OPTIMA 103

Mathematical Optimization Society Newsletter

MOS Chair's Column

September 1, 2017. In this issue of *Optima* we are seriously starting the countdown to ISMP in Bordeaux! All prize committees are up and running and the Symposium Advisory Committee is busy soliciting preproposals for the 2021 ISMP. The Bordeaux program and organizing committees are working hard on putting together a fantastic scientific program.

MOS is run purely by enthusiastic researchers and I feel very grateful to all colleagues who invest a lot of time and effort on all our committees, journals, newsletter, and meetings. We can all help in the committee work by nominating researchers for the prizes! Do not assume that "someone else most likely does it", but get together and nominate! All prize rules can be found in the MOS Bylaws on our Web page. The calls for nominations can be found in this issue of *Optima* and on our Web page. Notice that the deadlines for the Dantzig and Lagrange Prizes are earlier than the other ones, which is due to some practical reasons.

The ISMP is one of the backbones of MOS and we are looking for an enthusiastic group of organizers for the 2021 ISMP. Are you one of those who feel that it would be fun to be a part of an organizing team? To demonstrate the activities in your country? The 2021-ISMP should be held outside of Europe, which opens up for a lot of interesting alternatives! Get in touch with the Symposium Advisory Committee Chair Jon Lee (see the call for preproposals here in *Optima*) and discuss the possibilities.

Finally, at the ISMP there will be a Business meeting. Here, we can discuss topics that you feel are important that we deal with in the Society. Do you want to suggest such a topic? Get in touch with me and help me make the business meeting a lively relevant arena for discussion!

Karen Aardal
Delft Institute of Applied Mathematics
k.i.aardal@tudelft.nl

Contents of Issue 103 / September 2017

Note from the Editors

Dear MOS members,
Algebraic modeling languages provide an indispensable link between specific problems that need to be solved and all of our community's efforts over the last decades: fascinating structural insights, innovative algorithmic developments, and critical software engineering advancements. We invite you to read and enjoy this issue, which is dedicated to algebraic modeling languages – particularly ones that have appeared and grown quickly in the last few years.

Sam Burer, Co-Editor, Volker Kaibel, Editor,
Jeff Linderoth, Co-Editor

Matthew J. Saltzman

A Lightning Tour of the Optimization Modeling Software Landscape

This issue of *Optima* is devoted to a sampling of software tools that support the formulation of mathemtatical optimization models and the generation and management of model instances. The other articles in this issue describe particular packages representing the state of the art in modeling software. This article attempts to describe the range of different types of software modeling tools available (beyond those described in the other articles) and to make note of some of the more widely known representatives of each class. There is no hope in this limited space of including a comprehensive list of tools or even a comprehensive list of features of the tools that are covered, but we do try to indicate the range of types of tools, representative products, and key features.

The earliest computer solution of an optimization problem that I am aware of is a 1953 paper solving LPs up to 10×10 using the simplex method [2]. Even if the problem input format was dense, it would not have been too difficult to prepare input for 120 matrix and rim vector (objective, right-hand side, and bounds) values by hand. But even by the late 1950's, the process of preparing input for an optimization solver was challenging enough to require technological assistance. The first technology for input preparation was matrix generators: special-purpose codes in FORTRAN or other conventional programming languages that produced input files in MPS format – a sparse matrix representation developed by IBM, or some similar format. MPS is still in wide use today as a data interchange format for optimization models, although there was never a formal standard for it so different solver vendors support the format with minor variations.

MPS format is not a natural way to think about optimization model formulation. For one thing, it is column oriented, which is not the way we naturally formulate models. It utilizes fixed fields, natural for the ancient Hollerith punchcard but not for humans, and it requires row and column identifiers for every coefficient. Thus, it is impractical to create MPS input files in any other way than by writing a matrix generator program. Aside from being a "write-only" language, MPS format has no way to express the structure of a model other than through row and column naming conventions. We naturally think in blocks of variables and constraints, but MPS format supports only simple lists of coefficients. Also, writing a matrix generator requires knowledge of a conventional programming or scripting language, restricting the community of users to experienced programmers.

The natural next step in the evolution of technology for interacting with optimization solvers was the development of algebraic modeling languages. All algebraic modeling languages accomplish two objectives:

- They support the representation of an abstract model in a form that is familiar to the modeler, similar to the mathematical notation that we use to write formulations, with summation operators, symbolic constant and variable names, and indexes.
- They separate the abstract structure of the model from the data associated with an instance of the model, so that the same abstract representation can be associated with coefficient values for any instance.

Some of these tools are published by solver vendors and link specifically to the vendor's solver library, however the ones from independent modeling system vendors have an additional objective:

- They separate the construction of model and instance from the choice of solver, so that users can select the solver best suited to their needs.

There are three broad classes of modeling tools for optimization:

- standalone languages;
- spreadsheet-based tools; and
- tools integrated into object-oriented programming languages.

The latter class still requires significant programming experience. Several of the remaining articles in this collection review tools in this class. Following is a brief inventory of key players not covered in those articles.

1 Spreadsheet modeling

Spreadsheets are the closest thing in the business world to a universal language for dealing with numbers, so there is a natural urge among optimizers to integrate optimization tools into spreadsheets. Microsoft Excel includes a Solver add-in from Frontline Systems, Inc. (Frontline add-ins with more advanced features and access to commercial solver libraries are available separately at additional cost.) Similar functionality is available in OpenOffice.org and LibreOffice open-source office suites.

These plugins all work along the same lines.

1. The user sets aside a block of cells to serve as variables and creates formulas computing the values of the objective and left- and right-hand sides of constraints.
2. The user invokes the solver, sets a target cell containing the objective formula, the type of optimization, the block of variable cells, constraint types, and left- and right-hand constraint expressions.
3. The user presses a Solve button and the solver starts. On convergence, the optimal variable values are stored in the designated variable cells and the optimal objective value appears in the designated objective cell.

The pro for systems like this is that they operate within a spreadsheet, so the environment is familiar and they are easy to learn. The cons are that for large problems, they are difficult to validate and the default solver engines are not the most robust.

Andrew Mason et al.'s OpenSolver add-in for Excel leverages the popularity of Excel's Solver by supporting additional solver engines, adding a more intuitive model editor, and providing model validation tools that highlight model features in the spreadsheet display. OpenSolver for Google sheets adds similar capabilities for users of Google's online spreadsheet tool.

Mason's SolverStudio Excel add-in provides an interface between the spreadsheet and a number of other modeling languages that are more suitable than the spreadsheet for building large-scale production models. SolverStudio manages the integration of the spreadsheet data with these modeling languages, allowing users to build and solve complex models from within the Excel application.

2 Standalone modeling languages and systems

Most MILP solver vendors support a simple and straightforward input format commonly referred to as LP or LINDO format. This format is row oriented and supports linear expressions described as juxtapositions of (nonzero) numerical constants and identifiers separated by plus signs. While this format is sparse, it is suitable only for very small problems. It also does not achieve either of the goals of modeling systems. Problem structure is represented only through variable naming conventions and the values of constants are included in the problem statement. It is also not a reliable interchange format, as it is not standardized. For example, some versions indicate multiplication of a variable by a constant using juxtaposition, and others require the * operator to indicate multiplication.

Figure 1 displays an example of a linear program written in LP format (adapted from the COIN-OR Branch-and-Cut solver example files).

```
Minimize
 OBJ: COL01 + 2 COL05 - COL08
Subject To
 ROW01: 3 COL01 + COL02 - 2 COL04 - COL05 - COL08 >= 2.5
 ROW02: 2 COL02 + 1.1 COL03 <= 2.1
 ROW03: COL03 + COL06  = 4
 ROW04: 2.8 COL04 - 1.2 COL07 - RgROW04  = 1.8
 ROW05: 5.6 COL01 + COL05 + 1.9 COL08 - RgROW05  = 15
Bounds
      COL01 >= 2.5
 0 <= COL02 <= 4.1
 0 <= COL03 <= 1
 0 <= COL04 <= 1
 0.5 <= COL05 <= 4
 0 <= COL08 <= 4.3
 0 <= RgROW04 <= 3.2
-12 <= RgROW05 <= 0
Binaries
 COL03  COL04
End
```

Figure 1. *A linear program in LP format. (Variables are assumed to be nonnegative and continuous unless otherwise indicated with* `bounds`, `integer`, *or* `binaries` *keywords.)*

3 Algebraic languages

Algebraic modeling languages (AMLs) are small languages for describing models in an abstract, structured format similar to mathematical notation and for associating the data corresponding to an instance with the model. The AML "compiler" builds an internal representation of an instance and invokes a solver, either through a dynamically linked library or via an intermediate file and an external command. AMLs may also include scripting languages, allowing the user to control a workflow involving multiple, possibly dynamically constructed models and instances. They also provide mechanisms for setting solver options and parameters.

