Vol. 7, No. 2, January 2007, pp. 178–184
ISSN 1532-0545 | 07 | 0702 | 0178

INFORMS Transactions on Education

DOI 10.1287/ited.7.2.178
© 2007 INFORMS

Lessons from Modeling Sudoku in Excel

Howard J. Weiss
Department of Management Science/Operations Management, Fox School of Business, Temple University, Philadelphia, Pennsylvania 19122, USA, hweiss@temple.edu

Rasmus A. Rasmussen
Molde University College, 6402 Molde, Norway, Rasmus.Rasmussen@hiMolde.no

In a previous paper, Chlond (2005) presented the formulation of Sudoku as an integer program. Chlond claims that a spreadsheet formulation is not straightforward but we present a simple Excel formulation. In light of the current use of Excel in the classroom it can be highly instructive to formulate the problem using Excel. In addition, the formulation of this relatively simple model enables instructors to introduce students to the capability of programming Solver using VBA. In a follow-on paper (Rasmussen and Weiss 2007), we demonstrate advanced lessons that can be learned from using Premium Solver's powerful features to model Sudoku.

1. Introduction

Recently, in this journal, Chlond (2005) presented an integer programming formulation of the popular puzzle, Sudoku. If anything, Sudoku has become even more popular since Chlond's paper appeared. In addition to the many books on Sudoku and the many newspapers that publish daily Sudoku puzzles, it is interesting to note that *The Mathematical Association of America* published an article on Sudoku (Wilson 2006) in a recent newsletter and *The American Scientist* also recently published an article about Sudoku (Hayes 2006). Neither of these articles which appeared in a mathematical and scientific publication mentions integer programming directly. Wilson makes no mention of any formulation while Hayes refers to the puzzle as a "constraint-satisfaction problem." In addition, neither article mentions the possibility of using Excel to solve Sudoku puzzles.

Several Operations Researchers have written papers promoting the use of Excel in the classroom. For example, see Winston (1996) for one of the earlier papers or Ragsdale (2001) for a more recent paper. In addition, the current popularity of Excel in the Management Science/Operations Research (MSOR) classroom is evidenced by the number of MSOR textbooks that have been written explicitly for use with Excel. The website for Excel's Solver, http://www.solver.com/academic2.htm, lists 14 textbooks that include the Premium Solver for Education. Many other textbooks include Excel to a lesser extent. In light of the popularity of modeling in Excel and the popularity of using Solver it is highly instructive to express the popular Sudoku puzzle as an Excel Solver model.

Chlond used Mosel to formulate the problem, and writes

> When problems requiring multiple subscripts, as in these cases, are modeled effectively, the activity of converting mathematical formulations into working models using languages such as Mosel and AMPL represents a straightforward exercise. This is not always the case when building spreadsheet versions. Indeed, in many cases, a spreadsheet implementation provides an equal or perhaps even greater challenge than that provided by the actual formulation. This problem becomes especially acute as the number of subscripts required by the model increases.

It is correct that as the number of subscripts increases, modeling in a spreadsheet becomes more difficult. Fortunately, we can successfully model Sudoku with three subscripts rather than five and this will make formulation in Excel as easy as in Mosel. Our aim in this paper is not to compare modeling languages with Excel but simply to demonstrate that we can teach our students valuable lessons about modeling in Excel through the use of Sudoku.

In the next section we present a 3-dimensional model. Afterwards, we rearrange the spreadsheet to make it easier to enter the basic puzzle constraints into Solver. We then demonstrate how to include relatively simple VBA code for using Solver's subroutines to add constraints for the numbers that define a specific Sudoku instance.

2. A 3-Dimensional Formulation

Sudoku is usually a 9 by 9 puzzle as exemplified by Chlond's example displayed in Table 1a. It is easier

Table 1 **Sample Sudoku Puzzles**

a. Standard 9 by 9 puzzle

5		8			7		4	
4		2	9		5		3	
				1				
1	9							
			1		8			
							5	3
				4				
	5		6		9	4		7
	3		5			8		9

b. Smaller 4 by 4 puzzle

		1	
	3		
		4	
	1		

c. The 6 by 6 puzzle with non square interior blocks.

