editors, *Optimization with Multivalued Mappings*, pages 169–208. Springer, 2006.

[20] Johan Löfberg. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE Intl Symp on Computer Aided Control Systems Design*, 2004.

[21] Z.-Q. Lou, J.-S. Pang, and D. Ralph. *Mathematical Programming with Equilibrium Constraints*. Cambridge University Press, Cambridge, UK, 1996.

[22] MacMPEC. MacMPEC: AMPL collection of MPECs. http://www.mcs.anl.gov/~leyffer/MacMPEC/, 2000.

[23] MATLAB. *User's Guide*. The MathWorks, Inc., 1992.

[24] Todd S. Munson. *Algorithms and Environments for Complementarity*. PhD thesis, University of Wisconsin, Madison, 2000.

[25] J. Outrata, M. Kocvara, and J. Zowe. *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints*. Kluwer Academic Publishers, Dordrecht, 1998.

[26] Pyomo Home. Pyomo home page. https://www.pyomo.org, 2015.

[27] Pyomo Software. Pyomo software trac site. https://software.sandia.gov/svn/public/pyomo/, 2015.

[28] pyomo.data. pyomo.data: Models and examples for pyomo. https://software.sandia.gov/svn/public/pyomo/pyomo.data, 2015.

[29] John D. Siirola. Modeling generalized disjunctive programs in pyomo. Technical report, Sandia National Laboratories, 2015. (in preparation).

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)

Sandia National Laboratories