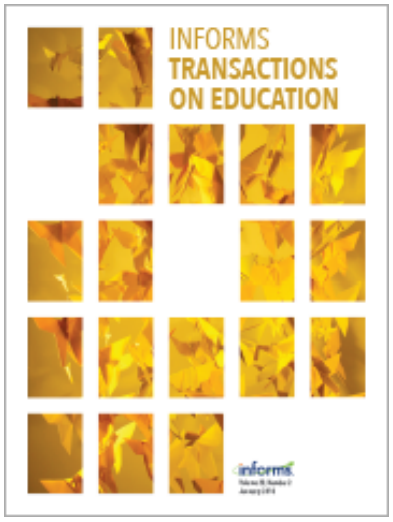


INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

SolverStudio: A New Tool for Better Optimisation and Simulation Modelling in Excel

Andrew J Masonhttp://www.des.auckland.ac.nz/Mason

To cite this article:
Andrew J Masonhttp://www.des.auckland.ac.nz/Mason (2013) SolverStudio: A New Tool for Better Optimisation and Simulation Modelling in Excel. INFORMS Transactions on Education 14(1):45-52. https://doi.org/10.1287/ited.2013.0112

Vol. 14, No. 1, September 2013, pp. 45–52
ISSN 1532-0545 (online)
INFORMS Transactions on Education
http://dx.doi.org/10.1287/ited.2013.0112

SolverStudio: A New Tool for Better Optimisation and Simulation Modelling in Excel

Andrew J. Mason
Department of Engineering Science, University of Auckland, Auckland 1142, New Zealand
{a.mason@auckland.ac.nz, http://www.des.auckland.ac.nz/Mason}

Excel's built-in *Solver* optimiser provides millions of spreadsheet users with easy access to optimisation, and is often used in teaching introductory optimisation courses. However, more sophisticated users often prefer to use modelling languages such as AMPL, GAMS, or PuLP, and so these are often taught in more advanced courses. Unfortunately, the command line and text file interfaces these systems employ present unnecessary barriers to their use. We have developed a new free Excel add-in, SolverStudio, that combines the power of modelling languages—including AMPL, GAMS, PuLP, GMPL, and Gurobi's Python environment—with the familiarity and ease of use of Excel. SolverStudio also brings cloud-based optimisation to Excel by providing easy access to the NEOS online solvers, and simulation capabilities via the Python-based SimPy software. We believe that SolverStudio will support and encourage the greater use in classrooms of Excel-based optimisation and simulation.

Key words: teaching optimisation; spreadsheet optimisation; spreadsheet simulation; open source; solver; AMPL; GAMS; GMPL; PuLP; NEOS
History: Received July 2012; accepted November 2012.

1. Introduction

Because of its ubiquity, Excel is perhaps the most popular modelling platform for teaching the formulation and optimisation of linear, nonlinear, and integer programming models. Every copy of Excel includes the optimisation add-in *Solver* developed for Microsoft by Frontline Systems Inc (Frontline 2011a). Frontline has made a significant contribution in making optimisation so readily available to so many users. Others have also recognised the importance of spreadsheet-based modelling using Excel. These include Lindo Systems Inc, who have developed a competing optimiser known as *What's Best* (Lindo 2012); Oracle, who markets *Crystal Ball* (Oracle 2012), a simulation add-in; and Palisade Corporation, developers of another simulation add-in *@RISK* (Palisade 2012). Frontline also sells advanced Excel products such as the simulation tool *Risk Solver Pro* and the optimiser *Premium Solver Pro* that use an innovative calculation engine to dramatically reduce calculation times. Academics have also developed their own Excel tools. The most well known of these are the freely available Excel add-ins developed by the late Paul Jensen (Jensen 2012) that make an extremely wide variety of optimisation algorithms available to students. LeBlanc and Grossman (2008) and Grossman (2010) provide further details of Excel usage for modelling. However, because of its widespread availability, Excel Solver remains perhaps the most popular tool for spreadsheet optimisation. Unfortunately, the use of Excel and Solver presents a number of problems.

1. Solver has size limits that prevent problems with more than 200 variables from being solved (Frontline 2011b). In our teaching, we have found it is very easy to hit these limits. For example, Excel is an ideal environment for formulating large set partitioning problems because it allows students to work very naturally with the columns in the model. These models often have many hundreds of variables, and so quickly go beyond Solver's capabilities. Assignment and transportation problems or time-staged models can similarly become very large. Frontline's more powerful premium Solver products could be purchased to solve these larger problems, but this is often not an option for solving a homework problem where we expect large numbers of students to be working on their own computers.

2. Solver models exist partly as formulae in the spreadsheet and partly as references to these formulae that are accessible only from the Solver dialog. This separation makes the checking of models difficult, tedious, and error prone.