3			4		2
	2		6		
1					6
5					3
		3		6	
6		5			1

to present the spreadsheet formulation with a smaller puzzle, such as the one displayed in Table 1b. Some authors, such as Hayes (2006), refer to the 9 by 9 problem as order-3 and the 4 by 4 puzzle as order-2 for obvious reasons. Not all Sudoku puzzles have square interior blocks. For example, USA Today publishes both traditional 9 by 9 puzzles and 6 by 6 puzzles with 2 by 3 interior blocks as in Table 1c.

In this paper we consider only square interior puzzles and let n refer to the number of rows and columns in the problem. All concepts in this paper are easily transferred from 4 by 4 puzzles to 9 by 9 or any other sized Sudoku puzzle. We demonstrate this with the inclusion of the model for a 6 by 6 problem in the Excel file that accompanies this paper.

Lesson 1: Formulation Matters

Chlond observes that dimensionality can be a problem when formulating problems in Excel. The difficulty arises because of the 5-dimensional definition of the problem given by Chlond. Reducing the number of dimensions leads to a relatively simple Excel formulation.

Let $x_{i,j,k} = 1$ if the cell in row i and column j of the puzzle has the value k, where $i, j, k = 1, 2, \ldots, n$. Because this model is 3-dimensional it is easy to represent this problem in a spreadsheet.

Chlond lists the four Sudoku conditions:

Each cell contains a single integer.

Each integer appears only once in each (2 by 2) or (3 by 3) grid.

Each integer appears only once in each row.

Each integer appears only once in each column.

Note that "appears only once" can be replaced by "appears at least once" for Sudoku. The subtle difference between the two wordings is that we can use either strict equality constraints or inequality (">=") constraints.

Each condition generates n^2 constraints for a total of $4n^2$ constraints. The total number of (binary) variables is n^3. Incidentally, in *American Scientist* Hayes writes that "In one obvious encoding there are 810 constraints in an order-3 grid" whereas Chlond's formulation and our formulation have only 324 constraints.

The Excel spreadsheet that contains a sample 4 by 4 puzzle and its solution is displayed in Figure 1. Please note how simple the spreadsheet is.

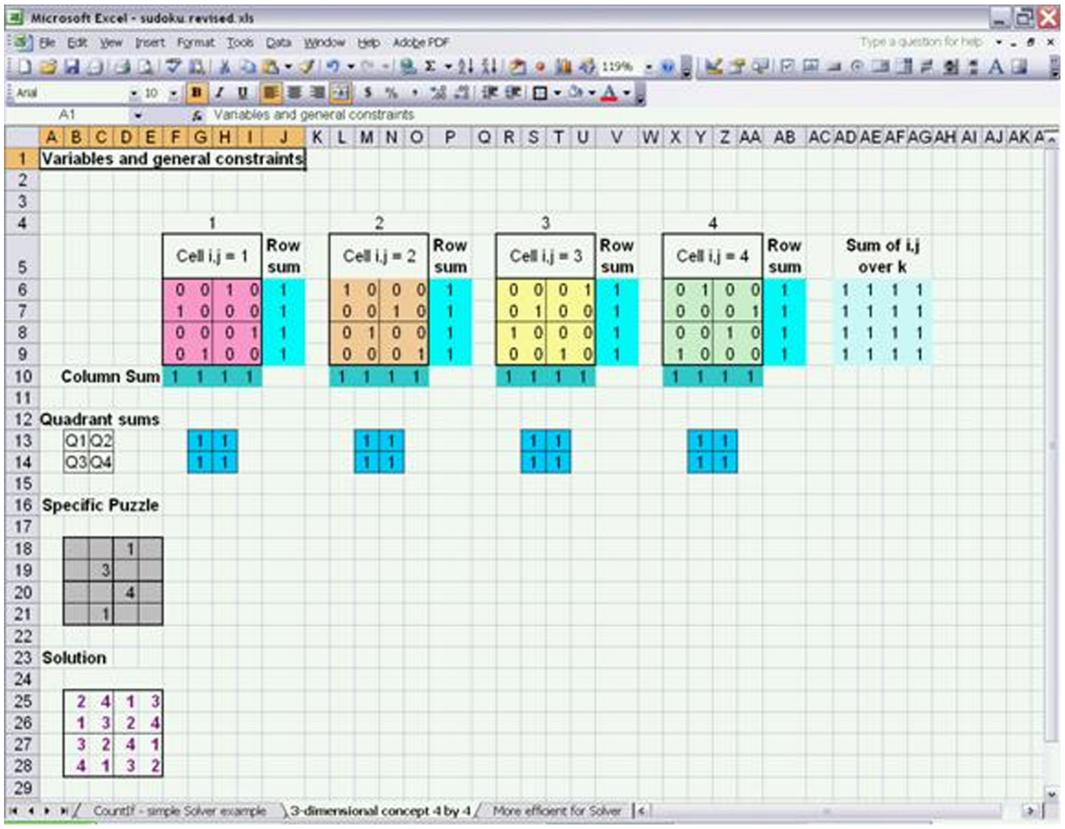
Lesson 2: Colors Can Be Used to Display the Third Dimension

