OpenSolver: Open Source Optimisation for Excel

Article · January 2010

CITATIONS
21

READS
8,866

3 authors, including:

Andrew J. Mason
University of Auckland
56 PUBLICATIONS **1,056** CITATIONS

SEE PROFILE

OpenSolver: Open Source Optimisation for Excel

www.opensolver.org

Andrew J Mason
Department of Engineering Science
University of Auckland
New Zealand
a.mason@auckland.ac.nz

Iain Dunning
Department of Engineering Science
University of Auckland
New Zealand

Abstract

We have developed an open source Excel add-in, known as OpenSolver, that allows linear and integer programming models developed in Excel to be solved using the COIN-OR CBC solver. This paper describes OpenSolver's development and features.

Key words: OpenSolver, Open Source, Excel, Solver, Add-in, COIN-OR, CBC.

1 Introduction

Microsoft Excel (Wikipedia, 2010) contains a built-in optimisation tool known as Solver (Frontline, 2010). Solver is developed by Frontline Systems, who provide the software to Microsoft. Using Solver, a user can develop a spreadsheet optimisation model and then solve it to find an optimal solution. Many introductory optimisation courses use Solver and Excel to introduce students to modelling and optimisation. Unfortunately, when students apply their optimisation skills to real-world problems, they often discover that the size of problem Solver can optimise is artificially limited to no more than 200 decision variables (Solver Limits, 2010), and that they need to upgrade Solver to one of Frontline's more expensive products, such as Premium Solver. (Beta versions of the latest Excel version, Excel 2010, have a smaller variable limit (Dunning 2010); the current variable limit in shipping versions of Excel 2010 has not been confirmed.)

We have recently been involved in the development of a large staff scheduling model. This model was developed as part of a consultancy exercise where it was important that the model could be used and scrutinised by a range of different users. Satisfying this requirement using commercial products would have been logistically difficult and expensive.

For a number of years, a group known as COIN-OR (Computational Infrastructure for OR) (COIN-OR, 2010) have been developing and promoting open source optimisation software, including the linear/integer programming optimiser known as

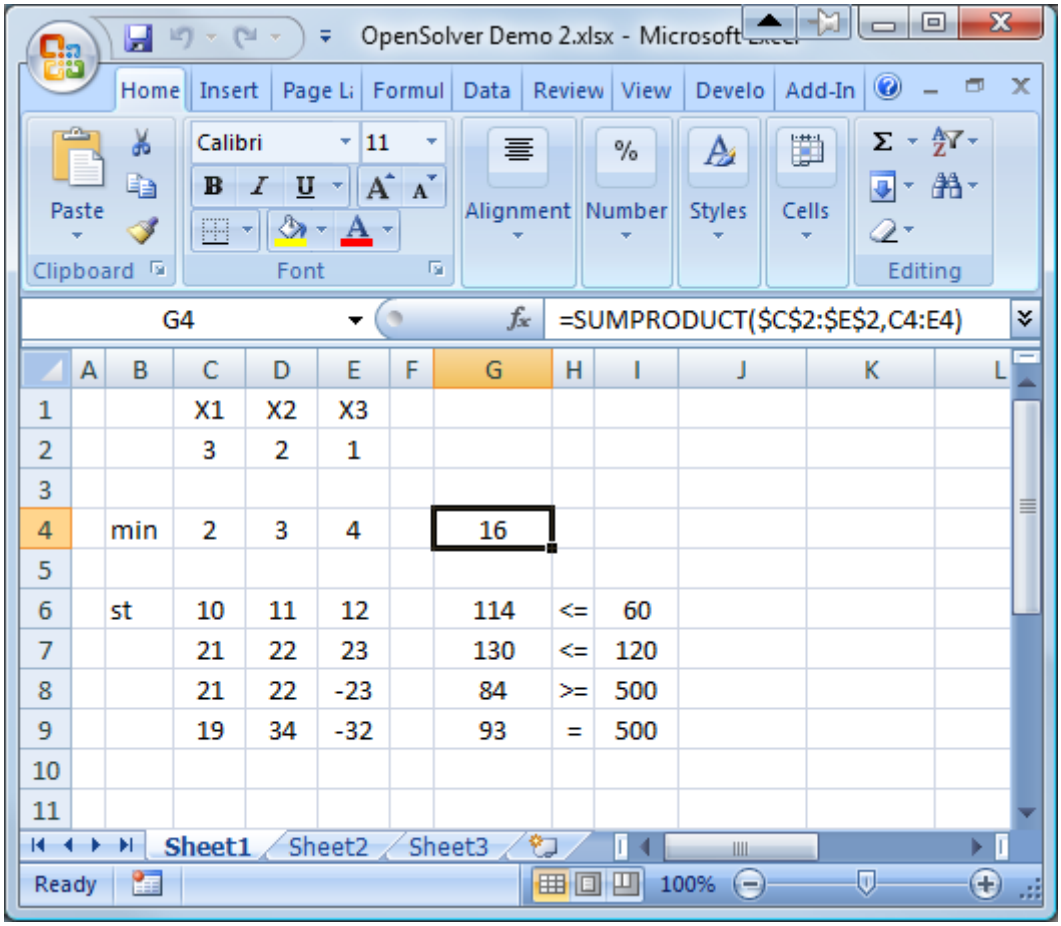
CBC (CBC, 2010). While typically not as fast as its commercial equivalents, the COIN-OR software has gained a reputation for quality and reliability and is now widely used in industrial applications.