3. Many operations research teaching programmes progress from Excel to a formal modelling language such as AMPL (AMPL 2012) or GAMS (GAMS 2012). Unlike Solver, modelling languages are valued for providing a clear distinction between the model and its associated data, and supporting a more formal model development process. However, these modelling languages do not work within the familiar Excel environment, but instead typically have a text-based interface, making them harder for students to learn and use.

We have developed SolverStudio, to help overcome these difficulties. We are now successfully using SolverStudio in our engineering classes to provide enhanced optimisation capabilities within the familiar Excel environment. This paper describes the benefits offered by this free tool.

2. Traditional Excel Modelling

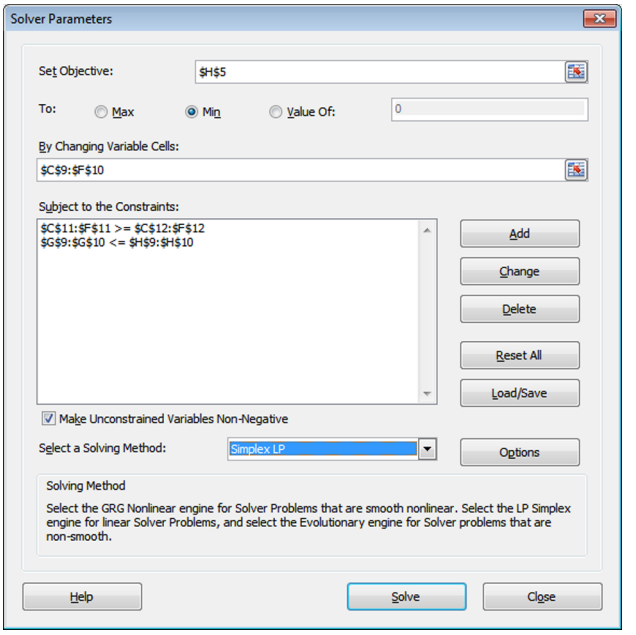
Before we introduce SolverStudio, we first review the standard approach used to develop optimisation models in Excel. Figure 1 shows a typical example of an optimisation model, in this case a transportation problem, that has been developed using Excel. (The highlighting of the model in Figure 1 has been generated using OpenSolver (OpenSolver 2012). In this example, we wish to determine a set of flows (in cells C9:F10) from warehouses to bars that meets supply and demand requirements. We wish to minimise the total cost (calculated in cell H5), where the unit cost of each warehouse/bar flow is given in cells C4:F5. We wish to ensure the total flows into the bars (as calculated, using column sums, in cells C11:F11) meet some minimum requirements (given in cells C12:F12) and the total flows out from the warehouses (as calculated, using row sums, in cells G9:G10) do not exceed the supplies available at those warehouses (given in cells H9:H10).

Once the equations and data for this model have been set up on the sheet, the user then opens up the

Figure 1 **An Example of a Standard Optimisation Model Developed in Excel**

	A	B	C	D	E	F	G	H
1								
2		Costs:						
3		Warehouses\Bars	1	2	3	4		
4		A	2	4	5	1		Total Cost
5		B	3	1	3	3		min 8000
6								
7		Deliveries:						
8		Warehouses\Bars	1	2	3	4	Outflow	Supply
9		A	500	0	0	700	1200	≤ 3100
10		B	0	900	1800	0	2700	≤ 4000
11		Inflow	500	900	1800	700		
12		Demand	≥500	900	1800	700		

Figure 2 **A Solver Model as Set Up for the Transportation Problem Shown in Figure 1**



Solver dialog, and uses this to specify the objective function, the decision variables, and the constraints. This dialog (as it appears in Excel 2010) is shown in Figure 2. Upon clicking the Solve button, Solver then builds the optimisation model from the data and formulae in the sheet, runs an optimisation solver, and then writes the solution flows back onto the sheet. This tightly integrated transfer of data between the spreadsheet and the optimisation solver is an important contributor to the success of Excel as an environment for optimisation.

Although a Solver model, such as that above, provides an excellent introduction to optimisation, we consider it important that our students are also exposed to a formal modelling language such as AMPL (AMPL 2012) or GAMS (GAMS 2012). These tools are often used by companies in their operations research departments and allow large models to be formulated quickly and easily. However, although these tools provide excellent modelling capabilities, they typically use a command line interface and text files with which our students have little familiarity.