Some solver vendors support proprietary AML tools. ILOG/IBM (purveyor of CPLEX) provides OPL, FICO (purveyor of Xpress) provides Mosel, and SAS provides SAS/OR. These systems are tied closely to their respective optimization engines – they support all types of models that their solvers handle, but they don't support changing solvers.

```
set ORIG; # origins
set DEST; # destinations

set LINKS within {ORIG,DEST};

param supply {ORIG} >= 0; # amounts available at origins
param demand {DEST} >= 0; # amounts required at destinations

 check: sum {i in ORIG} supply[i] = sum {j in DEST} demand[j
     ↪ ];

param cost {LINKS} >= 0; # shipment costs per unit
var Trans {LINKS} >= 0; # units to be shipped

minimize Total_Cost:
 sum {(i,j) in LINKS} cost[i,j] * Trans[i,j];

subject to Supply {i in ORIG}:
 sum {(i,j) in LINKS} Trans[i,j] = supply[i];

subject to Demand {j in DEST}:
 sum {(i,j) in LINKS} Trans[i,j] = demand[j];

data;

param: ORIG: supply :=
  GARY 1400 CLEV 2600 PITT 2900 ;

param: DEST: demand :=
  FRA 900 DET 1200 LAN 600 WIN 400
  STL 1700 FRE 1100 LAF 1000 ;

param: LINKS: cost :=
  GARY DET 14 GARY LAN 11 GARY STL 16 GARY LAF 8
  CLEV FRA 27 CLEV DET 9 CLEV LAN 12 CLEV WIN 9
  CLEV STL 26 CLEV LAF 17
  PITT FRA 24 PITT WIN 13 PITT STL 28 PITT FRE 99 ;
```

Figure 2. *A transportation problem in AMPL. The contents of the* `data` *section can be stored in a separate file or pulled from a database.*

There are a number of commercial and open-source solver-independent AMLs, which can interact with a variety of commercial and open-source solver libraries. Among the most widely known of these are the commercial products GAMS, AMPL, AIMMS, LINGO, and Maximal MPL.

All of these AMLs support mixed-integer linear programs (MILP). Most also support some or all of quadratic, quadratically constrained, second-order conic, general nonlinear, and constraint programs, as well as access to spreadsheets or databases to acquire instance data.

As an example of the capabilities of these languages, Figure 2 displays a simple transportation model in AMPL from [1]. Other AMLs support similar capabilities and syntax.

Graphical user interfaces

The language itself is the main focus of AML technology, but a number of AML projects include graphical interfaces to help make the development process more friendly to users, especially less technically experienced ones. For most projects, these "integrated development environments" (IDEs) simply collect pieces of the development environment in one convenient window. Typically, there are panels for browsing directories and files, for editing files, and for a "console window" where the user can enter commands to run the compiler and solver and view the results of a run. Some IDEs also offer object browsers that display properties of the modeling objects, such as a variable's domain, a constraint's sense, a coefficient's value, etc and matrix viewers that display the nonzero structure of a system of linear constraints. Many of these environments are built on the Eclipse open-source IDE construction system, although some are custom built.

CPLEX's OPL Studio, AMPL, GAMS, LINGO, and MPL all provide this sort of IDE. The AIMMS IDE is a powerful, object-oriented GUI construction tool that supports building production environments for creating and deploying model instances, including structured windows for inputting instance data and generating reports in a form directly usable by the decision maker. FICO's Optimization Workbench is also designed for building and deploying production GUI tools. SAS/OR supports formatted reporting and access to simulation and project scheduling tools.

4 Object-oriented language extensions

Object-oriented programming languages such as C++, Java, and Python support the creation of language extensions through creation of new object types (called "classes"). A number of optimization modeling tools exploit this capability to add modeling objects (variables, objective functions, constraints) to these languages. The relationship is symbiotic: the language gains these new object types and the modeling system uses the language to dynamically build and operate on problem instances, similar to the scripting capabilities in some standalone modeling languages. Having these tools integrated directly with powerful programming languages means that they can be used to deploy complex optimization-based systems efficiently and robustly, but they do require significant levels of skill to work with the underlying languages.

Several object-oreinted tools are described in the other essays in this newsletter. Some optimization engine vendors support proprietary object-based tools. CPLEX's Concert Technology supports C++, Microsoft's .NET, and Java. Gurobi supports objects in C++, Java, .NET, and Python. Other open-source tools in this class available from COIN-OR include Pyomo (for Python), Rehearse (for C++), FLOPC++ (for C++) and jORLib (for Java).

5 Conclusion

Managing optimization models and instances is a challenging task for practitioners and researchers alike. Also, it is desirable to teach students about optimization without them getting hung up on writing computer programs to generate instances. A wide variety of software tools is available to support these efforts in a number of different ways, including spreadsheet tools, algebraic modeling languages and development environments, and embedded enhancements for object-oriented programming languages.

Matthew J. Saltzman, Dept. of Mathematical Sciences, Clemson University, Clemson SC, USA. mjs@clemson.edu

References

[1] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming, 2/e*, Duxbury/Thomson, 2003.

[2] A. Hoffman, M. Mannos, D. Solokowsky, and N. Weigmann, "Computational experience in solving linear programs," *SIAM Journal 1*, 1953, 1–33.

Iain Dunning, Joey Huchette, and Miles Lubin

JuMP: An algebraic modeling language in Julia

JuMP (github.com/JuliaOpt/JuMP.jl) is an algebraic modeling language (AML), written and embedded in the Julia programming language. JuMP, a relatively new addition to the AML landscape, has seen rapidly growing usage in research, teaching, and industry over its five-year history. JuMP's impact was recently recognized by the 2016 INFORMS Computing Society Prize. In this article, we provide a brief overview of JuMP, its history, and our current efforts to release JuMP 1.0.

1 Origins and motivations

Work on the project that would become JuMP started in October 2012 as an experiment by Miles and Iain to test out the newly announced Julia programming language for tasks important to the practice of operations research. Julia claimed to offer both the performance of low-level languages like C++ and the ease-of-use of high-level languages like Python and MATLAB, a very powerful combination. Miles and Iain's initial experiments [1] provided strong evidence that Julia could indeed be a suitable programming language for both modeling optimization problems and developing solvers. The prototype of JuMP was able to generate linear programming (LP) models in times competitive with commercial AMLs. Julia's suitability for writing solvers was evaluated by comparing Julia with C++, MATLAB, Python, PyPy (a Python accelerator), and Java on key subroutines of the simplex method for LP. At that time, Julia claimed to perform within a factor of 2 of C++, and that claim was verified in these experiments. Compiler enhancements since then have made it possible more generally to match or outperform C++ on certain tasks. This early success, combined with Miles and Iain's dissatisfaction with the trade-offs presented by existing AMLs [2], motivated continued development.

Joey joined the JuMP team before its first public release in October 2013, and JuMP quickly grew to support the wide range of problem classes and solvers which it does today: quadratically constrained quadratic optimization, semidefinite optimization, and derivative-based constrained nonlinear optimization.

2 Characteristics of JuMP

We briefly list some of the defining design decisions and characteristics of JuMP that, taken as a whole, distinguish it both from traditional commercial AMLs and from other open-source tools. These are described in much more elaborate detail in both [2] and the JuMP documentation. JuMP:

- ▷ is embedded in a programming language, which naturally provides facilities for modular problem generation, data input and output, interactivity, and visualization (e.g., Jupyter notebooks).
- ▷ is solver-independent, with support for over a dozen mainstream solvers.
- ▷ offers very expressive and compact syntax for an embedded AML by using Julia's *macro* functionality to capture expressions.
- ▷ lets users index variables and constraints with arbitrary sets.
- ▷ generally does not, however, allow parametric modification of index sets or data after a model has been constructed.
- ▷ avoids transformations and presolve reductions.
- ▷ enables efficient, direct, in-memory modification of models for hot-starts during iterative algorithms.
- ▷ is readily extensible to nontraditional problem classes like robust optimization and sum-of-squares optimization.
- ▷ has native support for automatic differentiation, including user-defined functions.
- ▷ has connections to best-of-breed commercial and open-source optimization solvers, including: Artelys Knitro, BARON, Clp, Cbc, CPLEX, ECOS, GLPK, Gurobi, Ipopt, Mosek, SCIP, Xpress, and more.
- ▷ is available on any platform with Julia support, from a Windows or Mac personal computer to a supercomputer running Linux on thousands of cores.