The cells shaded in pink, orange, yellow, and green represent the 64 variables for the 4 by 4 example. Given the variable definition, x_{ijk}, the first dimension is the row, the second dimension is the column, and the third dimension refers to whether that entry is 1, 2, 3, or 4. Using different colors enables us to easily identify the third dimension. In addition, the colors make the spreadsheets that follow more readable. The pink variables will be 1 if the corresponding element in that cell in the 4 by 4 solution table in B25:E28 is 1. The orange variables will be 1 if the corresponding element in the 4 by 4 solution table is 2 and so on. For example, in the yellow table which represents values of 3, cells $(1,4)$, $(2,2)$, $(3,1)$, and $(4,3)$ are 1 and the corresponding elements of the solution table all have the value of 3.

The cells shaded blue represent the four constraint types and are entered as shown in Table 2. We have used different shades of blue to distinguish among the constraint types and to aid in understanding the transition to the spreadsheet that follows in Figure 2. The Excel formulas used for the spreadsheet in Figure 1 appear in Table 2.

Thus, the basic constraints have been created and are very easily read and understood in the Excel formulation.

Figure 1 An Excel Spreadsheet for the 4 by 4 Sudoku Puzzle Modeled with 3-Dimensional Variables


Microsoft Excel - sudoku revised.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Type a question for help
Arial
10
119%
A1
Variables and general constraints
CountIf - simple Solver example
3-dimensional concept 4 by 4
More efficient for Solver

Variables and general constraints

	1					2					3					4					Sum of i,j over k			
	Cell i,j = 1				Row sum	Cell i,j = 2				Row sum	Cell i,j = 3				Row sum	Cell i,j = 4				Row sum				
	0	0	1	0	1	1	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	1	1
	1	0	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	1	1	1	1	1	1
	0	0	0	1	1	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1	1	1	1	1
	0	1	0	0	1	0	0	0	1	1	0	0	1	0	1	1	0	0	0	1	1	1	1	1
Column Sum	1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1					