The combination of Excel and COIN-OR is a compelling one, and appeared to be a perfect solution for our large staff scheduling model. Unfortunately, we were unable to find any software that allowed Excel spreadsheet models to be solved using any of the COIN-OR optimisers. A decision was taken to develop such software to progress the scheduling project. This decision eventually led to the development of OpenSolver, a freely available Excel add-in that allows existing spreadsheet linear and integer programming models to be solved using the COIN-OR CBC optimiser

2 Solver Operation

We start by briefly describing how optimisation models are built using Solver. Readers familiar with Solver may wish to skip this section.

A typical linear programming model is shown in Figure 1. The decision variables are given in cells C2:E2. Cell G4 defines the objective function using a 'sumproduct' formula defined in terms of the decision variables and the objective coefficients in cells C4:E4. Each of the constraints are defined in terms of a constraint left hand side (LHS) in G6:G9 and right hand side (RHS) in I6:I9. Note that the 'min' in B4 and the constraint relations in H6:H9 are for display purposes only, and are not used by Solver.



	A	B	C	D	E	F	G	H	I	J	K	L
1			X1	X2	X3							
2			3	2	1							
3												
4		min	2	3	4		16					
5												
6		st	10	11	12		114	<=	60			
7			21	22	23		130	<=	120			
8			21	22	-23		84	>=	500			
9			19	34	-32		93	=	500			
10												
11												

Figure 1: A typical spreadsheet optimization model. Many models do not follow this layout, but instead 'hide' the model inside the spreadsheet formulae.

Figure 2 shows how this model is set up using Solver. The decision variables, objective function cell and constraint left and right hand side cells must all be specified by the user within Solver. Note that a constraint can either consist of two multi-cell ranges of the same dimension, such as A4:A8 >= B4:B8, or a multi-cell range and a constant or single-cell range such as A4:A8 <= 8. Integer and binary restrictions on the variables can also be specified using constraints.

