Journal of Statistical Software

May 2005, Volume 14, Issue 4. *http://www.jstatsoft.org/*

Calling the `lp_solve` Linear Program Software from R, S-PLUS and Excel

Samuel E. Buttrey
Naval Postgraduate School

Abstract

We present a link that allows R, S-PLUS and Excel to call the functions in the `lp_solve` system. `lp_solve` is free software (licensed under the GNU Lesser GPL) that solves linear and mixed integer linear programs of moderate size (on the order of 10,000 variables and 50,000 constraints). R does not include this ability (though two add-on packages offer linear programs without integer variables), while S-PLUS users need to pay extra for the **NuOPT** library in order to solve these problems. Our link manages the interface between these statistical packages and `lp_solve`.

Excel has a built-in add-in named **Solver** that is capable of solving mixed integer programs, but only with fewer than 200 variables. This link allows Excel users to handle substantially larger problems at no extra cost. While our primary concern has been the Windows operating system, the package has been tested on some Unix-type systems as well.

Keywords: linear programming, **lpSolve**, R, S-PLUS, Excel.

1. Introduction

1.1. The Excel Solver and its limitations

The spreadsheet program Excel for Windows (see, e.g., Walkenbach 1999) comes supplied with an add-in named **Solver** that performs numerical optimization including linear and integer programming. However, **Solver** can handle only comparatively small problems (200 "adjustable cells," according to the on-line help for Excel 2002). Additional "add-ins" that allow the solution of larger problems are available for purchase. This paper addresses a need for a free solver to handle linear or mixed integer programs of substantial size (on the order of 10,000 variables and 50,000 constraints). We use the free software `lp_solve` due to a team including Berkelaar, Dirks, Eikland, Notebaert (see the newsgroup and ftp site at http://groups.yahoo.com/group/lp_solve), re-compiled into a dynamic linked library (DLL) using a free compiler and development environment. (The approach in this paper uses lp_solve version 4.0, but newer versions of our technique calling version 5 are available.) This DLL works together with an interface written in Visual Basic for Applications on the Excel end and in C on the DLL end to allow calls to the lp_solve application programming interface (API). (We refer to DLLs because our concern is primarily for the Windows environment, where Excel runs. However our R software has also been shown to work on Unix-style systems when compiled into a shared library. We will refer to DLLs in the remainder of this article.)

The same DLL and interface allow calls to lp_solve from the statistical environments S-PLUS (Insightful Corporation 2001) and R (R Development Core Team 2005, http://www.R-project.org/). The latter is another piece of freeware. Neither of the two packages comes with a built-in mixed integer program solver, although a library for S-PLUS called **NuOPT** is available for an additional price. The solveLP function in the **linprog** package, and the simplex function in the **boot** package, solve linear (but not integer) programs in R. lp_solve is licensed under the GNU Lesser General Public License; see the documentation for specific terms of the licensing agreement.

1.2. DLLs and the development environment

For this project we used the Cygwin development environment (Cygwin project, http://www.cygwin.com/). This environment supplies a set of Unix-like tools to the Windows application developer. In particular, it allows the use of the free GNU compiler gcc (GNU project, http://gcc.gnu.org/). Through this environment the developer is spared the necessity of having to purchase a commercial development environment. An alternative version of the gcc compiler, known as MinGW-gcc (http://www.mingw.com/), works as well and is perhaps slightly easier to use.

The gcc compiler and its associated tools allow the production of DLLs. The DLL contains the essential instructions associated with the program. The DLL acts as a "library" that can be attached to the main program (Excel, for example) and whose member functions can be called as needed.

2. The pieces of the link

2.1. Description

The link that we have constructed consists of three parts. The first is Visual Basic for Applications (VBA) code that is included in the Excel add-in. This code includes a form that allows the user to specify the objective function, constraints, integer variables, and the place for the results to appear. It then declares and calls the second part, a C function (lpslink) that acts as the interface between the VBA call and the functions exposed by the lp_solve API.