JuMP offers a number of important features that were typically only accessible in solver-specific APIs in C or C++, making it an ideal platform for developing new models and algorithmic methods with much less effort than before. JuMP is simple enough to use in a classroom environment, even though it was not designed primarily for education.

We provide a basic integer programming formulation for Sudoku to illustrate JuMP's syntax.

```
m = Model()
# x[i,j,k] = 1 <-> cell (i,j) has value k
@variable(m, x[1:9, 1:9, 1:9], Bin)

@constraints(m, begin
    # Only one value appears in each cell
    cell[i=1:9, j=1:9], sum(x[i,j,:]) == 1
    # Each value appears in each row once only
    row[i=1:9, k=1:9], sum(x[i,:,k]) == 1
    # Each value appears in each column once only
    col[j=1:9, k=1:9], sum(x[:,j,k]) == 1
    # Each value appears in each 3x3 subgrid once only
    subgrid[i=1:3:7,j=1:3:7,val=1:9], sum(x[i:i+2,j:j+2,val])
         ↪  == 1
end)
```

3 What's happening now

In June of this year we held our first "Developers Meetup/Workshop" (www.juliaopt.org/developersmeetup/) at MIT's Sloan School of Business. The 5-day event had 14 half-hour-or-longer talks, in addition to which we had plenty of time for informal discussions and brainstorming. During the meetup, the developer community came to a consensus on replacing our solver-independent abstraction layer `MathProgBase` with a new second-generation abstraction layer to be called `MathOptInterface` (MOI). MOI will provide a solid foundation for the JuMP ecosystem to grow and support more problem classes and functionality than before. JuMP 0.19, to be released this fall, will be the first to support the new abstraction layer.

JuMP 1.0 is planned for 2018. Until then, we advise users to continue viewing JuMP and Julia itself from the perspective of early adopters. Things may break along the way, but you might just get where you want faster than with other tools!

Iain Dunning, Research Engineer, DeepMind, London, UK.

Joey Huchette, Operations Research Center, MIT, Cambridge, MA, USA
huchette@mit.edu

Miles Lubin, Operations Research Center, MIT, Cambridge, MA, USA
miles.lubin@gmail.com

References

[1] M. Lubin and I. Dunning. Computing in Operations Research using Julia. *INFORMS J Comput*, 27(2):238–248, 2015.
[2] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Rev*, 59(2):295–320, 2017.

Stuart Mitchell

PuLP

PuLP [1] is a library for Python [2] that enables users to describe mathematical programs. PuLP works entirely within the syntax and natural idioms of Python by providing objects that represent optimization problems and decision variables, and allowing constraints to be expressed in a way that is very similar to the original mathematical expression. To keep the syntax as simple and intuitive as possible, PuLP has focused on supporting linear and mixed-integer models.

PuLP can easily be deployed on any system that has a Python interpreter, as it has no dependencies on any other software packages. It supports a wide range of both commercial and open-source solvers, and can be easily extended to support additional solvers. Finally, it is available under a permissive open-source license that encourages and facilitates the use of PuLP inside other projects that need linear optimization capabilities.

1 History

PuLP was originally written by Jean-Sebastien Roy [3]. Unfortunately, Jean-Sebastien Roy passed away in 2008 and Stuart Mitchell took over the development and maintenance of the project. Documentation for PuLP [4] was developed within the Department of Engineering Science at the University of Auckland between 2008 and 2011.

PuLP is part of Coin-or [5]; development and issues are tracked on github [6]; it is distributed on pypi [7]; discussions and comments are recieved via the mailing list [8] and stackoverflow [9].

2 Installation

PuLP is very easy to install. The provision of a setup.py file and registration on pipy, allows the user to use the command `$easy_install pulp-or` to download and install the latest version of PuLP on their system. For Windows, Linux and OSX users this package also includes the coin-or [5] solver so PuLP will be immediately functional. For users on other platforms a compatible solver must be installed for a PuLP model to be solved.

3 Design and features of PuLP

Several factors were considered in the design of PuLP and in the selection of Python as the language to use.

3.1 Free, open source, portable

It was desirable that PuLP be usable anywhere, whether it was as a straight forward modelling and experimentation tool, or as part of a larger industrial application. This required that PuLP be easily licensed, and adaptable to different hardware and software environments. Python itself more than meets these requirements: it has a permissive open-source license and has implementations available at no cost for a wide variety of platforms, both conventional and exotic. PuLP builds on these strengths by also being free and licensed under the very permissive MIT License[11]. It is written in pure Python code, creating no new dependencies that may inhibit distribution or implementation.

3.2 Interfacing with solvers

Many mixed-integer linear programming (MILP) solvers are available, both commercial (e.g., CPLEX [10], Gurobi [11]) and open-source (e.g., CBC [5]). PuLP takes a modular approach to solvers by handling the conversion of Python-PuLP expressions into sparse matrix and vector representations of the model internally, and then exposing this data to a solver interface class. As the interface to many solvers is similar, or can be handled by writing the model to the standard ‘LP’ or ‘MPS’ file formats, base generic solver classes are included with PuLP in addition to specific interfaces to the currently popular solvers. These generic solver classes can then be extended by users or the developers of new solvers with minimal effort.

3.3 Syntax, simplicity, style

A formalised style of writing Python code [12], that is referred to as “Pythonic” code, has developed over the past 20 years of Python development. This style is well established and focuses on readability and maintainability of code over “clever” manipulations that are more terse but are considered harmful to the maintainability of software projects.

PuLP builds on this style by using the natural idioms of Python programming wherever possible. It does this by having very few special functions or “keywords”, to avoid polluting the namespace of the language. Instead it provides two main objects (for a problem and for a variable) and then uses Python’s control structures and arithmetic operators.

In contrast to Pyomo [13], another Python-based modelling language, PuLP does not allow users to create purely abstract models. While in a theoretical sense this restricts the user, we believe that abstract model construction is not needed for a large number of approaches in dynamic, flexible modern languages like Python. These languages do not distinguish between data or functions until the code is run, allowing users to still construct complex models in a pseudo-abstract fashion.

4 Sudoku example

A sudoku problem is solved by the following file [14] please read the code comments for further explanation.

```
# Import PuLP module functions
from pulp import *

# A list of strings from "1" to "9" is created
# This allows us to represent the value of each
# sudoku square with a binary variable
Sequence = ["1", "2", "3", "4", "5", "6", "7",
               "8", "9"]

# The Vals, Rows and Cols lists all follow this form
Vals = Sequence
Rows = Sequence
Cols = Sequence

# The prob variable is created to contain all the
# problem data
prob = LpProblem("Sudoku Problem",LpMinimize)

# The problem variables are created
# as a python dictionary defining the position
# and the value of each sudoku square
# as a binary variable
choices = LpVariable.dicts(
          "Choice",(Vals,Rows,Cols),0,1,LpInteger)

# The arbitrary objective function is added
prob += 0, "Arbitrary Objective Function"

# A constraint ensuring that only one value 1-9
# can be in each square
# PuLP expresses constraints with the use of
# python list comprehensions
for r in Rows:
    for c in Cols:
        prob += lpSum([choices[v][r][c]
                        for v in Vals]) == 1

# The row, column constraints are added for each value
for v in Vals:
    for r in Rows:
        prob += lpSum([choices[v][r][c] for c in Cols]
                       ) == 1

    for c in Cols:
        prob += lpSum([choices[v][r][c] for r in Rows]
                       ) == 1

# create the 3x3 boxes constraint

# The boxes list is created, with the row and
# column index of each square in each box
Boxes =[]
for i in range(3):
    for j in range(3):
        # this constraint creates 9 3x3 boxes
        Boxes += [[(Rows[3*i+k],Cols[3*j+l])
                    for k in range(3)
                    for l in range(3)]]

# Then a constraint is created that only allows a
# value to exist once in each 3x3 box
for v in Vals:
    for b in Boxes:
        prob += lpSum([choices[v][r][c] for (r,c) in b]
                       ) == 1

# after this formulation the initial numbers can be
# entered as constraints e.g.
prob += choices["5"]["1"]["1"] == 1

# Then the problem is solved using the default solver
prob.solve()

# details on how to print out the solution can be found
# the original file
```