A major strength of Excel is its ease of use as both a data entry tool and for performing calculations using this data, and thus spreadsheets are often the most natural source of problem data. Indeed, Solver's success arises from the ease with which it integrates with this data source. This contrasts with a system such as AMPL in which users typically create text files to define their problem data. As Figure 3 shows, these files require a very particular syntax that students often find difficult to remember and get correct. Furthermore, students who have previously been

Figure 3 An AMPL Data File for the Data Used in the Transportation Problem Shown in Figure 2

```
data;
param: Warehouses: supply:=
    A  3,100   B  4,000;
param: Bars: demand :=
    1   500   2  900   3  1,800   4  700;
param costs:  1  2  3  4 :=
         A    2  4  5  1
         B    3  1  3  3;
```

introduced to Solver often question why the change to a supposedly better optimisation tool requires such a step backward in the handling of their model data. (We acknowledge that AMPL can read and write spreadsheets through its database interface, but observe that this requires complex AMPL syntax and very specific data layout in the spreadsheet, and thus is often more complicated than using text files.) Thus, one of the goals in developing SolverStudio was to support modelling languages without sacrificing the ease of data entry and manipulation associated with Excel Solver models.

Once the goal of using Excel as a data source has been accepted, it is perhaps an obvious step to again follow the Solver model and use Excel to also provide the interface to the modelling language itself. Thus, we wanted a system that would allow our students to work entirely within Excel but still have access to a fully fledged modelling language. Several such systems are available commercially, including the AIMMS Excel add-in (AIMMS 2012) that is part of the larger AIMMS modelling and interface development system, and the Microsoft Solver Foundation add-in (Solver Foundation 2012). Both of these have their own proprietary modelling languages that, we suggest, are not as commonly used as AMPL or GAMS. They also have licensing requirements that make widespread usage more difficult. Furthermore, the future of Microsoft Solver Foundation is currently uncertain (DevLabs 2012).

3. SolverStudio

To avoid these difficulties, we have created SolverStudio, a free add-in that embeds within Excel three modelling languages (AMPL, GAMS, and GMPL) and three Python-based modelling environments (PuLP, COOPR/Pyomo, and Gurobi). As we show next, we believe SolverStudio has achieved our goal of seamless integration by providing an essentially invisible transfer of data and solution values between the spreadsheet and the optimisation model. Furthermore, recent versions of SolverStudio also provide support for the Python-based SimPy, thereby extending this paradigm into the simulation space and thus making SolverStudio a natural platform for teaching both optimisation and simulation modelling.

SolverStudio (http://solverstudio.org) is downloaded and installed as a Visual Studio Tools for Office (VSTO) .Net Excel add-in. (This installation can typically be completed without requiring administrator privileges, making it suitable for a teaching computer laboratory.) Once installed, the SolverStudio group of menu items becomes available in the Excel Data ribbon, as shown in Figure 4. This figure also shows a spreadsheet containing Python code for a

Figure 4 A SolverStudio Transportation Model in PuLP with the Data Items Highlighted

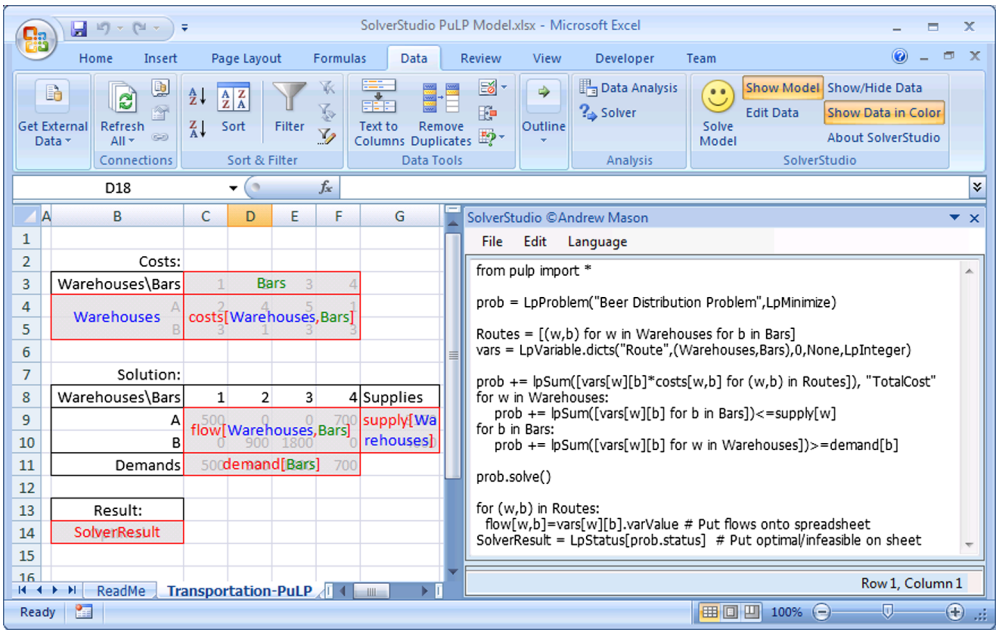
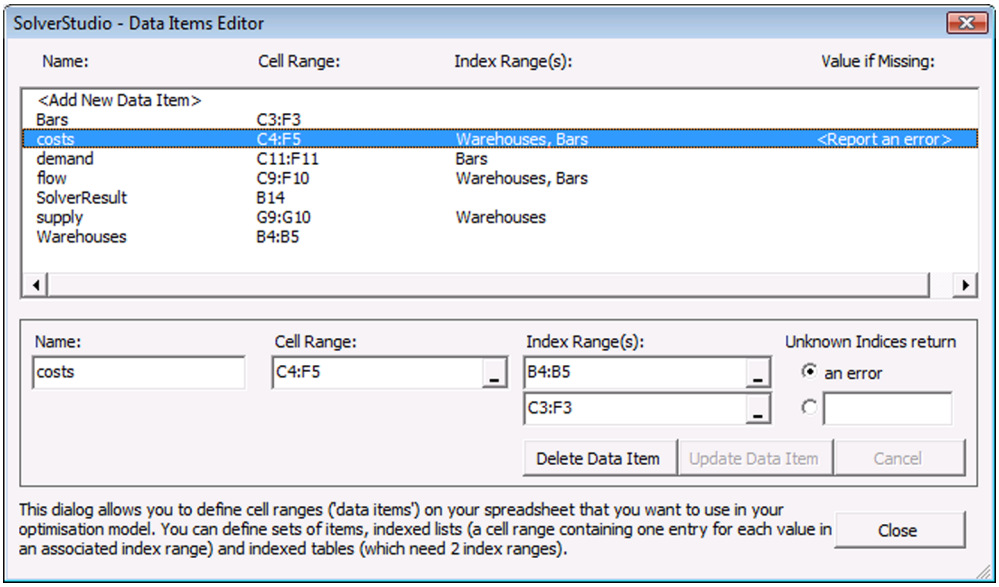