Not technically part of the link is the DLL that allows lpslink to call the functions in lp_solve. Although the lp_solve distribution includes a DLL, we needed to re-build the DLL in order to include our link function. (The link is necessary for reasons described in

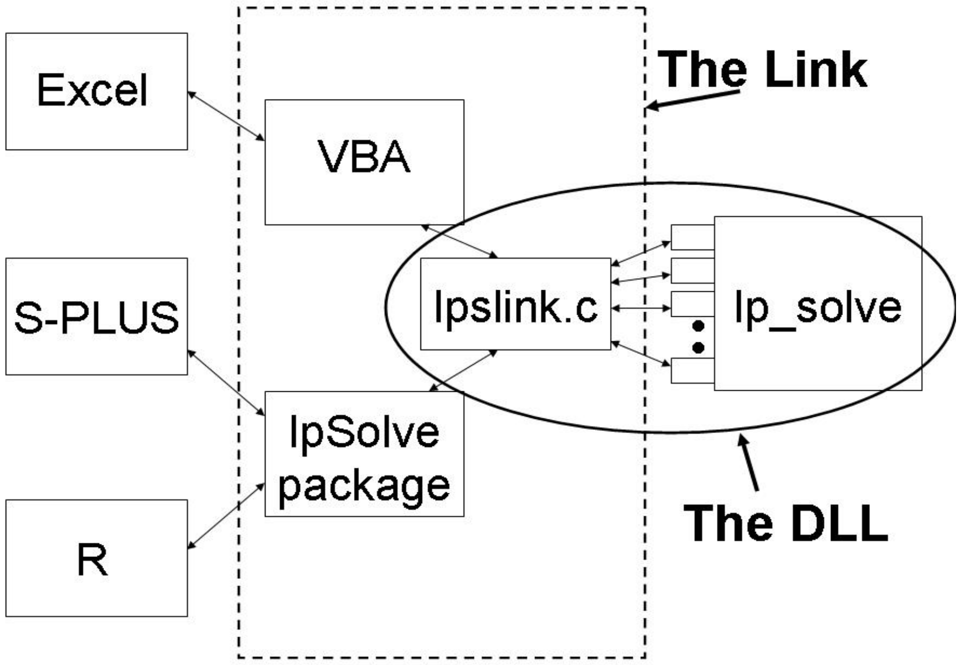


Figure 1: Schematic of link between clients (Excel, S-PLUS, R) and `lp_solve`

section 2.2 below.)

The third part of the link is the **lpSolve** package for R and S-PLUS. This is a set of four functions in the S language (the language common to R and S-PLUS) that allow the user to specify the objective function, constraints, and integer variables from within the package. These functions then handle the details of calling `lpslink` and managing the results. Figure 1 shows a schematic diagram of the link.

2.2. The C link

The C link is necessary in order to call `lp_solve` from the S language (that is, from S-PLUS or R; for brevity we will use S to mean either S-PLUS or R). This is because in some instances `lp_solve` passes function arguments by value, but when S calls a DLL all arguments must be passed by reference. Furthermore, C functions called from S cannot return values. The C link allows function calls originating in S to be properly "packaged." The link is not necessary when calling `lp_solve` from VBA, since VBA can accept return values and pass arguments in either style. Still, the C link provides a simple, one-call interface to the `lp_solve` system. Currently, the caller (VBA for Excel, the **lpSolve** package for S) provides the coefficients of the objective function, the matrix of constraints, arrays containing the directions and right-hand sides of each constraint, and a vector indicating which variables are required to be integers. Details of handling these arguments (for example, `lp_solve` requires that the objective function array have an additional leading 0) are currently managed by the caller.

Inside the C link, the code dispatches calls to some of the `lp_solve` functions. There are about 150 of these, but in the implementation presented here only nine are used. These calls create the program, set the objective function and the optimization direction, add constraints, set integer variables, solve the program, extract the results, and delete the program.

2.3. The lpSolve package for S

The **lpSolve** package contains four S functions. These are `lp` (to solve general mixed integer linear programs), `lp.assign` (to solve assignment problems), `lp.transport` (to solve transportation problems), and `print.lp` (to print the results of a linear program). These problems are defined in section 3.1 below. Examples can be found in section 4.4.