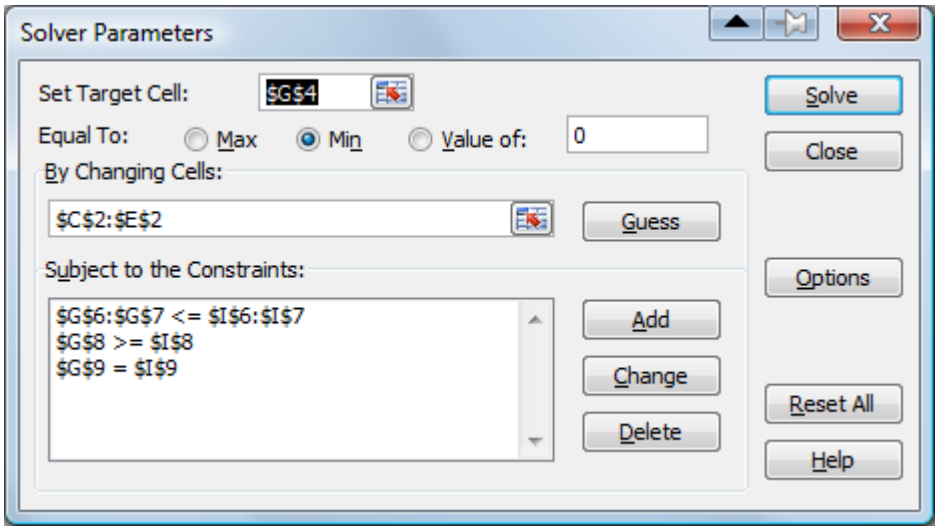


Figure 2: Solver setup for the model above.

The means by which Solver stores a model does not appear to be documented. However, a Google search eventually uncovered a comment that Solver uses hidden named ranges (see, e.g., Walkenbach 2010). The Excel Name Manager add-in (Pieterse, 2010), developed by Jan Karel Pieterse of Decision Models UK, was then used to reveal this hidden data. The names used by Solver are documented in Appendix 1. OpenSolver uses these named ranges to determine the decision variables, the objective function and LHS and RHS ranges of each constraint.

3 Constructing a Model from Excel Ranges

We wish to analyse the spreadsheet data to build an optimisation model which has equations of the form:

Min/max	$c_1x_1+c_2x_2+...$	c_nx_n
Subject to	$a_{11}x_1+a_{12}x_2+...$	$a_{1n}x_n \leq/=/\geq b_1$
	$a_{21}x_1+a_{22}x_2+...$	$a_{2n}x_n \leq/=/\geq b_2$
	...	
	$a_{m1}x_1+a_{m2}x_2+...$	$a_{mn}x_n \leq/=/\geq b_m$
	$x_1, x_2, ..., x_n >= 0$	

Assuming the model is linear, then the Excel data can be thought of as defining an objective function given by

$$\text{Obj}(\mathbf{x}) = c_0+c_1x_1+c_2x_2+...c_nx_n$$

where $\mathbf{x}=(x_1, x_2, ..., x_n)$ are the decision variable values, Obj($\mathbf{x}$) is the objective function cell value, and c_0 is a constant. Similarly, each constraint equation *i* is defined in Excel by

$$LHS_i(\mathbf{x}) \leq/=/\geq RHS_i(\mathbf{x}) \Leftrightarrow LHS_i(\mathbf{x}) - RHS_i(\mathbf{x}) \leq/=/\geq 0, \quad i=1, 2, ..., m$$

where $LHS_i(\mathbf{x})$ and $RHS_i(\mathbf{x})$ are the cell values for the LHS and RHS of constraint *i* respectively given decision variable values **x**. Because Solver allows both $LHS_i(\mathbf{x})$ and $RHS_i(\mathbf{x})$ to be expressions, we assume that both of these are linear functions of the decision variables. Thus, we have

$$a_{i1}x_1+a_{i2}x_2+...a_{in}x_n- b_i = LHS_i(\mathbf{x}) - RHS_i(\mathbf{x}), \quad i=1, 2, ..., m.$$

OpenSolver determines the coefficients for the objective function and constraints through numerical differentiation. First, all the decision variables are set to zero, $\mathbf{x}=\mathbf{x}^0=(0,0,...,0)$ giving:

$$c_0=Obj(\mathbf{x}^0)$$
$$b_i = RHS_i(\mathbf{x}^0) - LHS_i(\mathbf{x}^0), \quad i=1, 2, ..., m$$

Then, each variable x_j is set to 1 in turn, giving decision variables values $\mathbf{x}=\mathbf{x}^j$, where $\mathbf{x}^j=(x_1^j, x_2^j, ..., x_n^j)$ is a unit vector with the single non-zero element $x_j^j=1$. Then, the spreadsheet is recalculated, and the change in the objective and LHS and RHS of each constraint recorded. This allows us to calculate the following coefficients:

$$c_j = Obj(\mathbf{x}^j)-c_0$$
$$a_{ij} = LHS(\mathbf{x}^j)-RHS(\mathbf{x}^j)+b_i, \quad i=1, 2, ..., m$$

This process assumes that these equations are indeed all linear. OpenSolver does not currently check this, but will do so in a future release.