Figure 5 SolverStudio's Data Items Editor Showing the Data Items for the Model in Figure 4



PuLP (PuLP 2012) implementation of the transportation model in Figure 1.

The user's code for a model is edited directly within Excel using SolverStudio's text editor pane (shown on the right of Figure 4), and the data is entered as tables on the spreadsheet. SolverStudio provides a data items editor that allows the user to give names to these tables for use within the model, and specify any indices these data items require. Figure 5 shows SolverStudio's data items editor as used for the model in Figure 4. The data items for this transportation example include the sets *Warehouses* and *Bars* and the indexed parameters *costs*, *demand*, *supply* and *flow*. We see, for example, that the *Warehouses* data item is defined as a set of values contained in cells B4:B5, and the *costs* data item is a collection of values in cells C4:F5 indexed over two other data items, *Warehouses* and *Bars*. The *SolverResult* data item refers to a single cell, B14, which is used in this example to display the status result (e.g. "optimal" or "infeasible") when the model is solved. To help the user check their data items, SolverStudio provides a "Show/Hide Data" button that visually overlays these data items on the sheet with their names and associated indices; this is illustrated in Figure 4.

As the PuLP model in Figure 4 shows, data items can simply be referenced by name in the model once they have been set up with the data items editor. Because this is a Python-based PuLP model, these data items are created by SolverStudio as Python lists and dictionaries, and are referenced as such in the Python code used to build the PuLP model. Any changes made to a data item by the model are automatically written back to the spreadsheet. This feature is used in this example to display the solution on the spreadsheet by defining a data item *flow* into which the optimal solution is written. This example also writes the optimiser result (optimal, infeasible, or unbounded) onto the sheet using the *SolverResult* data item.

Once a SolverStudio model has been developed, it can be run using the SolverStudio "Solve Model" button, resulting in an end-user experience very similar to that provided by Solver. SolverStudio also mimics Solver in that it stores the model and all the associated data information within the spreadsheet itself, making it easy to distribute and share SolverStudio models.

SolverStudio was initially created to support the Python-based modelling language PuLP. However, the Python code used to underpin SolverStudio made it easy to support other languages, and so SolverStudio now provides a language menu from which any of the following modelling languages can be selected:

1. PuLP (PuLP 2012), an open-source Python-based COIN-OR (COIN-OR 2011) modelling language, developed by Stuart Mitchell, that is included in the SolverStudio download.

2. The commercial modelling languages AMPL (AMPL 2012) and GAMS (GAMS 2012). These are not provided with SolverStudio, but must be installed by the user. However, SolverStudio provides a menu item to automatically download and install the free restricted student edition of AMPL, which is often used in classroom teaching. (AMPL also provides full but time-limited versions of AMPL for use in teaching that can be used with SolverStudio.) SolverStudio will also use the free restricted version of GAMS if it is installed. Both of these restricted versions allow no more than 300 variables and 300 constraints.