Quadrant sums

Q1	Q2	1	1	1	1	1	1	1	1
Q3	Q4	1	1	1	1	1	1	1	1

Specific Puzzle

		1	
	3		
		4	
	1		

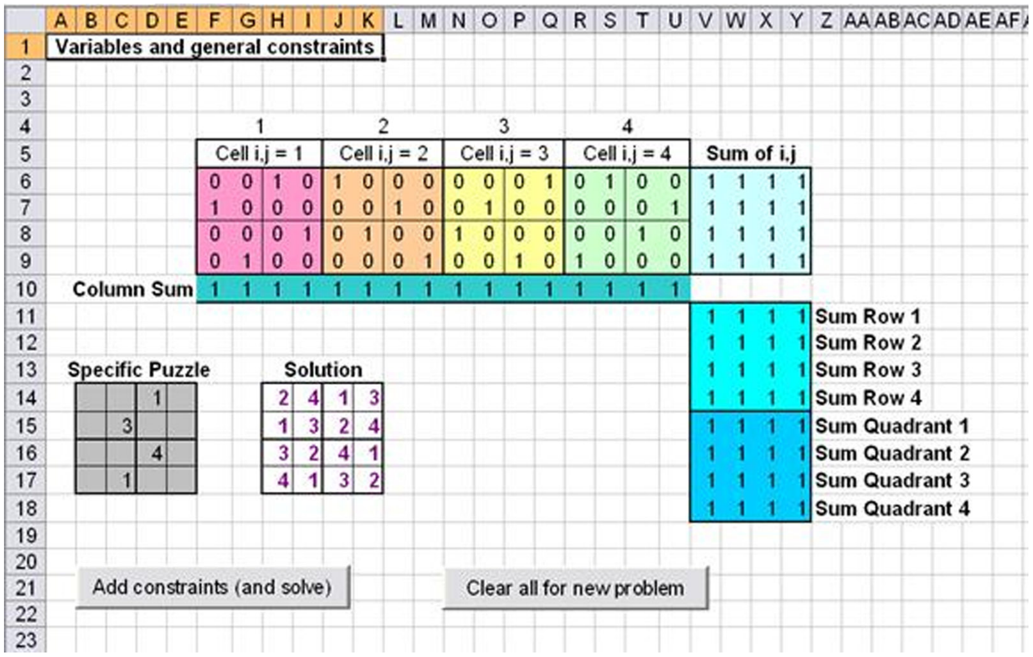
Solution

2	4	1	3
1	3	2	4
3	2	4	1
4	1	3	2

Table 2 Steps Used to Create the Basic Puzzle Constraints

	Constraint Type	Cell	Formula	Copied to
a)	Each cell contains a single integer	AD6	=F6+L6+R6+X6	AD6:AG9
	The sum in cell AD6 will be one if and only if exactly one of the variables, x_{ij1} (cell F6) or x_{ij2} (cell L6) or x_{ij3} (cell R6) or x_{ij4} (cell X6) has a value of 1. The 16 cells in AD6:AG9 will be used to test that each cell in the puzzle contains a single integer.			
b)	Each integer appears once in each grid	G13	=SUM(F6:G7)	
		H13	=SUM(H6:I7)	
		G14	=SUM(F8:G9)	
		H14	=SUM(H8:I9)	
		G13:H14		M13, S13, Y13,
	The sums in each of the 16 cells in rows 13 and 14 will be used to test that each integer appears in each 2 by 2 grid.			
c)	Each integer appears once in each row	J6	=SUM(F6:I6)	J7:J9
		J6:J9		P6, V6, AB6
	The 4 sums to the right of each of the 4 tables will be used to test that each integer appears in each row.			
d)	Each integer appears once in each column	F10	=SUM(F6:F9)	G10:I10
		F10:I10		L10, R10, X10
	The 16 sums below the tables in row 10 will be used to test that each integer appears in each column			

Figure 2 More Efficient Solver Formulation Due to Constraint Adjacency



Variables and general constraints

	1				2				3				4								
	Cell i,j = 1				Cell i,j = 2				Cell i,j = 3				Cell i,j = 4				Sum of i,j				
	0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	
	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	1	1	
	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	1	1	1	1	
	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	1	1	1	
Column Sum	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					
																	1	1	1	1	Sum Row 1
																	1	1	1	1	Sum Row 2
																	1	1	1	1	Sum Row 3
																	1	1	1	1	Sum Row 4
																	1	1	1	1	Sum Quadrant 1
																	1	1	1	1	Sum Quadrant 2
																	1	1	1	1	Sum Quadrant 3
																	1	1	1	1	Sum Quadrant 4

Specific Puzzle

		1	
	3		
		4	
	1		

Solution

2	4	1	3
1	3	2	4
3	2	4	1
4	1	3	2

The last step is to convert the Integer Program solution as given in the 64 pink, orange, yellow and green decision variable cells to the required 4 by 4 Sudoku grid. This is done by entering the formula

=F6*$F$4+L6*$L$4+R6*$R$4+X6*$X$4

into cell B25 and copying this formula to the remainder of the solution cells (B25:E28).

This formula will select the 1 in cell F4 or the 2 in cell L4 or the 3 in cell R4 or the 4 in cell X4 depending on which of the variables representing the upper left hand entry in the four blocks of decision variables equals one.

The next step is to enter the 64 structural constraints in Solver and add the individual constraints imposed by the specific puzzle that is displayed in cells B18:E21 in Figure 1.

3. Preparing for Solver

Solver allows us to enter multiple constraints at one time (on one line) if the cells are adjacent. Therefore, it is useful to modify the current spreadsheet by moving the cells so that the (blue) constraint cells form contiguous blocks whenever possible.