2.4. The VBA link

VBA is a dialect of Visual Basic that serves as a scripting language for all Microsoft applications (for an in-depth introduction, see Walkenbach (1999)). In the present case, its usefulness descends from its ability to handle forms and to call DLLs. The VBA interface to the link has been implemented as an Excel add-in, which means that once the add-in has been loaded, it is available from any workbook. When the add-in is invoked, it produces a simple form which allows the user to select whether the problem is a general mixed linear program, or specifically an assignment or transportation problem (these types of problem are described below). This selection produces a second form. For a general mixed integer program, the user fills in the form to describe whether the problem is a maximization or a minimization, and to give the ranges of the objective function, constraints, integer variables, and the location where results should be placed. A press of a button then produces the solution. The current interface is admittedly primitive. Figure 2 shows a screen shot of a spreadsheet; the "Choose a Problem" and "Linear/Integer Program" windows are visible.

3. Details and some technical notes

3.1. Types of problems

As Figure 2 suggests, three types of problems are currently supported in the link. The first is a general linear program. Here the objective function and constraints are laid out in rows, with each variable occupying a column. Any linear program in standard form can be represented in this way.

However, two specific types of linear programs that arise in practice are also made available. In the transportation problem, the decision variables are arranged in a rectangular matrix, say I by J, so that the decision variables can be denoted by $x_{ij}, i = 1, \ldots, I; j = 1, \ldots, J$. There is a cost associated with each of the x_{ij}, and there are constraints on row and column sums. For this problem the user needs to enter a rectangular matrix containing the decision variables, the signs of the constraints, and the constraint values, and a second matrix containing the costs. The program converts the constraints to the needed form and requires that all variables be non-negative integers.

The assignment problem is a special case of the transportation problem in which all the rows and columns are constrained to add up to one. In our implementation, the user needs to provide one matrix with decision variables and a second with cost; the program then prepares the constraints and requires that all variables be non-negative integers.

We note that by default every variable in lp_solve is constrained to be $\geq$ zero. This is not a serious restriction, because an unconstrained variable can be re-expressed as the difference between two positive variables. In the examples below, we do not explicitly state the positivity requirement.

3.2. Compilation

We created DLLs containing both the C link and the lp_solve code using the gcc compiler in the Cygwin environment. This allows other users to modify the code without having to purchase a commercial development environment. Note that by default the Cygwin compiler produces DLLs that themselves depend on another DLL, cygwin1.dll, that is installed with that environment. Since most users will not have this DLL, Cygwin gcc users should include the -mnocygwin flag and references to the MinGW libraries so that the resulting DLL can stand alone. The alternative MinGW compiler produces executables that do not depend on external DLLs and this compiler has worked for us as well.

When producing a DLL for Excel, calls to the compiler should include the -mrtd flag so that the resulting objects use the Pascal (stdcall) calling convention. This flag is included in the Makefile. The DLL for R needs to use the C calling convention, and so calls to the compiler need to omit the -mrtd flag. Although S-PLUS can seemingly use either convention, we use the Excel DLL for S-PLUS use. Users creating DLLs for R should use the Makefile.R file, which also defines a compile-time definition BUILDING_FOR_R that accounts for the fact that integer variables are passed as int * in R but as long * in S-PLUS.

3.3. Calling from S-PLUS or R

In S-PLUS 6.2 (Insightful Corporation 2001) DLLs are loaded with the dyn.open() function; dyn.load() accomplishes that task in R (we are currently using version 1.9.1). Recall that while the code used to produce the DLLs for S-PLUS and R is the same the compilation schemes (and therefore the DLLs themselves) are different. Three functions, useable in either S-PLUS or R, serve as the interface to the DLLs. The function lp() accepts, as arguments, the vector of objective function coefficients, a matrix of constraints, vectors containing the signs of the constraints and their right-hand side values, and a vector indicating which variables should be required to be integers. The return value is a list that includes (among other things) the optimal values of the decision variables and the objective function value. The related functions lp.assign() and lp.transport() handle the assignment and tranportation problems in a way analogous to the way they are handled in Excel (see section 4.4).