Stuart Mitchell, Stuart Mitchell Consulting, Auckland, New Zealand
pulp@stuartmitchell.com

References

[1] S. Mitchell, M. O'Sullivan, and I. Dunning, "Pulp: A linear programming toolkit for python," 2011. [Online]. www.optimization-online.org/DB_FILE/2011/09/3178.pdf
[2] P. S. Foundation. Python programming language. [Online]. www.python.org
[3] Contributions of jean-sébastien roy. [Online]. js2007.free.fr
[4] S. Mitchell, A. Kean, A. Mason, M. O'Sullivan, and A. Phillips. Pulp documentation. [Online]. pythonhosted.org/PuLP/
[5] R. Lougee-Heimer, "The common optimization interface for operations research," *IBM Journal of Research and Development*, vol. 47, no. 1, pp. 57–66, January 2003.
[6] Github. [Online]. http://github.com
[7] Pypi – the python package index. [Online]. pypi.python.org/pypi
[8] Pulp – email discussion group. [Online]. https://groups.google.com/forum/#!forum/pulp-or-discuss
[9] Stack overflow – items tagged pulp. [Online]. https://stackoverflow.com/questions/tagged/pulp
[10] Cplex website. [Online]. www.ilog.com/products/cplex/
[11] Gurobi website. [Online]. www.gurobi.com
[12] G. van Rossum, B. Warsaw, and N. Coghlan. Pep 8 – style guide for python code. [Online]. www.python.org/dev/peps/pep-0008/
[13] W. Hart, "Python optimization modeling objects (pyomo)," in *Proc INFORMS Computing Society Conference*, 2009. [Online]. www.optimization-online.org/DB_HTML/2008/09/2095.html
[14] Pulp sudoku example. [Online]. github.com/coin-or/pulp/blob/master/examples/Sudoku1.py

Steven Diamond and Stephen Boyd

Convex Optimization in Python with CVXPY

CVXPY [1] began in 2013 as a Python implementation of CVX [2], a widely used modeling language in MATLAB for convex optimization. The original motivation for a Python implementation was to drop the dependence on MATLAB in order to have an entirely open-source framework and to make convex optimization accessible in an increasingly popular language for numerical and scientific computing. Since its development CVXPY has seen extensive adoption, being used in university classes, in dozens of research projects, and at major companies in a variety of industries. CVXPY is available for Python 2 and 3 on all platforms at www.cvxpy.org. It supports the solvers ECOS [3], SCS [4], CVXOPT [5], GLPK [6], Cbc [7], Elemental [8], GUROBI [9], MOSEK [10], and Xpress [11].

The high-level, object-oriented features available in Python have made it easy to build packages that extend CVXPY. Several extensions add functionality for modeling and (heuristically) solving special classes of nonconvex optimization problems, such as difference-of-convex [12], multi-convex [13], and others [14]. Another set of extensions specialize CVXPY's optimization modeling framework to a particular field, such as finance [15], computational imaging [16], and dynamic energy management [17]. The two types of extensions complement each other, as new support for nonconvex problems enables applications from new domains.

The functionality of CVXPY is based on CVX, but the implementation follows a different philosophy that emphasizes the symbolic representation of a problem via expression trees over a conventional sparse matrix representation. The conversion of the problem into standard form is entirely symbolic, with an explicit sparse matrix representation of the standard form only constructed as the last step before calling the solver. Our focus on symbolic problem representations when implementing CVXPY led to research on matrix-free convex modeling and optimization [18] and integrating convex solvers and modeling languages with modern deep learning frameworks [19].

CVXPY's symbolic processing of problems made it easy to add support for parameters, or constants with fixed symbolic attributes but numeric value unknown until solve time. Parameters are useful for simplifying code and caching work when solving a series of similar problems. Another feature CVXPY added to those existing in CVX was incorporating sign information into disciplined convex programming (DCP) [20], a system for analyzing the convexity properties of user defined problems. Sign-aware DCP was the first of many variants of DCP developed for CVXPY and its extensions.

As a concrete example of CVXPY syntax and features, consider the Lasso problem

$$\text{minimize } \|Ax - b\|^2 + \gamma\|x\|_1$$

with optimization variable $x \in \mathbf{R}^n$, problem data $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$, and parameter $\gamma > 0$. We express the Lasso problem in CVXPY as follows:

```
x = Variable(n)
fit = sum_squares(A*x - b)
prob = Problem(Minimize(fit + gamma*norm(x, 1)))
# The optimal objective is returned by prob.solve().
result = prob.solve()
# The optimal value for x is stored in x.value.
print x.value
```

Here `A` and `b` are NumPy ndarrays, the primary matrix representation in Python, while `gamma` is a Python float.

If we instead define `gamma` as a parameter, we can vary the value of `gamma` to efficiently construct a trade-off curve, warm-starting with the previous solution. We specify the sign of `gamma` as positive so that CVXPY can verify the problem is convex.

```
gamma = Parameter(sign='positive')
...
# Iterate over values in [10^-2, 10^2].
for val in logspace(-2, 2):
  gamma.value = val
  prob.solve(warm_start=True)
  # Save the solution x.value.
  ...
```

CVXPY has evolved from its initial conception as an implementation of CVX in Python to become the center of a diverse ecosystem of optimization packages, which treat CVXPY as a black box for modeling and solving convex problems. The vision for CVXPY going forward is to open the black box and expose the process of converting problems into standard form to new contributions. We will express the conversion into standard form as a series of reductions, each taking a CVXPY problem as input and giving as output an equivalent (but transformed) CVXPY problem. Adding new reductions and other modules that process transformed problems, such as presolvers and code generation, will be as easy as writing a CVXPY extension. We also envision more solvers that take a CVXPY problem object as canonical input (e.g., [19, 21]).

Steven Diamond, Department of Computer Science, Stanford University, Stanford, CA, USA. diamond@cs.stanford.edu

Stephen Boyd, Department of Electrical Engineering, Stanford University, Stanford, CA, USA. boyd@stanford.edu

References

[1] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[2] M. Grant and S. Boyd, "CVX: MATLAB software for disciplined convex programming, version 2.1." http://cvxr.com/cvx, Mar. 2014.

[3] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proceedings of the European Control Conference*, pp. 3071–3076, 2013.

[4] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, 2016.

[5] M. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT: Python software for convex optimization." http://cvxopt.org/.

[6] A. Makhorin, "GLPK – GNU linear programming kit." www.gnu.org/software/glpk/.

[7] J. Forrest, "Cbc – COIN-OR branch and cut." https://projects.coin-or.org/Cbc.

[8] J. Poulson, "Elemental." http://libelemental.org/.

[9] I. Gurobi Optimization, "Gurobi Optimizer Reference Manual." www.gurobi.com, 2016.

[10] "MOSEK optimization software." https://mosek.com/.

[11] "FICO Xpress optimization." www.fico.com/en/products/fico-xpress-optimization.

[12] X. Shen, S. Diamond, Y. Gu, and S. Boyd, "Disciplined convex-concave programming," in *Proceedings of the IEEE Conference on Decision and Control*, 2016.

[13] X. Shen, S. Diamond, M. Udell, Y. Gu, and S. Boyd, "Disciplined multi-convex programming," in *Proceedings of the Chinese Conference on Decision and Control*, 2016.

[14] S. Diamond, R. Takapoui, and S. Boyd, "A general system for heuristic minimization of convex objectives over nonconvex sets," *To appear, Optimization Methods and Software*, 2017.

[15] S. Boyd, E. Busseti, S. Diamond, R. Kahn, K. Koh, P. Nystrup, and J. Speth, "Multi-period trading via convex optimization," *To appear, Foundations and Trends in Optimization*, 2017.

[16] F. Heide, S. Diamond, M. Niessner, J. Ragan-Kelley, W. Heidrich, and G. Wetzstein, "ProxImaL: Efficient image optimization using proximal algorithms," in *Proceedings of ACM SIGGRAPH*, vol. 35, pp. 1–15, 2016.

[17] M. Wytock, N. Moehle, and S. Boyd, "Dynamic energy management with scenario-based robust MPC," in *Proceedings of the American Control Conference*, pp. 2042–2047, 2017.

[18] S. Diamond and S. Boyd, "Matrix-free convex optimization modeling," in *Optimization and Its Applications in Control and Data Sciences* (B. Goldengorin, ed.), vol. 115 of *Springer Optimization and Its Applications*, pp. 221–264, Springer, 2016.