Lesson 3: Excel Models Can Be Modified to Improve Ease of Use with Solver

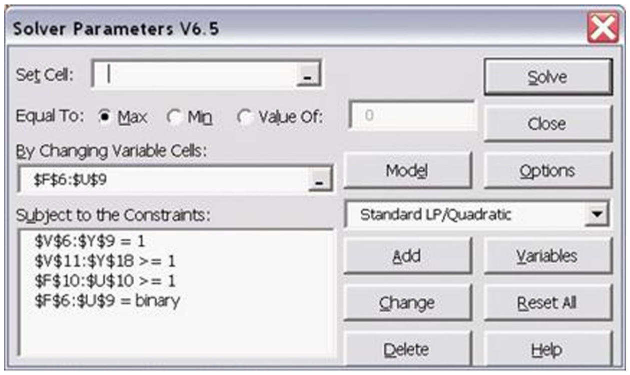
We have rearranged cells in the spreadsheet to make it easier to enter the problem into Solver. The modified spreadsheet is displayed in Figure 2. The color-coding and labeling should make it very easy to understand that this formulation and the original 3-dimensional formulation in Figure 1 are identical.

The solver input is displayed in Figure 3. The objective is left blank since the goal is simply to find a feasible solution.

The variables are easily entered into Solver as F6:U9. Note that rearranging the variables to form a single contiguous block makes it is easier to enter the variables into Solver.

We have used three lines in Solver to represent the basic constraints. The first constraint (V6:Y9) is used to ensure that each cell has exactly one integer. The second line (V11:Y18) is used to ensure that each row and each grid contains a 1, 2, 3, and 4 while the third line (F10:U10) is used to ensure that every column contains a 1, 2, 3, and 4. The original formulation of this 4 by 4 puzzle displayed in Figure 1 would have

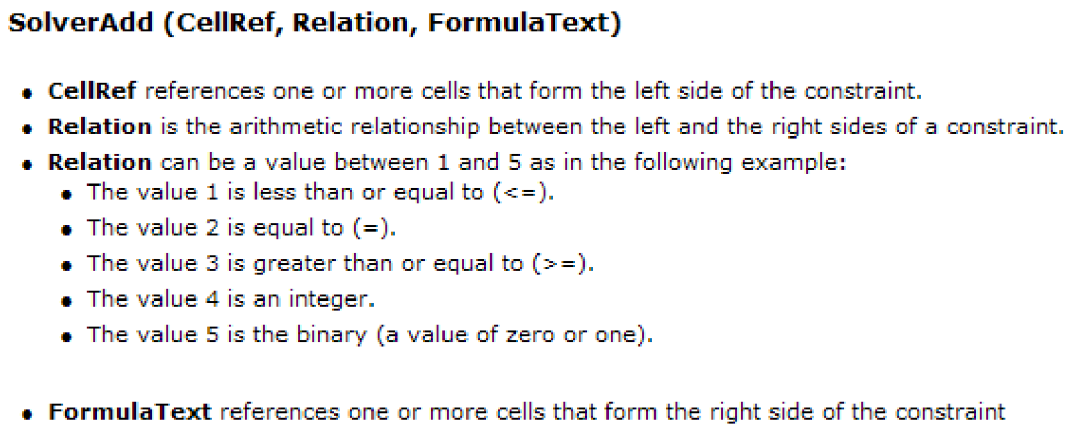
Figure 3 Solver Parameters for the 4 by 4 Puzzle



Solver Parameters V6.5

Set Cell:
Equal To: [✓ Max] Min Value Of: 0
By Changing Variable Cells: $F$6:$U$9
Subject to the Constraints:
$V$6:$Y$9 = 1
$V$11:$Y$18 >= 1
$F$10:$U$10 >= 1
$F$6:$U$9 = binary
Standard LP/Quadratic

Figure 4 Adding Solver Constraints Using VBA



SolverAdd (CellRef, Relation, FormulaText)

- **CellRef** references one or more cells that form the left side of the constraint.
- **Relation** is the arithmetic relationship between the left and the right sides of a constraint.
- **Relation** can be a value between 1 and 5 as in the following example:
  - The value 1 is less than or equal to (<=).
  - The value 2 is equal to (=).
  - The value 3 is greater than or equal to (>=).
  - The value 4 is an integer.
  - The value 5 is the binary (a value of zero or one).
- **FormulaText** references one or more cells that form the right side of the constraint

required 13 lines of constraints for these constraints rather than 3.