3.4. Calling from Excel

Calling a DLL from Excel requires two steps. First the DLL is declared, using a Declare statement in the VBA code. Second, it is called using a Call statement. The documentation suggests that arrays should be declared as arrays in the Declare statement, and passed as arrays in the Call statement, like this (where the underscore character denotes line continuation):

```
Private Declare Sub lpslink Lib "lpsolve.dll" _
```

```
(ByRef objOut() As Double, ....)
Call lpslink (objOut(), ...)
```

In fact, though, this does not seem to work. Instead, array arguments should be declared as scalars, and the call should refer to the first element of the array (element 0 unless specified otherwise).

```
Private Declare Sub lpslink Lib "lpsolve.dll" _
(ByRef objOut As Double, ....)
Call lpslink (objOut(0), ...)
```

A workbook containing examples including the ones in this document is available from `http://web.nps.navy.mil/~buttrey/Software/Lpsolve/`.

4. Examples

4.1. General integer program

An example of a simple integer program appears in Nemhauser and Wolsey (1988, p. 443). (This example was brought to our attention by the `lp_solve` documentation.) The problem is:

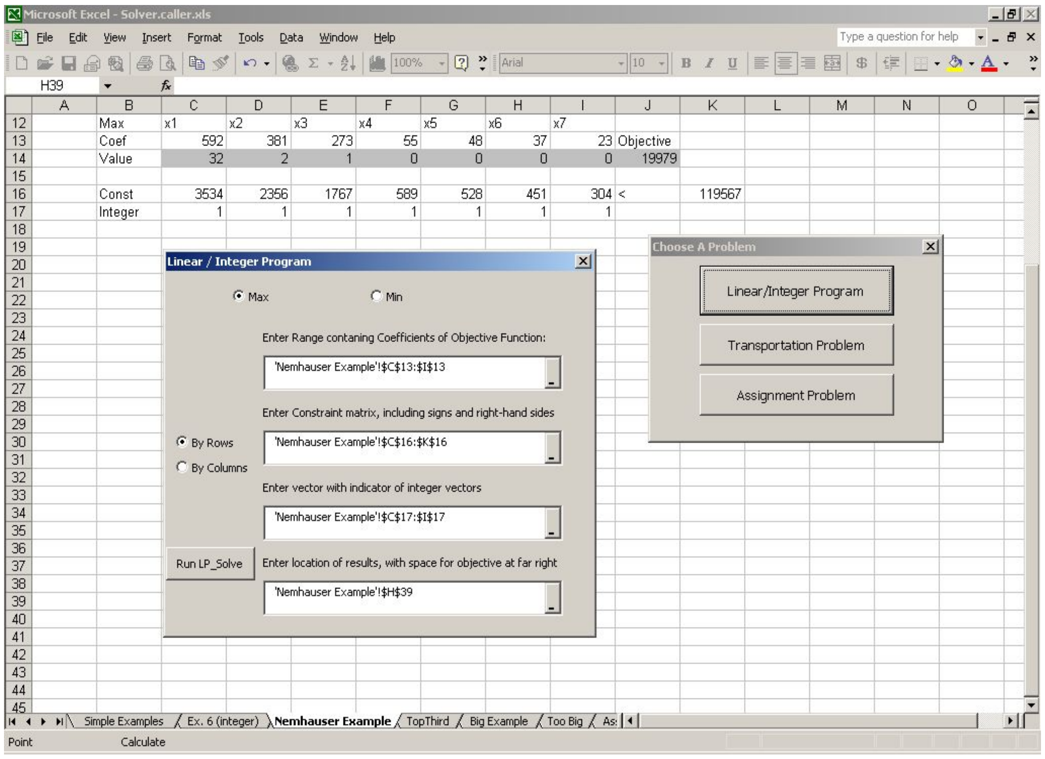
Maximize $592x_1 + 381x_2 + 273x_3 + 55x_4 + 48x_5 + 37x_6 + 23x_7$
subject to
$3534x_1 + 2356x_2 + 1767x_3 + 589x_4 + 528x_5 + 451x_6 + 304x_7 \geq 119567$
with $x_1, x_2, x_3, x_4, x_5, x_6, x_7$ all integer.