3. SolverStudio also allows AMPL and GAMS models to be solved in the cloud using the online NEOS servers (NEOS 2012). This gives users the ability to solve full-sized linear and nonlinear AMPL and GAMS models using a wide range of solvers. SolverStudio automatically handles the model changes, solver selection, and data transfers required by NEOS, thereby shielding users from the complexities of the NEOS-specific software and Web submission processes.

4. GMPL (GNU MathProg Language), an open-source system that provides a subset of the AMPL language. GMPL is distributed as part of the GNU Linear Programming Kit (GPLK 2012), and is included in the SolverStudio download.

5. The open-source COOPR/Pyomo optimisation system (COOPR 2013). This requires the user to first download and install a full version of Python (such as Active Python (ActivePython 2013)) and then download and install the COOPR system. It is planned to include a more tightly integrated version of COOPR in a future release of SolverStudio.

6. Gurobi (Gurobi 2012), a commercial solver accessed using the Gurobi Python modelling interface. This is not included with SolverStudio, but instead must be installed by the user. A free academic version is available with no size limits.

7. The open-source Python-based SimPy simulation language (SimPy 2013). This is included with the SolverStudio download.

8. Any other Python software that the user has installed.

We note that data items defined on a spreadsheet become available to use in an optimisation (or simulation) model written using any of these modelling languages. This makes it easy for a user to set up their data and then experiment with different languages.

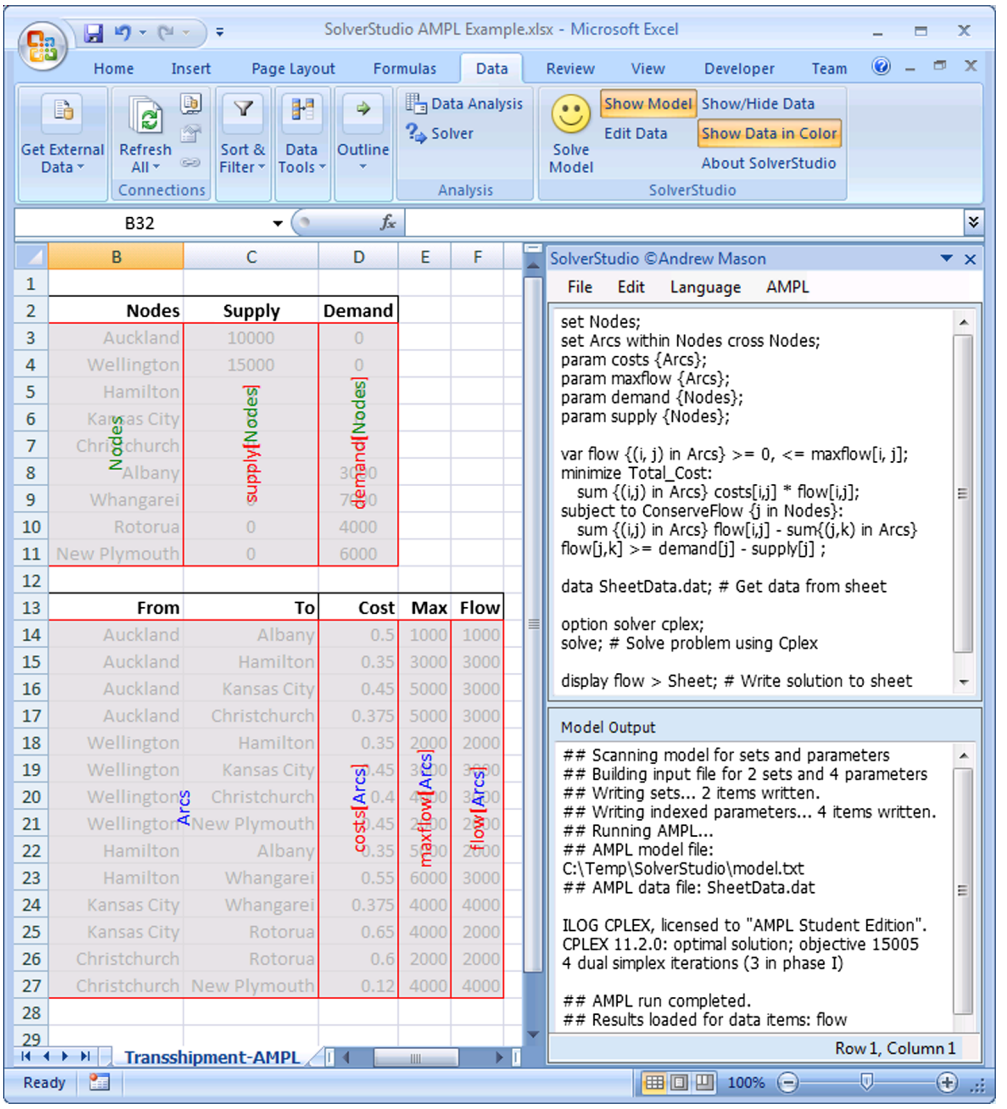
Figure 6 shows a second, more complex, SolverStudio model in which a transshipment formulation has been developed, this time using AMPL. This model is less well suited to the double-indexed tables used in Figure 4, and instead the parameters *costs* and *maxflow* and decision variables *flow* are indexed over a tuple formed by a pair of nodes. (Arbitrarily complex indexing is supported by SolverStudio, allowing, for example, a table with four indices arranged as pairs identifying each row and column.) AMPL users reading this model will note that commands such as "solve" are included directly within the model file (instead of in a ".run" file), and that "data SheetData.dat" and "display flow > Sheet" are used to get data from the sheet and then put results back into the sheet. Recent versions of SolverStudio do not require these commands in the model, but will add them automatically if they are not present. Similar approaches are used in the other supported modelling languages.