Lastly, the fourth constraint defines all of the variables (F6:U9) as binary. For the original spreadsheet displayed in Figure 1 we would need 4 such constraints.

The only constraints missing are the constraints for the specified elements of the individual puzzle.

4. Entering the Specific Problem—Programming Solver

One option for entering the specific problems is to enter the constraints one at a time by hand into Solver. Alternatively, this is an excellent opportunity to explain how to program Solver in VBA.

Lesson 4: It Is Easy and Useful to Use VBA to Program Solver

In order to access Solver's subroutines in VBA they must be referenced. Open the VBA editor (Tools, Macro, Visual Basic Editor) and then in VBA use Tools, References and check the SOLVER option. Albright (2007) has an extensive discussion about calling Solver from VBA.

The syntax for Solver's subroutines can be found at Microsoft's web site, "How to create Visual Basic macros by using Excel Solver in Excel 97" at http://support.microsoft.com/kb/843304. In spite of the Excel 97 term in the title of the web site the document was revised as recently as August 2005 and the commands work for the latest version of Excel.

The command for adding a constraint is given at this site by the syntax shown in Figure 4.

The five relations displayed in Figure 4 are available for the basic Solver that ships with Excel. Note that Premium Solver has additional relations available and we will take advantage of this in our follow-on paper.

We have added a button that will add the constraints for the specific problem using VBA code. The code for adding the constraints is very similar to and only slightly more difficult than Chlond's Mosel code and is displayed in Figure 5. The major portion of code that is equivalent to Chlond's code is displayed in boldface between the two dashed lines.

PuzRowOffset and PuzColOffset are VBA constants that are used to indicate the puzzle's position in the spreadsheet while varRowOffset and varColOffset are used to indicate the location of the variables in the spreadsheet.

The code to solve a problem rather than needing to click on Tools, Solver, Solve is given simply by **SolverSolve** which we have added to the subroutine.

The spreadsheet also includes a button to clear the problem so that the spreadsheet can be used repeatedly for different problems. The added code uses the **SolverReset** and **SolverOK** subroutines. The syntax for these commands can be found on the Microsoft site or in the Premium Solver Platform User Guide (Anonymous 2006b) and the implementation can be seen in the accompanying Excel workbook.

With the use of the buttons, a student simply needs to enter the puzzle into cells B14:E17, press the "Add constraints (and solve)" button and then read the solution message from Solver to ensure that it states "Solver found a solution." In the event that a puzzle is entered that has no feasible solution Solver will report, "Solver could not find a feasible solution."

5. The 9 by 9 Puzzle

The spreadsheet for the 9 by 9 puzzle is displayed in Figure 6. While in the figure below the cells are largely unreadable it should be easily seen due to the color coding of the cells that the structure of this table is identical to the structure of the 4 by 4 example. That is, the spreadsheet contains nine 9 by 9 squares for the variables, 81 column sums below the variables to ensure that each integer appears once in each column of the solution, 81 row sums to ensure that

Figure 5 **VBA Code for Creating Constraints for a Specific Puzzle and Solving**

```
Private Sub CommandButton1_Click()

Const puzSize% = 4
Const puzRowOffset% = 13 'Puzzle starts in the row below this
Const puzColOffset% = 1  'Puzzle starts in the column after this
Const varRowOffset% = 5  'The variables begin in the row below this
Const varColOffset% = 5  'The variables begin in the column after this

Dim k%                  ' Used to pick up the cells that have been specified
Dim conRow%, conCol%     ' This is the variable cell that corresponds to the constraint

resetSolver 'in case the user did not push the reset button, reset the basic Solver conditions

'----------------------------------------------------------------------------------------------
For i = 1 To puzSize% 'loop through the puzzle
  For j = 1 To puzSize%
    k = Cells(puzRowOffset% + i, puzColOffset% + j) ' k is the value that was specified
    If k > 0 Then                              ' a value has been specified
      conRow% = varRowOffset% + i
      conCol% = varColOffset% + (k - 1) * puzSize% + j ' move to the right by k-1 tables
                                              ' worth of columns
      z = SolverAdd(Cells(conRow%, conCol%), 2, "=1")  '2 is for "="
    End If
  Next j
Next i
'----------------------------------------------------------------------------------------------
SolverSolve
End Sub
```