The authors say (using our notation) "[i]t is not hard to show an optimal solution is $x_1 = 33, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 0, x_7 = 0$, and that the optimal [objective function] value is 19972." This is the solution produced by Excel 97 using the default settings. In fact, though, the solution $(32, 2, 1, 0, 0, 0, 0)$ meets the constraint and produces an objective function value of 19979. This is the solution produced by current versions of Excel and by the `lp_solve` link. Figure 2 shows a screen shot of the `lp_solve` link producing the correct answer, shown in the grey cells.

At the moment the Excel link cannot handle problems with exactly one adjustable cell. Although this is not a very restrictive limitation, we will correct this in a subsequent release.

4.2. Problems too big for Excel's Solver

It is easy to construct examples too big for Excel's **Solver**. A simple one demonstrating the assignment problem is shown in Figure 3. Here there are 15 sources (say, operators) to be assigned to 15 destinations (say, jobs). Each decision variable represents the assignment of a source to a destination, so in this example there are 225 variables. There is one constraint per row to ensure that each operator is assigned exactly one job, and one contrainst per column to ensure that each job is assigned exactly one operator. Thus in this example there are 30 constraints. Figure 4 shows a screen shot of the assignment problem in the example worksheet. (The lower matrix shows the assignments, with the total cost of 24 just visible to


Microsoft Excel - Solver.caller.xls

	B	C	D	E	F	G	H	I	J	K
12	Max	x1	x2	x3	x4	x5	x6	x7		
13	Coef	592	381	273	55	48	37	23	Objective	
14	Value	32	2	1	0	0	0	0	19979	
15										
16	Const	3534	2356	1767	589	528	451	304	<	119567
17	Integer	1	1	1	1	1	1	1		

Linear / Integer Program

(● Max) ○ Min

Enter Range contaning Coefficients of Objective Function: 'Nemhauser Example'!$C$13:$I$13

Enter Constraint matrix, including signs and right-hand sides: 'Nemhauser Example'!$C$16:$K$16

(● By Rows) ○ By Columns

Enter vector with indicator of integer vectors: 'Nemhauser Example'!$C$17:$I$17

Enter location of results, with space for objective at far right: 'Nemhauser Example'!$H$39

Run LP_Solve

Choose A Problem: Linear/Integer Program | Transportation Problem | Assignment Problem

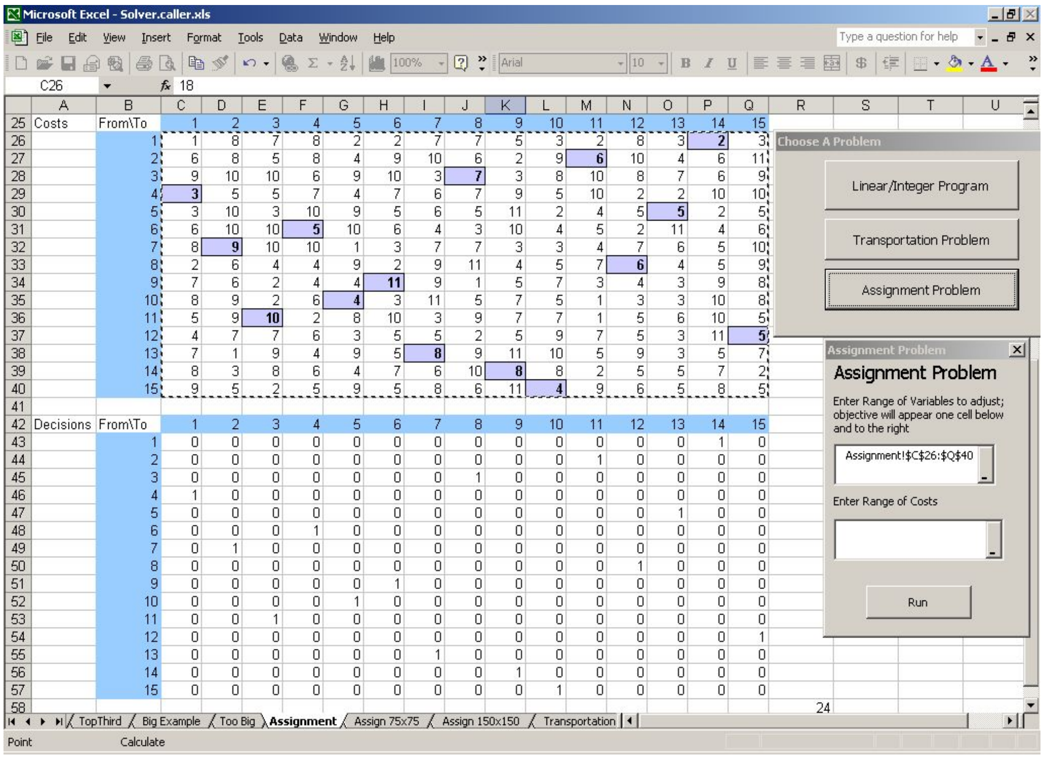
Figure 2: `lp_solve` finding the correct answer to example 1