Each language supported in SolverStudio has its own menu that provides features specific to that language. For example, the AMPL menu provides access to the recently released online version of the AMPL book (Fourer et al. 2003), allows viewing of the last AMPL data file created by SolverStudio from the data items, and as mentioned above, allows the restricted student version of AMPL to be installed. To support existing AMPL users, it also provides an "Import…" function that will read an AMPL data file into a spreadsheet and then create appropriately named data items. (Similar support is provided for loading the proprietary GAMS GDX data files.) The Gurobi support is more extensive in that it allows users to install and register license files using the dialog shown in Figure 7, removing the need to manually run `grbgetkey` from the command line. Gurobi keys are also saved, making it easier to reregister an existing license as is often required on a machine used by multiple users.

The newest addition to SolverStudio is the SimPy simulation toolbox for Python. SimPy is an object-oriented, process-based, discrete-event simulation language made available under a GNU Lesser GPL License. SimPy has excellent supporting resources including an associated online book (Matloff 2008a) and tutorial (Matloff 2008b). Integrating SimPy with SolverStudio makes it easy to transfer data into the simulation, and also to extract results for further analysis and plotting. The combination of both PuLP and SimPy also allows integrated models to be developed where, for example, an optimization problem can be solved as part of a simulation run.

Since being released in November 2011, SolverStudio has been downloaded over 2,500 times, with download counts recently increasing to around 500 per month. Contact with users suggests it is being used by both companies and universities. For example, Dr. Daniel Frances makes SolverStudio available for students to use at the University of Toronto in his *Cases in OR* course in Industrial Engineering, and is also using SolverStudio in his math programming to introduce AMPL to students who have used Excel Solver. (His approach is interesting in that students have access to Excel and SolverStudio during their tests and examinations.) Ted Ralphs (Department of Industrial and Systems Engineering, Lehigh University) is trialling SolverStudio in a short course on financial optimization, where he presents it as one of several tools for solving financial optimization problems. We are also using SolverStudio to teach AMPL to students at the University of Auckland, where this AMPL course follows an earlier introductory course using Solver. We are also aware of commercial users who are using SolverStudio and PuLP to further develop optimisation tools that have started

Figure 6 A SolverStudio Transhipment Model in AMPL Showing Highlighted Data



as Excel Solver models. We hope that SolverStudio will encourage many more such progressions.

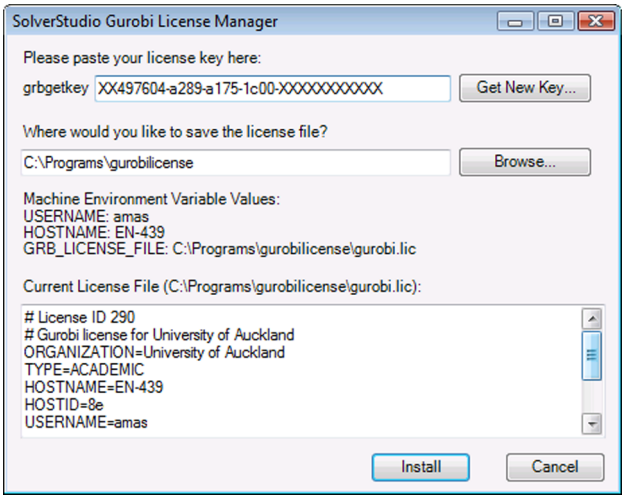
4. Implementation and Advanced Capabilities

SolverStudio is written in C# using the Microsoft .Net framework. As mentioned earlier, SolverStudio was initially designed to support the Python PuLP software. This support was provided by including within SolverStudio the open-source IronPython system (IronPython 2013) that allows Python software to be compiled and run on the .Net framework. Because IronPython is a fully fledged .Net language, SolverStudio can simply convert the data items into IronPython data structures and then run the PuLP model within an IronPython context that makes these data structures accessible. This results in minimal overhead associated with the exchange of data between Excel and the PuLP model.