Figure 6 **The 9 by 9 Puzzle**

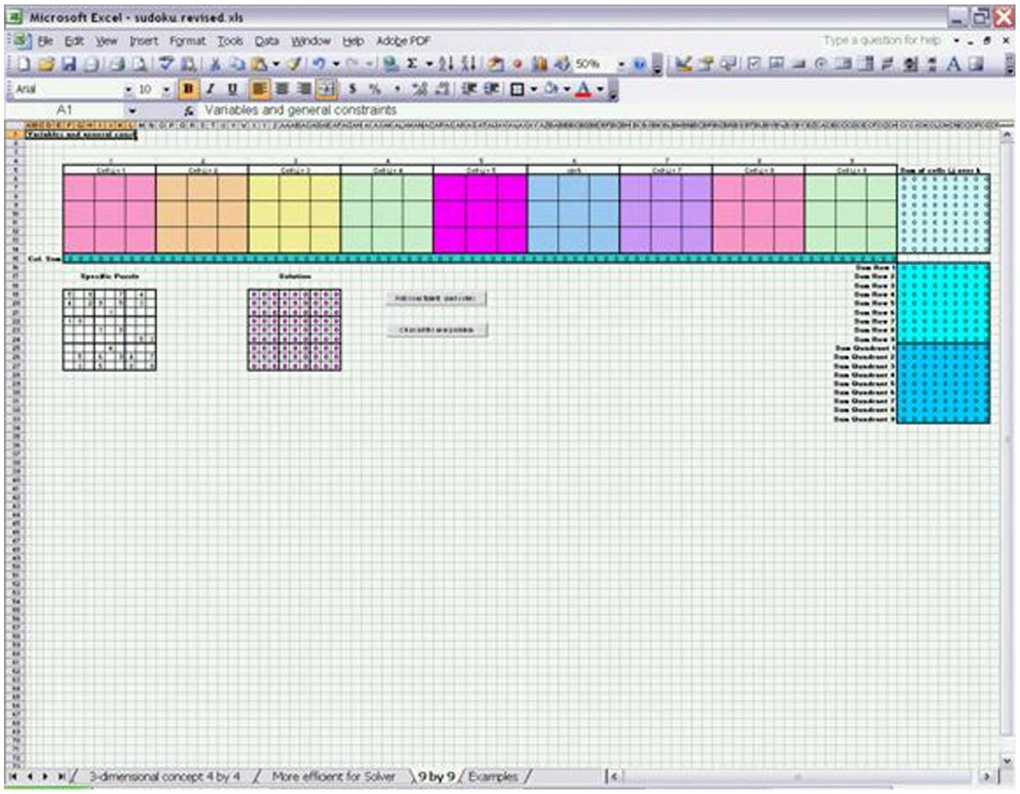
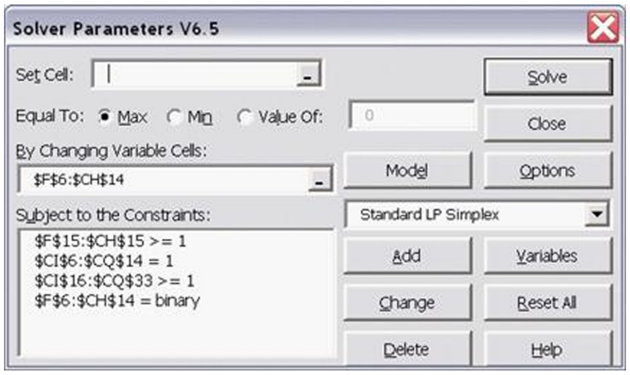

Microsoft Excel - sudoku revised.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Type a question for help
Arial
10
50%
A1
Variables and general constraints
Specific Puzzle
Solution
3-dimensional concept 4 by 4
More efficient for Solver
9 by 9
Examples

Figure 7 Solver Parameters for the Basic 9 by 9 Puzzle


Solver Parameters V6.5

Set Cell: |

Equal To: (● Max) (○ Min) (○ Value Of:) 0

By Changing Variable Cells:
$F$6:$CH$14

Subject to the Constraints:
$F$15:$CH$15 >= 1
$CI$6:$CQ$14 = 1
$CI$16:$CQ$33 >= 1
$F$6:$CH$14 = binary

Standard LP Simplex

each variable appears once in each row and 81 quadrant sums to ensure that each integer appears once in each 3 by 3 quadrant. All that remains is to create the table of sums on the top right that guarantee that each cell contains only one integer, by entering the formula for the upper left cell (CI6) and copying it to the remaining 80 cells.