[19] M. Wytock, S. Diamond, F. Heide, and S. Boyd, "A new architecture for optimization modeling frameworks," in *Proceedings of the Workshop on Python for High-Performance and Scientific Computing*, pp. 36–44, 2016.

[20] M. Grant, *Disciplined Convex Programming*. PhD thesis, Stanford University, 2004.

[21] P.-W. Wang, M. Wytock, and Z. Kolter, "Epigraph projections for fast general convex programming," in *Proceedings of the International Conference on Machine Learning*, pp. 2868–2877, 2016.

Dirk Schumacher

Mixed integer linear programming in R with *ompr*

R [2] is a popular language and computational environment among statisticians. In recent years R has gained a lot of popularity in other disciplines as well. With over 11,000 packages on CRAN [4], the R package repository, there is a package for almost every problem. And of course, packages for solving mixed integer linear programs are also available. R as a language makes it easy to create domain specific languages through metaprogramming using so-called non-standard evaluation [1].

ompr (the optimization modelling package) [3] is an attempt to develop an algebraic modelling language within R. The goal is to create a modelling API that uses idiomatic R and integrates well with other popular packages, such as *dplyr* for data processing. *ompr* models are solver-independent and currently support mixed integer linear programming problems. It works on all platforms supported by R (Linux, MacOS and Windows). Compared to other modelling languages it comes closest to *JuMP* [6] for *julia* but is currently more limited in terms of features and scope. In fact *JuMP* inspired me to build *ompr*.

The features of *ompr* are best described by modelling a mixed integer linear program in R. As an example for this article we model a *warehouse location problem* where we would like to find the cost optimal location and number of warehouses and the assignment of customers to those hubs.

$$
\begin{array}{lll}
\text{min} & \displaystyle\sum_{i=1}^{n}\sum_{j=1}^{m} \mathrm{tc}_{i,j}\cdot x_{i,j} & +\displaystyle\sum_{j=1}^{m} \mathrm{fc}_j \cdot y_j \\
\text{subject to} & \displaystyle\sum_{j=1}^{m} x_{i,j} = 1 & \forall\, i = 1,\dots,n \\
 & x_{i,j} \le y_j, & \forall\, i = 1,\dots,n,\ j = 1,\dots,m \\
 & x_{i,j} \in \{0,1\} & \forall\, i = 1,\dots,n,\ j = 1,\dots,m \\
 & y_j \in \{0,1\} & \forall\, j = 1,\dots,m
\end{array}
$$

Figure 1. *A warehouse location problem as an illustrative example. n customers need to be assigned to exactly one of at most m warehouses. Setting up a warehouse $y_j = 1$ has a fixed cost of $\mathrm{fc}_j > 0$ and assigning a customer to a warehouse $x_{i,j} = 1$ leads to travel cost of $\mathrm{tc}_{i,j} > 0$.*

Every *ompr* model starts with an empty model to which necessary elements (like variables, constraints and an objective function) are added. Expressions can be mixed with variables defined in the general environment. For example in the model below the functions tc and fc are not part of the optimization problem, but are regular R functions defined in the environment. Like other algebraic domain specific languages, equations can be written down directly and the package takes care of transforming these to the required format for the API of the solver.

```
library(ompr)
model <- MIPModel() %>%
  add_variable(x[i, j], i=1:n, j=1:m, type="binary") %>%
  add_variable(y[j], j=1:m, type="binary") %>%
  set_objective(
    sum_expr(tc(i, j) * x[i, j], i=1:n, j=1:m) +
    sum_expr(fc(j) * y[j], j=1:m),
    sense = "min") %>%
  add_constraint(sum_expr(x[i, j], j=1:m) == 1, i=1:n) %>%
  add_constraint(x[i, j] <= y[j], i=1:n, j=1:m)
```

Figure 2. *The warehouse location problem* modeled with ompr *using so called pipes to chain together successive functions*

```
library(ompr.roi)
library(ROI.plugin.glpk)
result <- solve_model(model, with_ROI(solver = "glpk"))
assignment <- get_solution(result, x[i, j])
warehouses <- get_solution(result, y[i])
```

Figure 3. *The model defined earlier is solved with the GNU Linear Programming Kit using the solver package* ompr.roi *which offers access to a variety of open source and commercial solvers*

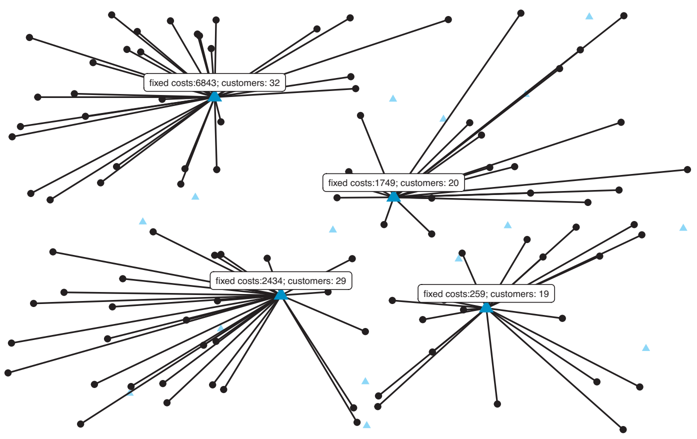

Figure 4. *A solution to the warehouse location problem generated by* ggplot2 *and* ompr

After an *ompr* model is built, it can be passed to a solver. Within *ompr* a solver is simply a function mapping a model to a solution. In this case the GNU Linear Programming Kit [7] is used to find an optimal solution. To support most popular solvers out of the box, the package *ompr.roi* can be used. It makes it possible to use any solver supported by the *R optimization infrastructure* (*ROI*) packages [5]. *ROI* gives access to many open source (*CLP*, *GLPK*, *lpsolve*, *Symphony*) and commercial solvers (*CPLEX*, *Gurobi*) through a standardized interface.

The result can then be further processed within the R ecosystem. For example, using R's plotting facilities, we can plot the assignment (Figure 4).

Currently *ompr*'s cost of abstraction can be rather high for larger models, but this is something I plan to work on in the future. Up to now the focus has been on creating an idiomatic API for modelling mixed integer linear programs directly in R. *ompr* as well as *ompr.roi* are published on CRAN and developed on GitHub. The packages are open source and I encourage anyone to send feedback, ideas or code contributions. On the project's website [3] I have compiled some articles showcasing the modelling features of the package and tutorials on how to model selected optimization problems in R with *ompr*.

In my opinion a big advantage of modelling optimization problems in R is the ability to easily utilize other packages within the R ecosystem for data wrangling, statistics, (interactive) visualizations and reproducible research. For example, this article was generated in a completely reproducible manner with *knitr* [8] and the code shown here is the actual code that led to the output in Figure 4; it is published on GitHub (https://github.com/dirkschumacher/ompr-optima) as well.

Dirk Schumacher, Berlin, Germany. mail@dirk-schumacher.net

References

[1] H. Wickham. Advanced R. CRC Press, 2014.
[2] R Development Core Team. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL https://www.R-project.org, 2008.
[3] D. Schumacher. ompr: Optimization modelling package. R package version 0.6, https://cran.r-project.org/package=ompr, 2017.
[4] CRAN: The Comprehensive R Archive Network. https://cran.r-project.org/, 2017-07-16.
[5] K. Hornik, D. Meyer, F. Schwendinger, and S. Theussl. ROI: R Optimization Infrastructure. R package version 0.2-1, https://cran.r-project.org/package=ROI, 2016.
[6] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. SIAM Rev, 59(2): 295–320, 2017.
[7] GNU Linear Programming Kit. http://www.gnu.org/software/glpk/glpk.html, 2017.
[8] Y. Xie. knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.1, 2017.

Thorsten Koch

The ZIMPL modeling language

The Zuse Institute Mathematical Programming Language (ZIMPL) [8] is a small tool to describe optimization problems and then translate these descriptions into files that solvers can read and process. ZIMPL generates LP and MPS files that can be read directly by all major LP and MIP solvers.

Its development started in 1999 when we failed to purchase a commercial modeling language from a then well-known vendor of optimization software. At that point, it became obvious that there was no really useable open-source modeling language available. After reading the inspiring AMPL book [4] what we envisioned was a modeling language that is solver independent, is quick and easy to learn, can be used for lectures as well as for industry projects, is freely available with source code, and is highly portable between computer systems.