The IronPython integration means SolverStudio is not restricted to only running simple PuLP models, but supports a very broad set of Python features. For example, Python-based models can open external databases, access data over the Internet, and execute processes remotely, all from within Excel. SolverStudio also provides access to Excel's API (Application Programming Interface), meaning that Python models in SolverStudio provide an Excel scripting capability similar to that of VBA (Visual Basic for Applications). This gives advanced users direct access to the spreadsheet, allowing changes to be made without using SolverStudio data items.

This strong Python capability has been used to add multiple languages such as AMPL and GAMS to SolverStudio by creating specific Python programs

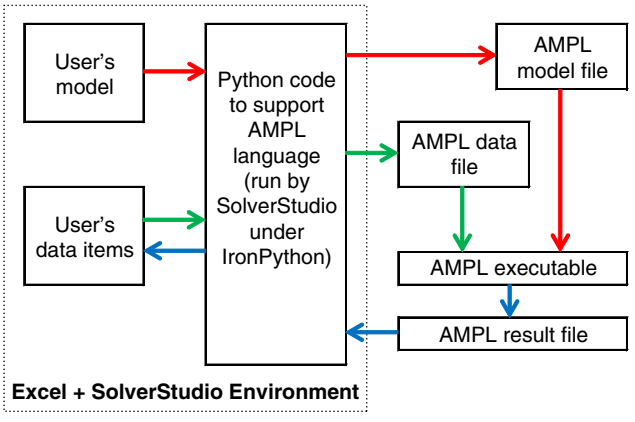
Figure 7 SolverStudio Provides Support for Installing Gurobi Licences



that support each of the available languages. As an example, Figure 8 shows how this works for AMPL. When the user solves an AMPL model, SolverStudio executes the Python AMPL support code that writes out the AMPL model and data files, runs AMPL, and then reads the AMPL output file to get the results to write back to the spreadsheet. To prepare the AMPL data file, the AMPL model is parsed to classify the data items as sets, parameters, or variables, and this information is used to create these items correctly in the AMPL data file. This approach means this classification of data items is only made once (in the user's model file) and results in a simple data items editor that is language independent.

Unfortunately, IronPython has some limitations, and so some Python packages such as COOPR/Pyomo need to be run under a more standard Python implementation such as ActivePython for Windows (ActivePython 2013) that needs to be installed separately by the user. Some users also prefer to run

Figure 8 SolverStudio Supports Multiple Languages, Such a AMPL, Using Language-Specific Support Files Written in Python



PuLP and SimPy models in this way. SolverStudio supports this by providing a "Python (external)" language that is treated in a similar way to other external solvers in that files are used to manage the exchange of data between SolverStudio and the external Python environment. Careful use of Python's *atexit* handler allows changes made by the user's Python code to any SolverStudio data items to be identified and then written back onto the spreadsheet. This results in minimal differences for the user when switching between IronPython and an external Python implementation.

This approach of using external Python files to provide language-specific support has proven to be very flexible and allows new languages to be added easily. For example, after introducing SolverStudio in one of our taught courses, we found that students were wanting to solve larger models than could be handled using the free student version of AMPL. To address this, we first looked at converting the models from AMPL or GMPL. This conversion was not difficult, but the run times under GMPL's GLPK solver were much longer than under the AMPL/Gurobi setup we had been using. We then learned that students were manually submitting their AMPL models to NEOS. Thanks to sample Python code provided by NEOS, it took just a few days to add NEOS support for AMPL to SolverStudio. This approach gave much shorter run times than the GMPL alternative, and proved to be popular with our students. When it came to adding GAMS support at a later stage, we found that the GAMS implementation on NEOS produced a result file on the NEOS servers that could only be accessed using FTP (file transfer protocol). Python allowed us to automatically fetch this file and decode it using software kindly provided to us by GAMS, providing a fully seamless experience to the user.

We encourage users to look at the Python support files provided in the SolverStudio distribution and to customise these to support their own preferred modelling environments. We would welcome then including these in future SolverStudio distributions.

5. Conclusions

We believe that spreadsheet-based modelling is an important operations research tool that will continue to be used for many years to come. Although we are strong advocates of spreadsheet-based teaching for optimisation classes, we also recognise that modelling languages are another important item in the operations research toolbox. SolverStudio exploits Excel's familiarity and ease of use to help make modelling languages more accessible to students. The integrated environment provided by SolverStudio allows students to enjoy the benefits of a training in a formal modelling language while recognising the value of

Excel as a natural platform for teaching optimisation. SolverStudio provides Excel users with ready access to both commercial and free open-source modelling tools, including AMPL, GAMS, PuLP, and Gurobi. It also allows users to exploit a wide range of online solvers provided by NEOS. The recent addition of SimPy support to SolverStudio makes it a natural single platform to use when teaching both optimisation and simulation. We hope that SolverStudio will be valued by students and educators for many years to come.