The speed of this process is set by the speed at which Excel can re-calculate the spreadsheet. As an example, the large staff scheduling model discussed earlier has 532 variables and 1909 constraints, and took 22s to build (and just 1 second to solve) on an Intel Core-2 Duo 2.66GHz laptop. There are a number of different Excel calculation options available, such as just recalculating individual cells or recalculating the full worksheet or workbook. OpenSolver currently does a full workbook recalculation as the other options did not appear to give any faster results.

In our model form above, we have specified that the decision variables are all non-negative. This assumption is also made by the COIN-OR CBC solver we use. To ensure this is the case, OpenSolver requires the user to set Solver's "Assume Non-Negative" option. OpenSolver cannot currently solve models for which this is not the case. OpenSolver also requires that Solver's "Assume Linear Model" is set.

4 Integration with COIN-OR CBC Solver

OpenSolver uses the COIN-OR CBC linear/integer programming solver to generate its solutions. While this solver can be called directly as a DLL, OpenSolver adopts the simpler process of writing a '.lp' file that contains the model definition and then calling the command line version of CBC with this file as an input argument. CBC then produces an output file which is read back by OpenSolver. If a solution has been found, OpenSolver uses this to set the values of the decision variable cells. Any infeasibility or unboundedness, or early termination caused by time limits, is reported using a dialog.

Solver provides the user with a number of solution options. Of these, the "Max Time" option is passed to CBC (using the CBC parameter "-seconds"), and the "Tolerance" option is passed (using "-ratioGap") to set the percentage tolerance required for CBC integer solutions. Other CBC options can be set by creating a two column

range named "OpenSolver_CBCParameters"on the spreadsheet that contains CBC parameter names and values.

If the "Show Iteration Results" option is set, OpenSolver displays the CBC command line window while CBC is running. (OpenSolver does not provide the step-by-step functionality that this option enables in Solver.)

5 Excel Integration and User Interface

OpenSolver is coded in Visual Basic for Applications and runs as an Excel add-in. The add-in presents the user with new OpenSolver controls using the standard ribbon interface. These OpenSolver buttons and menus are shown in Figure 3 and Figure 4.

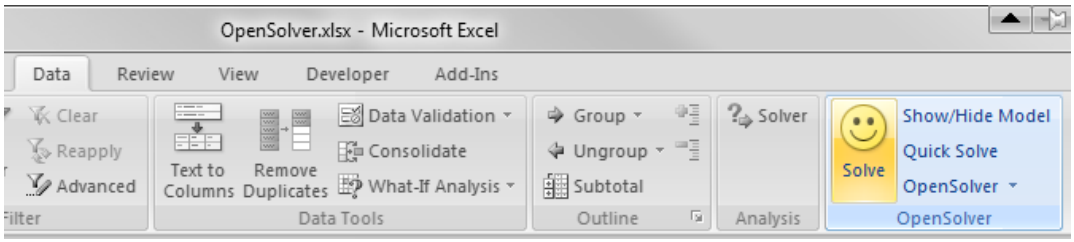


Figure 3: OpenSolver's buttons and menu appear in Excel's Data ribbon.

Figure 4: OpenSolver's menu gives access to its more advanced options.

OpenSolver is downloaded as a single .zip file which when expanded gives a folder containing the CBC files and the OpenSolver.xlam add-in. Double clicking OpenSolver.xlam loads OpenSolver and adds the new OpenSolver buttons and menu to Excel. OpenSolver then remains available until Excel is quit. If required, the OpenSolver and CBC files can be copied to the appropriate Office folder to ensure OpenSolver is available every time Excel is launched. Note that no installation program needs to be run to install either OpenSolver or CBC.