What makes ZIMPL special is the use of rational arithmetic. With a few noted exceptions, all computations in ZIMPL are done with infinite precision rational arithmetic, ensuring that no rounding errors can occur during modeling. This is achieved by employing the GNU Multi Precision Library [18]. One reason for this is our ongoing interest in exact solvers [1, 5, 7]. There is little point in having an exact LP or MIP solver if the rounding errors are already introduced when generating the instance.

Another special feature is an automatic conversion of functions with decision variables as arguments into a system of inequalities. The arguments of these functions have to be linear terms consisting of bounded integer or binary variables. For example, it is possible to have constraints dependent on the value of a variable or compute the absolute value of a variable automatically.

ZIMPL also supports polynomial terms, but of course, the solver used has to support this. In connection with SCIP, it is now possible to solve non-convex MINLPs [6].

There are in general two trends in modeling languages: One direction is to further integrate features like database query tools, solvers, report generators, and graphical user interfaces, see, e.g., [12, 20]. This sometimes even allows to build complete graphical (business) applications around the mathematical model. The other direction is to use a general programming language, e.g., Python [19, 24] and the the previous articles, and extend it with special commands for generating models and executing solvers.

ZIMPL is purposely not intended to go either way. As the user manual is just about 25 pages, you can learn ZIMPL within a few hours, and the models are still very near to the mathematical notation. However, as the source is available, adding for example a database connection if needed, is reasonably easy to do.

According to [2], modeling languages can be separated into two classes, namely the *independent modeling languages*, like, e.g., [13, 23], which do not rely on a specific solver, and the *solver modeling languages*, like, e.g., [21, 22], which are deeply integrated with a specific solver. In exchange for having a better integration with the solvers, it is no longer possible to easily switch between solvers as with independent languages. Given that the performance of a particular solver varies highly with the particular model used, see, e.g. [10], we believe that the ability to switch between different solvers is one of the more useful features of a modeling language.

During the development of ZIMPL special focus has been placed on software engineering. ZIMPL comes together with a test suite that when executed covers more than 80 % of the program code. This assures that changes and new features do not break existing functionality. Assert statements are used extensively in the code to test preconditions and invariants. The code is regularly run through a suite of dynamic and static program checkers, like Valgrind [25], Flexelint [17], CPP-check [16], Clang-Analyzer [14].

Each ZIMPL model consists of six types of statements: Sets, Parameters, Variables, Objective, Constraints, and Function definitions. Below you can see the ZIMPL code for a model that will check whether a given solution of a Sudoku puzzle [9] is unique. In general, Sudokus are supposed to have only one solution. The model will take a file with the prefixed numbers of the Sudoku and a file with the known solution. If the generated instance is run through a solver it should come out *integer infeasible* – otherwise there is another solution.

```
# Model to check for uniqueness of Sudoku solution.
#
param p := 3;
set K := { 1 .. p*p };
set M := { 1 .. p};

var x[K * K * K] binary;

# File format: row col value
set F := { read "prefixed.dat" as "<1n,2n,3n>" };
set S := { read "solution.dat" as "<1n,2n,3n>" };

subto per_field:
   forall <i,j> in K*K do sum <k> in K : x[i,j,k] == 1;
subto per_column:
   forall <j,k> in K*K do sum <i> in K : x[i,j,k] == 1;
subto per_row:
   forall <i,k> in K*K do sum <j> in K : x[i,j,k] == 1;

subto subsquare: forall <m,n,k> in M*M*K do
   sum <i,j> in M*M : x[(m-1)*p+i,(n-1)*p+j,k] == 1;

# Fix the prefixed entries in the puzzle
subto prefixed:
   forall <i,j,k> in F do x[i,j,k] == 1;

# Forbid the existing solution
subto forbid_old:
   sum <i,j,k> in S : x[i,j,k] <= card(K*K) - 1;
```

It is important to note that ZIMPL statements never change the already existing part of the model but only add to it. This makes it easier to understand ZIMPL models. ZIMPL has been (and is being) used to model many real-world [3] and educational questions, as diverse as location planning in telecommunications, 3D-Steiner tree packing for chip design, track auctioning, protein folding, to give hands-on lectures at many universities, and courses like Combinatorial Optimization at Work [15].

In the future we plan to explore the following topics:

▷ Parallel execution. As it is straightforward to compute the dependency of the individual lines of ZIMPL programs between each other, we want to exploit the availability of multiple cores.
▷ Having more general constraint types, e.g., an all-different constraint, which allow modeling on a higher level. ZIMPL should then transform this differently depending on the model size or target solver.
▷ Allow supplying extra information to the model, e.g., about symmetry or implicit integer variables.

ZIMPL can be downloaded as source code or pre-compiled binary at http://zimpl.zib.de and as part of the SCIP Optimization Suite [11] at http://scip.zib.de. Enjoy!

Thorsten Koch, Zuse Institute Berlin (ZIB) and TU Berlin, Germany.
koch@zib.de

References

[1] W. Cook, T. Koch, D.E. Steffy, K. Wolter *A hybrid branch-and-bound approach for exact rational mixed-integer programming*. *Mathematical Programming Computation*, 5(3):305–344, 2013.
[2] K. Cunningham, L. Schrage. *The LINGO Algebraic Modeling Language*. *Modeling Languages in Mathematical Optimization*, Kluwer, 2004.
[3] U. Dorndorf, S. Droste, T. Koch. *Using ZIMPL for Modeling Production Planning Problems*. in *Algebraic Modeling Systems*, Springer, 2012.
[4] R. Fourer, D.M. Gay, B.W. Kernighan. *AMPL: A Modelling Language for Mathematical Programming*. 2nd edition, Brooks/Cole—Thomson Learning, 2003
[5] A.M. Gleixner, D.E. Steffy, K. Wolter. *Iterative Refinement for Linear Programming*. *INFORMS Journal on Computing*, 28(3):449–464, 2016
[6] S. Vigerske, A.M. Gleixner. *Global Optimization of Mixed-Integer Nonlinear Programs in a Branch-and-Cut Framework*. *Optimization Methods and Software*, 2017, doi:10.1080/10556788.2017.1335312
[7] T. Koch *The final NETLIB-LP results*. *Operations Research Letters 32(2):138–142, 2004*, .
[8] T. Koch. *Rapid Mathematial Programming*. PhD-thesis, TU-Berlin, 2004, http://nbn-resolving.de/urn:nbn:de:0297-zib-8346
[9] T. Koch *Rapid Mathematical Programming or How to Solve Sudoku Puzzles in a few Seconds* . *Operations Research Proceedings 2005*. Selected papers of the Annual Int. Conf. of the German Operations Research Society, Bremen, September 7–9, 2005, pp 21–27, 2006.
[10] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R.E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, K. Wolter. *MIPLIB 2010*. *Mathematical Programming Computation* 3(2):103–163, 2011.
[11] S.J. Maher, T. Fischer, T. Gally, G. Gamrath, A. Gleixner, R. L. Gottwald, G. Hendel, T. Koch, M. E. Lübbecke, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, S. Schenker, R. Schwarz, F. Serrano, Y. Shinano, D. Weninger, J. T. Witt, J. Witzig. *The SCIP Optimization Suite 4.0*. Techical Report ZR-17-12, ZIB, 2017, http://nbn-resolving.de/urn:nbn:de:0297-zib-62170.
[12] https://aimms.com
[13] http://ampl.com
[14] https://clang-analyzer.llvm.org
[15] http://co-at-work.zib.de
[16] http://cppcheck.sourceforge.net
[17] http://www.gimpel.com
[18] http://gmplib.org
[19] http://www.gurobi.com/resources/seminars-and-videos/modeling-with-the-gurobi-python-interface
[20] https://litic.com
[21] http://www.fico.com/de/products/fico-xpress-optimization-suite
[22] https://www-01.ibm.com/software/commerce/optimization/modeling
[23] https://www.gams.com
[24] http://www.pyomo.org
[25] http://valgrind.org

Calls for Nominations

The following prizes will be presented or announced at the *23nd International Symposium on Mathematical Programming* (ISMP 2018) to take place in Bordeaux, France, July 1–6, 2018.

Beale–Orchard-Hays Prize

Nominations are invited for the 2018 Beale–Orchard-Hays Prize for Excellence in Computational Mathematical Programming.