In terms of preparing the problem for solver there now are still only 3 lines required to represent the basic constraints of the 9 by 9 puzzle. The basic Solver parameters without the constraints for the specific puzzle are displayed below.

The last aspect is to add the constraints for the specific puzzle and solve the problem. The code for the button for adding constraints is identical to the code for the 4 by 4 puzzle except that the constants for the puzzle row and column need to be changed, and the constant for the puzzle size must be changed to 9. Obviously this is easy.

Excel's standard solver is limited to 200 variables and the 9 by 9 problem has 729 variables. The 6 by 6 problem has 216 variables. Therefore, for either of these problems, a premium solver is required. Unfortunately, the required Solver is not the Premium Solver for Education (PSE) that ships with several textbooks because it also has a restriction of 200 variables. Fortunately, a 15-day version of Premium Solver Version 7.0 (the professional version rather the educational version) can be easily obtained by registering at http://www.solver.com/ and then downloading the free trial version. (An academic license can be purchased for $225.)

The 4 by 4, 9 by 9, and 6 by 6 spreadsheets have successfully solved all puzzles that we have attempted.

6. Conclusions

Students can learn several lessons from Sudoku. Chlond has demonstrated that the puzzle can be modeled as a linear (binary) integer program. In a different context, Koch (2005) writes, "Choosing the right formulation is often more important than having the best solver algorithm" and this is exemplified by the usage of the 3-dimensional model in this paper versus the 5-dimensional model in Chlond's paper. In addition this paper demonstrates the value of color for representing three-dimensional problems in Excel, and the ease of use of Solver's subroutines in VBA. In a follow-on paper we will use Sudoku to introduce students to advanced features of Premium Solver.

Acknowledgments

We are extremely grateful to the editor and referees for their comments.

References

Albright, S. C. 2007. *VBA for Modelers—Developing Decision Support Systems Using Microsoft Excel*, 2nd ed. Duxbury.

Anonymous. 2006a. Microsoft, How to create Visual Basic macros by using Excel Solver in Excel 97. http://support.microsoft.com/kb/843304 (last accessed on May 11, 2006).

Anonymous. 2006b. Premium Solver Platform User Guide, Frontline Systems. http://www.solver.com/supp_pspguide70.php (login required) (last accessed on Nov. 3, 2006).

Chlond, M. J. 2005. Classroom exercises in IP modeling: Sudoku and the log pile. *INFORMS Trans. Ed.* **5**(2), http://ite.pubs.informs.org/Vol5No2/Chlond/ (last accessed on May 11, 2006).

Hayes. 2006. Unwed numbers. *American Scientist* **94**(1) 12–15, Also at http://www.americanscientist.org/template/AssetDetail/assetid/48550?&print=yes (last accessed on May 11, 2006).

Koch, T. 2005. Rapid mathematical programming or how to solve sudoku puzzles in a few seconds. Konrad-Zuse-Zentrum fur Informationstechnik Berlin, ZIB Report 05-51, http://www.zib.de/Publications/Reports/ZR-05-51.pdf (last accessed on May 11, 2006).

Ragsdale, C. T. 2001. Teaching management science with spreadsheets: From decision models to decision support. *INFORMS Trans. Ed.* **1**(2), http://ite.pubs.informs.org/Vol1No2/Ragsdale/ (last accessed on May 11, 2006).

Rasmussen, R. A., H. J. Weiss. 2007. Advanced lessons on the craft of optimization modeling based on modeling sudoku in excel. *INFORMS Trans. Ed.* To appear.

Wilson, R. 2006. The sudoku epidemic. *Focus* **26**(1) 5–7, (Also at http://www.maa.org/pubs/jan06web.pdf) (last accessed on May 11, 2006).

Winston, W. 1996. The teacher's forum: Management science with spreadsheets for MBA's at Indiana university. *Interfaces* **26**(2) 105–111.