6 Performance

We have found OpenSolver's performance to be similar or better than Solver's. CBC appears to be a more modern optimizer than Solver's, and so gives much improved

performance on some difficult problems. For example, large knapsack problems which take hours with Solver are solved instantly using OpenSolver, thanks to the newer techniques such as problem strengthening and preprocessing used by CBC.

7 Model Visualisation

To view an optimisation model developed using the built-in Solver, the user needs to check both the equations on the spreadsheet and the model formulation as entered into Solver. This separation between the equations and the model form makes checking and debugging difficult. OpenSolver provides a novel solution to this in the form of direct model visualisation on the spreadsheet. As Figure 5 shows, OpenSolver can annotate a spreadsheet to display a model as follows:

- The objective cell is highlighted and labelled min or max
- The adjustable cells are shaded. Binary and integer decision variable cells are labelled ‘b’ and ‘i’ respectively.
- Each constraint is highlighted, and its sense (≥,≤, or =) shown (using a ‘>’, ‘<’ or ‘=’ respectively).

We have found this model visualisation to be very useful for checking large models, and believe it will be very useful debugging tool for students learning to use Solver.

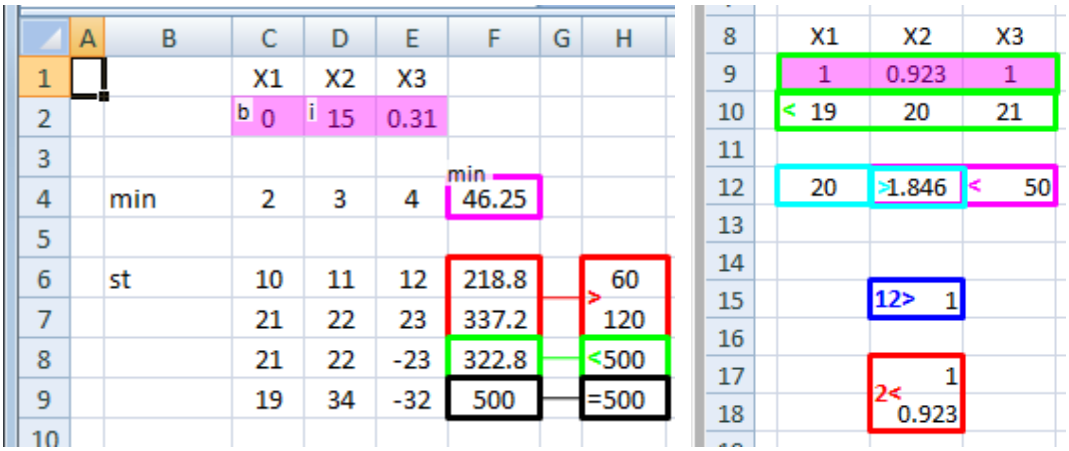


Figure 5: OpenSolver can display an optimisation model directly on the spreadsheet. The screenshot on the left shows OpenSolver’s highlighting for the model given earlier, while the screenshot on the right illustrates OpenSolver’s highlighting for several other common model representations.

8 Automatic Model Construction

The original design of OpenSolver envisaged the continued use of Solver to build (but not solve) the optimisation models. However, we have developed additional functionality that allows OpenSolver to build Solver-compatible models itself without requiring any user-interaction with Solver. Our approach builds on the philosophy that the model should be fully documented on the spreadsheet. Thus, we require that the spreadsheet identifies the objective sense (using the keyword ‘min’ or ‘max’ or variants of these), and gives the sense (>=,<=, or =) of each constraint. Our example problem shown in Figure 1 satisfies these requirements. In our teaching, we have always recommended this model layout as good practice, and note that many textbooks follow a similar approach. Our keyword-based approach creates only a minimal additional

burden for the user but, by allowing the model construction process to be automated, delivers what we consider to be a significantly improved modelling experience.