The Prize is sponsored by the Mathematical Optimization Society, in memory of Martin Beale and William Orchard-Hays, pioneers in computational mathematical programming. Nominated works must have been published between Jan 1, 2012 and Dec 31, 2017, and demonstrate excellence in any aspect of computational mathematical programming. "Computational mathematical programming" includes the development of high-quality mathematical programming algorithms and software, the experimental evaluation of mathematical programming algorithms, and the development of new methods for the empirical testing of mathematical programming techniques.

Full details of prize rules and eligibility requirements can be found at www.mathopt.org/?nav=boh.

The members of the Prize committee are Michael Grant (Chair) (CVX Research), Tobias Achterberg (Gurobi Optimization), Jeff Linderoth (University of Wisconsin), Petra Mutzel (University of Dortmund), and Ted Ralphs (Lehigh University).

Nominations can be submitted electronically or in writing, and should include detailed publication details of the nominated work. Electronic submissions should include an attachment with the final published version of the nominated work. If done in writing, submissions should include five copies of the nominated work. Supporting justification and any supplementary material are strongly encouraged but not mandatory. The prize committee reserves the right to request further supporting material and justification from the nominees.

The deadline for nominations is January 15, 2018. Nominations should be submitted to Dr. Michael Grant, mcg@cvxr.com. If you wish to submit a nomination in writing, please contact Dr. Grant for a mailing address.

Dantzig Prize

Nominations are solicited for the 2018 George B. Dantzig Prize, administered jointly by the Mathematical Optimization Society (MOS) and the Society for Industrial and Applied Mathematics (SIAM).

The Dantzig prize is awarded to one or more individuals for original research which by its originality, breadth and depth, is having a major impact on the field of mathematical optimization. The contribution(s) for which the award is made must be publicly available and may belong to any aspect of mathematical optimization in its broadest sense. The award will include a certificate containing the citation and a cash prize.

The members of the 2018 Dantzig Prize committee are: Gérard Cornuejols (Carnegie-Mellon University – Chair), Monique Laurent (CWI Amsterdam and Tilburg University), Jorge Nocedal (Northwestern University), and Michael Overton (New York Univerity).

Nominations should consist of a letter describing the nominee's qualifications for the prize, and a current curriculum vitae of the nominee including a list of publications. Nominations and questions regarding the nomination process should be sent to Professor Gérard Cornuejols, gc0v@andrew.cmu.edu The deadline for nominations is November 30, 2017.

More information on the prize, including past winners, is available at www.siam.org/prizes/sponsored/dantzig.php and www.mathopt.org/?nav=dantzig.

Fulkerson Prize

The Fulkerson Prize Committee invites nominations for the Delbert Ray Fulkerson Prize, sponsored jointly by the Mathematical Optimization Society (MOS) and the American Mathematical Society (AMS). Up to three awards of US$ 1,500 each are presented at each (triennial) International Symposium of the MOS. The Fulkerson Prize is for outstanding papers in the area of discrete mathematics.

Eligible papers should represent the final publication of the main result(s) and should have been published in a recognized journal or in a comparable, well-refereed volume intended to publish final publications only, during the six calendar years preceding the year of the Symposium (thus, from January 2012 through December 2017). The prizes will be given for single papers, not series of papers or books, and in the event of joint authorship the prize will be divided.

The term "discrete mathematics" is interpreted broadly and is intended to include graph theory, networks, mathematical programming, applied combinatorics, applications of discrete mathematics to computer science, and related subjects. While research work in these areas is usually not far removed from practical applications, the judging of papers will be based only on their mathematical quality and significance.

Previous winners of the Fulkerson Prize are listed at http://www.mathopt.org/?nav=fulkerson#winners. Further information about the Fulkerson Prize can be found at: tinyurl.com/y7j87okq and tinyurl.com/yabnp9mb.

The Fulkerson Prize Committee consists of Maria Chudnovsky (Princeton University, USA), Friedrich Eisenbrand (EPFL, Lausanne, chair), and Martin Grötschel (Berlin-Brandenburgische Akademie der Wissenschaften, Berlin, Germany).

Please send your nominations (including reference to the nominated article and an evaluation of the work) by February 15th, 2018 to the chair of the committee, Professor Friedrich Eisenbrand, EPFL, Station 8, 1015 Lausanne, Switzerland, friedrich.eisenbrand@epfl.ch. Electronic submissions are preferred.

Lagrange Prize

Nominations are solicited for the 2018 Lagrange Prize in Continuous Optimization, administered jointly by the Mathematical Optimization Society (MOS) and the Society for Industrial and Applied Mathematics (SIAM).

The Lagrange Prize, established in 2002, is awarded for an outstanding contribution in the area of continuous optimization. The work will be judged on its mathematical quality, significance, and originality. Clarity of the exposition and the value of the work in practical applications may be considered as secondary attributes.

The work to be considered should form the final publication of the main result(s) and should have been published between the years of 2012 to 2017 as an article in a recognized journal, a comparable, well-referenced volume intended to publish final publications only, or a monograph consisting chiefly of original results rather than previously published material. Extended abstracts and prepublications, and articles in published journals, journal sections or proceedings that are intended to publish non-final papers, are not eligible.

The award will include a certificate containing the citation and a cash prize of US$ 1,500. In the event of joint authorship, the prize will be divided. More information on the prize, including past winners, is available at: www.siam.org/prizes/sponsored/lagrange.php and www.mathopt.org/?nav=lagrange

The members of the 2018 Lagrange Prize committee are Philip E. Gill (University of California, San Diego), Andreas Griewank (Yachay University, Ecuador), Etienne de Klerk (Tilburg University and Delft

University of Technology), and Katya Scheinberg (Chair) (Lehigh University).

Nominations should include a letter evaluating the contribution(s) of the work and citing the works to be considered. Nominations and questions regarding the nomination process should be sent to Professor Katya Scheinberg, katyas@lehigh.edu. The deadline for nominations is November 30, 2017.

Tseng Lectureship

The Mathematical Optimization Society invites nominations for the Paul Y. Tseng Memorial Lectureship in Continuous Optimization. This prize was established in 2011 and will be presented for the third time at the International Symposium on Mathematical Programming 2018. The lectureship was established on the initiative of family and friends of Professor Tseng, with financial contributions to the endowment also from universities and companies in the Asia-Pacific region. The purposes of the lectureship are to commemorate the outstanding contributions of Professor Tseng in continuous optimization and to promote the research and applications of continuous optimization in the Asia-Pacific region.

The lectureship is awarded to an individual for outstanding contributions in the area of continuous optimization, consisting of original theoretical results, innovative applications, or successful software development. The primary consideration in the selection process is the quality and impact of the candidate's work in continuous optimization. See more details at www.mathopt.org/?nav=tseng.

The members of the 2018 Paul Y. Tseng Memorial Lectureship committee are Yu-Hong Dai (AMSS, Chinese Academy of Sciences), Luis Nunes Vicente (University of Coimbra), Ya-xiang Yuan (AMSS, Chinese Academy of Sciences), and Shuzhong Zhang (Chair) (University of Minnesota).

The nomination must include a nomination letter of no more than two pages and a short CV of the candidate (no more than two pages, including selected publications). In addition, the nominator should also arrange for 1–2 letters of recommendation. All nomination materials should be sent (preferably in electronic form, as pdf documents) to Professor Shuzhong Zhang, Department of Industrial & Systems Engineering, University of Minnesota, 111 Church Street S.E., Minneapolis, MN 55455, USA, zhangs@umn.edu. All nomination materials must be received by December 31, 2017.

Tucker Prize

The Mathematical Optimization Society solicits nominations for the 2018 A. W. Tucker Prize, which will be awarded for an outstanding doctoral thesis in any area of mathematical optimization. The Tucker Prize Committee will screen the nominations and select at most three finalists. The finalists will be invited to give oral presentations of their work at a special session of the International Symposium on Mathematical Programming 2018. The Tucker Prize Committee will select the winner before the symposium and present the award prior to the conclusion of the symposium.

The doctoral thesis must have been approved formally (with signatures) by the nominee's thesis committee between January 1, 2015 and January 1, 2018.

The winner will receive an award of US$ 1,000 and a certificate. The other finalists will also receive certificates. The Society will also pay partial travel expenses for each finalist to attend the Symposium. Reimbursements will normally be limited to US$ 750. The nominee's doctoral institution will be encouraged to assist any nominee selected as a finalist with additional travel expense.

The members of the 2018 Tucker Prize committee are Santanu Dey (Georgia Institute of Technology), Simge Küçükyavuz (Chair) (University of Washington), Sven Leyffer (Argonne National Laboratory), Britta Peis (RWTH Achen), and Anke van Zuylen (College of William & Mary).