Acknowledgments

Stuart Mitchell provided modified PuLP code used in SolverStudio, and Oscar Dowson made valuable coding contributions including adding COOPR/Pyomo, GAMS on NEOS, and SimPy support. Cameron Walker, Michael O'Sullivan, and their students provided invaluable feedback by testing SolverStudio in a classroom. Daniel Frances also provided very useful suggestions for improvements based on his experience in installing and using SolverStudio at the University of Toronto. Gurobi, AMPL, GAMS, and NEOS have all given us useful assistance during our development. Finally, we thank all those users and students who have taken time to provide feedback on SolverStudio, resulting in many improvements. This paper was much improved by the valuable feedback from the anonymous referees. All trademark terms, including AIMMS, Excel, AMPL, Gurobi, and GAMS, are the property of their respective owners.

References

ActivePython (2013) Download Python: ActivePython Community Edition. Retrieved 20 May. http://www.activestate.com/activepython/downloads.

AIMMS (2012) AIMMS Excel Add-In User's Guide—Introduction to the AIMMS Excel Add-In. Retrieved 20 July. http://www.aimms.com/aimms/download/manuals/aimms3exc_introduction.pdf.

AMPL (2012) AMPL®: A Modeling Language for Mathematical Programming. Retrieved 21 November, http://ampl.com/.

COIN-OR (2011) COmputational INfrastructure for Operations Research—Open source for the Operations Research community. Retrieved 8 November, http://www.coin-or.org.

COOPR (2013) Coopr: A COmmon Optimization Python Repository, Retrieved 20 January, https://software.sandia.gov/trac/coopr.

DevLabs (2012) The Future of Microsoft Solver Foundation. Retrieved 20 July, http://social.msdn.microsoft.com/Forums/en-US/solverfoundation/threads.

Fourer R, Gay DM, Kernighan BW (2003) *AMPL: A Modeling Language for Mathematical Programming* (Cengage Learning, Independence, KY). Available online at http://www.ampl.com/BOOK/download.html.

Frontline (2011a) Frontline's Optimization and Simulation Solvers. Retrieved 21 November, http://www.solver.com/products-overview.htm.

Frontline (2011b) Standard Excel Solver—Dealing with Problem Size Limits. Retrieved 21 November, http://www.solver.com/suppstdsizelim.htm.

GAMS (2012) GAMS Home Page. Retrieved 20 July, http://www.gams.com/.

GPLK (2012) GLPK (GNU Linear Programming Kit). Retrieved 20 July, http://www.gnu.org/software/glpk/glpk.html.

Grossman T (2010) Resources for Spreadsheet Analysts, Analytics, May/June. Retrieved 20 Nov 2011, http://viewer.zmags.com/publication/0dfc3a66#/0dfc3a66/9.

Gurobi (2012) Gurobi Optimization. Retrieved 20 July, http://www.gurobi.com/.

IronPython (2013) IronPython: The Python programming language for the .NET Framework. Retrieved May, http://ironpython.net/.

Jensen P (2012) Operations Research Models and Methods Internet, retrieved 20 July, http://www.me.utexas.edu/~jensen/ORMM/index.html.

LeBlanc LJ, Grossman TA (2008) Introduction: The use of spreadsheet software in the application of management science and operations research. *Interfaces* 38(4):225–227.

Lindo (2012) Lindo Systems Inc, Our Optimization Software. Retrieved June. http://www.lindo.com/.

Matloff N (2008a) A Discrete-Event Simulation Course Based on the SimPy Language. Retrieved May 2013. http://heather.cs.ucdavis.edu/~matloff/simcourse.html.

Matloff N (2008b) Introduction to Discrete-Event Simulation and the SimPy Language. Retrieved May 2013. http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf.

NEOS (2012) NEOS Server for Optimization. Retrieved November. http://www.neos-server.org/neos/.

OpenSolver (2012) An Open-Source Solver for Excel. Retrieved December. http://opensolver.org.

Oracle (2012) Oracle Crystal Ball. Retrieved Feb 2013. http://www.oracle.com/us/products/applications/crystalball/.

Palisade (2012) Palisade Corporation: Maker of Risk and Decision Analysis Software using Monte Carlo Simulation. Retrieved June. http://www.palisade.com/.

PuLP (2012) PuLP: An LP modelling system written in Python. Retrieved 20 July. https://projects.coin-or.org/PuLP.

Solver Foundation (2012) Microsoft Solver Foundation. Retrieved 20 July. http://msdn.microsoft.com/en-us/devlabs/hh145003.aspx.

SimPy (2013) SimPy Simulation Package. Retrieved 20 May. http://simpy.sourceforge.net/.