To identify the model, OpenSolver starts by searching for a cell containing the text `min' or 'max' (or variants on these). It then searches the cells in the vicinity of this min/max cell to find a cell containing a formula (giving preference to any cell containing a 'sumproduct' formula); if one is found, this is assumed to define the objective function. The user is then asked to confirm that this is the objective function cell, or manually set or change the objective function cell.

After identifying the objective function cell, OpenSolver then tries to locate the decision variables. Because the objective function depends on the decision variable cells, these decision cells must be contained in the set of the objective function's 'precedent' cells (i.e. the cells referred to by the objective function formula, either directly or indirectly through other intermediate cells). However, if a sumproduct is used, for example, these precedents will also contain the objective function coefficients. OpenSolver attempts to distinguish between decision variables and objective function coefficients by looking at the number of dependent cells. An objective function coefficient will typically only have one dependent cell, being the objective function cell. However, decision variables will have many dependents as they feature in both the objective function cell and in the constraint cells. Thus, OpenSolver examines each objective function precedent cell, and keeps as decision variables those cells with more than 1 successor. The resultant set of decision variables is presented to the user to confirm or change.

The next step is to identify any binary or integer restrictions on the decision variables. These are assumed to be indicated in the spreadsheet by the text 'binary' or 'integer' (and variants of these) entered in the cell beneath any restricted decision variable.

The final requirement is to determine the constraints. OpenSolver considers all cells (except for the objective function cell) that are successors of the decision variables, and searches in the vicinity of each of these for one of the symbols '<=', '<', '=', '>=', or '>'. A constraint is created for each occurrence of these symbols.

As shown in , OpenSolver allows the user to correct mistakes made when determining the objective function cell, the decision variable cells, and/or any binary or integer restrictions on these cells. Errors in the constraints currently need to be corrected using Solver. Once the model has been confirmed (and/or corrected) by the user, it is constructed and stored using the standard Solver named ranges, and thus can be solved using Solver or OpenSolver. It can also be edited using Solver. Testing on a range of spreadsheet models has shown that the auto-model algorithms work correctly for commonly taught model layouts including tabular models such as transportation.

The AutoModel feature has been developed by Iain Dunning, a student in the Engineering Science department at the University of Auckland

9 Advanced Features

OpenSolver offers a number of features for advanced users, including:

- The ability to easily solve an LP relaxation,
- Interaction with the CBC solver via the command line,
- Faster running using 'Quick Solve' when repeatedly solving the same problem with a succession of different right hand sides, and
- Viewing of the .lp problem file and CBC's solution file

Note that our large scheduling problem had to be solved repeatedly for 52 different right hand side values. The Quick Solve feature was developed to remove the need to repeatedly analyse the spreadsheet to determine the model coefficients. Using Quick Solve reduced the scheduling problem's run times for from one hour to one minute.