Nominations must be made by electronic mail to Professor Simge Küçükyavuz, simge@uw.edu.

The nominator must be a faculty member at the institution that awards the nominee's doctoral degree, or a member of the nominee's thesis committee. Applications should consist of the following four pdf files: (a) a letter of nomination; (b) the nominee's thesis; (c) a separate summary of the thesis' contributions, written by the nominee, no more than eight (8) pages in length; and (d) a brief biographical sketch of the nominee. Nominations and the accompanying documentation must be written in English. The Tucker Prize Committee may request additional information. The deadline for nominations is January 15, 2018.

Call for papers

Mathematical Programming Series B: Special Issue on Nonconvex Optimization for Statistical Learning

Motivation and Timeliness.. For many years now, convex optimization has been a principal venue for solving many problems in statistical learning and big-data research. Yet there is increasing evidence supporting the use of nonconvex formulations to enhance the realism of the models and improve their generalizations. Superior results and new advances have occurred in areas such as computational statistics, compressed sensing, imaging science, machine learning, bio-informatics and portfolio selection with the employment of nonconvex functionals to express model loss, promote sparsity, and enhance robustness. In particular, many nonconvex surrogates have been proposed as approximations of the univariate ell0 function that is key to sparsity representation in variable selection. Nonconvex loss functions have also been introduced as minimands in regression and classification. From the perspective of optimization, the study of nonconvex optimization, particularly in the joint presence of nondifferentiability, leads to many challenging open problems, such as a better understanding of various kinds of stationary points for nondifferentiable objectives and their computation, direct (i.e., penalty-free) treatment of nonconvex, nondifferentiable constrained problems and its connection to a penalty approach, statistical inference of stationary (instead of optimal) solutions, optimization of multi-step functions via surrogates with statistical motivations, and applications to problems in advanced sparsity representation with logical conditions, deep learning, truncated regression and classification, and image reconstruction with partial information, to name a few topics of interest.

With the above background, a two-day Conference on Nonconvex Statistical Learning (CNSL) was held May 26–27 2017 at the University of Southern California in Los Angeles. Along with this conference, a special issue of the journal *Mathematical Programming, Series B*, with Jong-Shi Pang, Yufeng, Liu, and Jack Xin as Guest Editors, aims to publish high-quality papers in this emergent area of nonconvex optimization for statistical learning. Besides the invited presentations at CNSL, we encourage researchers in the optimization and statistics community to submit papers that are within the broad domain of topics mentioned above for consideration of publication in this special issue. Of special interest are papers that develop, employ, analyze, and extend optimization models and methods to treat challenging classes of statistical learning problems and their applications.

Paper Submission.. All submissions will be reviewed according to the high standards of the Mathematical Programming journals. Manuscripts should be submitted using the Mathematical Programming style files on ftp.springer.de/pub/tex/latex/svjour3/global.zip for the LaTeX macro package. Authors are strongly advised to keep their papers to a maximum of 25 pages. The deadline for submission is November 01, 2017 with first-round reviews expected to be completed by March 15, 2018 and the volume published before the end of 2018. Authors should submit their manuscripts via www.editorialmanager.com/mapr/ and select Jong-Shi Pang as the handling editor for consideration in this special issue.

Call for site pre-proposals: ISMP 2021

The Symposium Advisory Committee (SAC) of the Mathematical Optimization Society issues a call for pre-proposals to organize and host ISMP 2021, the triennial International Symposium on Mathematical Programming.

ISMP is the flagship event of our society, regularly gathering over a thousand scientists from around the world. The conference is usually held in or around the month of August. Hosting ISMP provides a vital service to the mathematical optimization community and often has a lasting effect on the visibility of the hosting institution. It also presents a significant challenge. This call for pre-proposals is addressed at local groups willing to take up that challenge. The tradition would be that only sites outside of Europe are eligible to host ISMP 2021 (because ISMP 2018 is in Europe).

Preliminary bids will be examined by the Symposium Advisory Committee (SAC), which will then issue invitations for detailed bids. The final decision will be made and announced during ISMP 2018 in Bordeaux. Members of the SAC are Jon Lee (Chair, USA), John Birge (USA), Natashia Boland (USA), Jose Correa (Chile), Satoru Iwata (Japan), Martin Skutella (Germany), Levent Tuncel (Canada), and Yinyu Ye (USA) as well as Karen Aardal (Netherlands, ex officio) and Luis Nunes Vicente (Portugal, ex officio).

Preliminary bids should be brief and contain information pertaining to the location, facilities, logistics: accommodation and transportation, and likely local organizers.

Further information can be obtained from any member of the advisory committee.

Please address your preliminary bids until October 15, 2017 to Jon Lee (jonxlee@umich.edu).

Call for papers

Mathematical Programming Series B: Special Issue on The interface between optimization and probability

Probability theory and the theory of optimization jointly form the theoretical basis of several other fields of research. Important examples are statistics, stochastic optimization, and the theory of risk measures. Additionally, optimization as well as probability theory benefit from each other directly. Examples include the theory of optimal inequalities in probability theory and randomization approaches in optimization such as stochastic gradient descent and the theory of metaheuristics. This special issue aims at attracting state-of-the-art contributions that combine optimization and probability theory in an innovative way and transcend narrow disciplinary thinking. Since the notion of risk is inherently probabilistic and the theory of risk measures is deeply rooted in optimization, the issue puts an emphasis on the measurement and management of risk in a variety of contexts.

Possible topics include but are not limited to: Risk measures in stochastic optimization, Risk measures and distributionally robust optimization, Statistical risk and novel applications of optimization in statistics, Randomized optimization algorithms, Risk and probabilistic aspects in stochastic control and stochastic dynamic optimization, and Financial risk and novel applications of optimization in asset pricing for incomplete markets.

Papers should focus on methodological aspects, although parts of a paper may contain discussions of applications. Moreover, submitted papers should fit into the general scope of Mathematical Programming and will be reviewed according to the standards of of Mathematical Programming, Series A. Due to page limits of the volume, we are requesting that all papers be submitted using MP style files, and conform to a maximum of 25 pages. The deadline for submission of full papers is the 28. 2. 2018. We aim at completing a first review of all submissions by the 30. 6. 2018. Authors should submit their manuscripts via www.editorialmanager.com/mapr/ and select Jong-Shi Pang as the handling editor for consideration in this special issue. Additional information about the special issue can be obtained from the guest editors.

Raimund Kovacevic, Institute of Statistics and Mathematical Methods in Economics, Vienna University of Technology Vienna, Austria.
raimund.kovacevic@tuwien.ac.atraimund.kovacevic@tuwien.ac.at

Roger Wets, Department of Mathematics, University of California, Davis, California, USA. rjbwets@ucdavis.edu

David Wozabal, School of Management, Technical University of Munich, Munich, Germany. david.wozabal@tum.de

Application for Membership

I wish to enroll as a member of the Society. My subscription is for my personal use and not for the benefit of any library or institution.

☐ I will pay my membership dues on receipt of your invoice.
☐ I wish to pay by credit card (Master/Euro or Visa).

Credit card no.	*Expiration date*
Family name	
Mailing address	
Telephone no.	*Telefax no.*
E-mail	
Signature	

Mail to:
Mathematical Optimization Society
3600 Market St, 6th Floor
Philadelphia, PA 19104-2688
USA

Cheques or money orders should be made payable to The Mathematical Optimization Society, Inc. Dues for 2017, including subscription to the journal *Mathematical Programming*, are US $ 90. Retired are $ 45. Student applications: Dues are $ 22.50. Have a faculty member verify your student status and send application with dues to above address.

Faculty verifying status

Institution

IMPRINT
Editor: Volker Kaibel, Institut für Mathematische Optimierung, Otto-von-Guericke Universität Magdeburg, Universitätsplatz 2, 39108 Magdeburg, Germany. kaibel@ovgu.de ■ **Co-Editors:** Samuel Burer, Department of Management Sciences, The University of Iowa, Iowa City, IA 52242-1994, USA. samuel-burer@uiowa.edu ■ Jeff Linderoth, Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI 53706-1572, USA. linderoth@wisc.edu ■ **Founding Editor:** Donald W. Hearn ■ Published by the Mathematical Optimization Society ■ Design and typesetting by Christoph Eyrich, Berlin, Germany. optima@0x45.de ■ Printed by Oktoberdruck AG, Berlin, Germany.