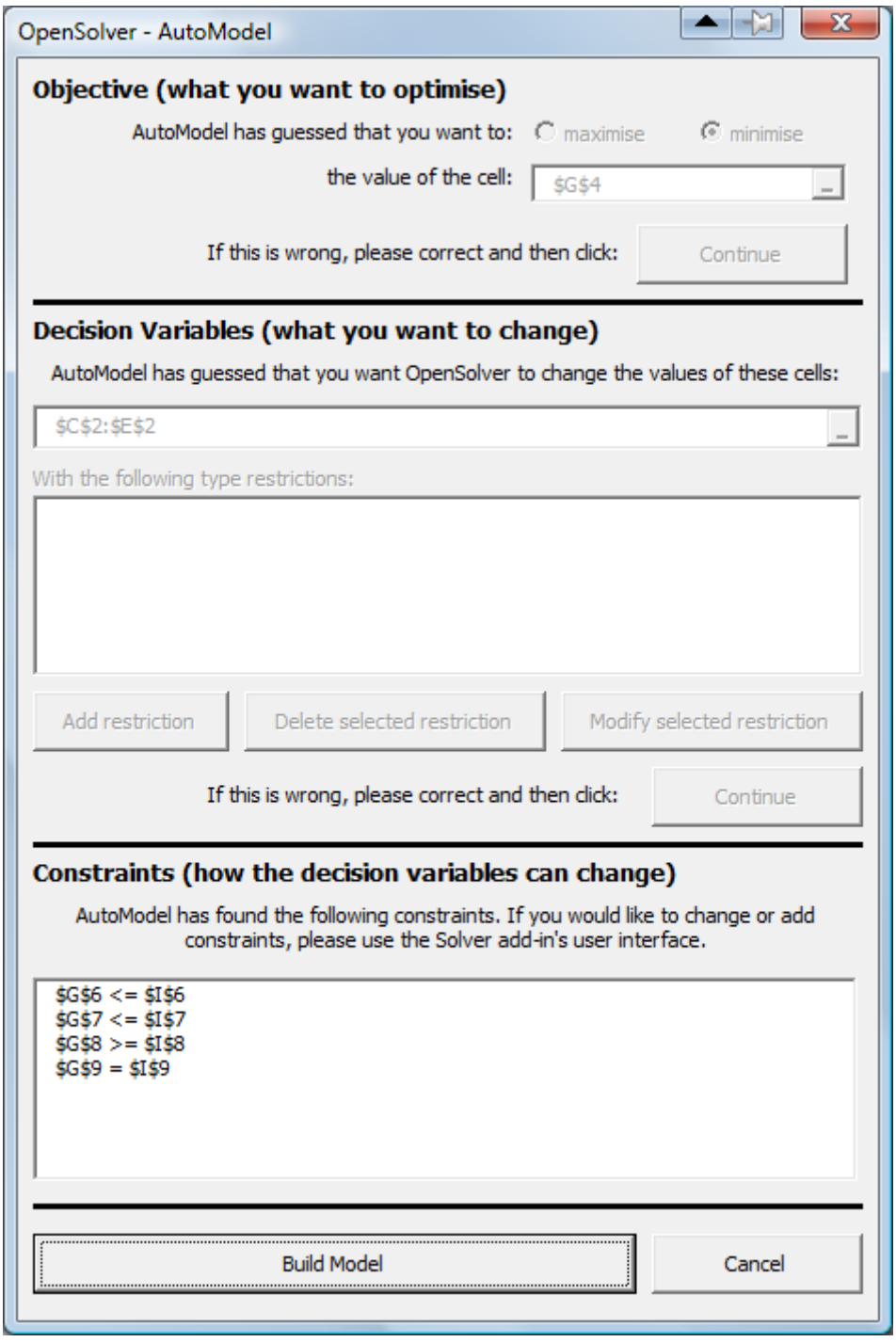


Figure 6: The auto-model dialog, shown here for the model in Figure 1, guides the user through the model building steps, allowing them to correct any mistakes made by OpenSolver's auto-model algorithms.

10 User Feedback

As at 10 November 2010, OpenSolver has been downloaded over 750 times since 16 July 2010. Since 20 June 2010, the OpenSolver web site, www.opensolver.org, has

served over 6000 page views to over 1800 visitors. Much of this interest can be attributed to OpenSolver being featured on Mike Trick's OR blog (Trick, 2010) and mentioned on the "IEOR Tools" website (Larry, 2010). It is difficult to determine how OpenSolver is being used, but a comment by Joel Sokol from Georgia Tech on the OpenSolver web site describes one use of OpenSolver as follows:

Thanks, Andrew! I'm using OpenSolver on a project I'm doing for a Major League Baseball team, and it's exactly what I needed. It lets the team use their preferred platform, creates and solves their LPs very quickly, and doesn't constrain them with any variable limits. Thanks again! - Joel Sokol, August 11, 2010

11 Conclusions

We have shown that it is possible to leverage the COIN-OR software to provide users with a new open source option for delivering spreadsheet-based operations research solutions. While our software is compatible with Solver, it also provides novel tools for visualising and building spreadsheet models that we hope will benefit both students and practitioners.

References

CBC, 8 November 2010, https://projects.coin-or.org/Cbc

COIN-OR, 8 November 2010, www.coin-or.org

Dunning, I., personal communication, August 2010

FrontLine, 8 November 2010, www.solver.org

Larry, 7 July 2010, http://industrialengineertools.blogspot.com/2010/07/open-source-solver-for-excel.html

Pieterse, J.K., 6 November 2010, http://www.jkp-ads.com/officemarketplacenm-en.asp

Solver Limits, 8 November 2010, http://www.solver.com/suppstdsizelim.htm

Trick, M., 7 July 2010, http://mat.tepper.cmu.edu/blog/?p=1167

Walkenbach, J., "Excel 2010 Formulas", p75, Wiley Publishing, Indiana, 2010, online on 7 November 2010 at Google Books.

Wikipedia, "Microsoft Excel", 7 November 2010, http://en.wikipedia.org/wiki/Microsoft_Excel

Appendix 1: Solver's internal model representation under Excel 2010 and earlier versions. (This is a partial listing for Excel 2010.)

Name[1]	**Interpretation**	**Example Values**[2]
solver_lhs1, solver_lhs2, ...[5]	A range defining the left hand side of constraints 1, 2, ...,	=Sheet1!$G$6:$G$7 =Sheet1!$G$8
solver_rhs1, solver_rhs2, ...[5]	The right hand side of constraints 1, 2, ... defined by either a range or a constant value or one of the keywords: integer, binary	=Sheet1!$I$6:$I$7 =7
solver_rel1, solver_rel2, ...[5]	The nature of the constraint, being one of the following integer values: 1: <=, 2: =, 3: >=, 4: integer, 5: binary	=1 =binary
solver_num[5]	The number of constraints.	=2
solver_adj	A range defining the decision variables (termed adjustable cells)	=Sheet1!$C$2:$E$2

solver_lin[3,7]	1 if the user has ticked "Assume linear model", 2 if this is unchecked.	=1 =2
solver_neg[3]	1 if the user has ticked "Assume non-negative", 2 if this is unchecked.	=1 =2
solver_nwt	Solver's "Search" option value, 1="Newton", 2="Conjugate"	=1
solver_pre	Solver's "Precision" option value	=0.000001
solver_cvg	Solver's "Convergence" option value	=0.0001
solver_drv	Solver's "Derivative" option value, 1="Forward", 2="Central"	=1
solver_est	Solver's "Estimates" option value, 1="Tangent", 2="Quadratic"	=1
solver_itr	Solver's "Iterations" option value, the maximum number of iterations (branch and bound nodes?) Solver can run for before the user is alerted	=100
solver_scl	Solver's "Use Automatic Scaling" option value, 1=true, 2=false	=2
solver_sho[8]	Solver's "Show Iteration Results" option value, 1=true, 2=false	=2
solver_tim	Solver's "Max time" option value, the maximum number of seconds Solver can run for before the user is alerted	=100
solver_tol	Solver's "Tolerance" option value, stored as a fraction and displayed to the user as a percentage value.	=0.05
solver_typ	Objective sense, as in "Set target cell to": 1="Max", 2="Min", 3="Value of"	=2
solver_val	If solver_typ=3, this gives the target value for the objective function.	=0
solver_ver[8]	Solver version.(=3 in Excel 2010)	=3
solver_eng[8]	Engine. 1=Nonlinear, 2=Simplex, 3=Evolutionary	=2

Notes:

1: The name is local to the sheet, and so is prefixed by the sheet name, e.g. Sheet1!solver_lhs1.
2: The value always begins with an "=", e.g. "=2" or "=Sheet1!$C$2:$E$2"
3: Models built using older versions of Solver may not have this defined.
4: Some values may be missing. For example, a model can be defined that has no decision variables.
5: solver_num may be less than the apparent number of constraints. This occurs when constraints are deleted because Solver does not delete the unused solver_lhs, solver_rhs and solver_rel names.
6: References such as =Sheet1!#REF! can occur in the current model if cells have been deleted. These are detected by OpenSolver.
7: Excel 2010 removes the "Assume linear model" option, and instead asks the user to choose the specific solver (linear solver, non-linear solver or genetic algorithm).
8: Not defined in versions before Excel 2010