the bottom right; the upper matrix shows the costs, with highlighted cells showing the actual assignments.)

Since there are more than 200 adjustable cells in this problem, the Excel **Solver** will not run. On a desktop computer equipped with Windows XP, a 3.6GHz processor and 3GB RAM, `lp_solve` handles this problem in very much less than a second. A 75×75 assignment problem (that is, one with 5,625 variables and 150 constraints) takes about a second; a 150×150 problem (22,500 variables, 300 constraints) takes about four seconds; and a 250×250 problem (62,500 variables, 500 constraints) takes around thirty seconds. It should be noted that the constraint matrices are typically sparse: in the 250×250 problem, the $500 \times 62,500$ constraint matrix contains $31,000,000$ zeros and $250,000$ ones. The overhead of manipulating such a matrix is of course considerable. Intelligent handling of this sort of sparsity would presumably enable `lp_solve` to solve much bigger problems, but we have not yet taken any steps in this direction.

4.3. Transportation problem

An example from Bronson (1982) demonstrates the transportation problem and its implmentation. In the transportation problem, the row and column sums are constrained, but not necessarily to be equal to one. This particular problem involves scheduling production in the garment industry. We are given demands for garments for each of the four seasons (the ”Demand” row). We are also given an initial inventory (the ”Init” row) and the ability to produce garments using regular labor (the ”Reg” rows) or overtime labor (the ”OT” rows). The costs as given in the ”Cost” matrix reflect the $7 cost per garment using regular labor



Costs	From\To	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	8	7	8	2	2	7	7	5	3	2	8	3	**2**	3
	2	6	8	5	8	4	9	10	6	2	9	**6**	10	4	6	11
	3	9	10	10	6	9	10	3	**7**	3	8	10	8	7	6	9
	4	**3**	5	5	7	4	7	6	7	9	5	10	2	2	10	10
	5	3	10	3	10	9	5	6	5	11	2	4	5	**5**	2	5
	6	6	10	10	**5**	10	6	4	3	10	4	5	2	11	4	6
	7	8	**9**	10	10	1	3	7	7	3	3	4	7	6	5	10
	8	2	6	4	4	9	2	9	11	4	5	7	**6**	4	5	9
	9	7	6	2	4	4	**11**	9	1	5	7	3	4	3	9	8
	10	8	9	2	6	**4**	3	11	5	7	5	1	3	3	10	8
	11	5	9	**10**	2	8	10	3	9	7	7	1	5	6	10	5
	12	4	7	7	6	3	5	5	2	5	9	7	5	3	11	**5**
	13	7	1	9	4	9	5	**8**	9	11	10	5	9	3	5	7
	14	8	3	8	6	4	7	6	10	**8**	8	2	5	5	7	2
	15	9	5	2	5	9	5	8	6	11	**4**	9	6	5	8	5

Decisions	From\To	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	9	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	11	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	13	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	14	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	15	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

24

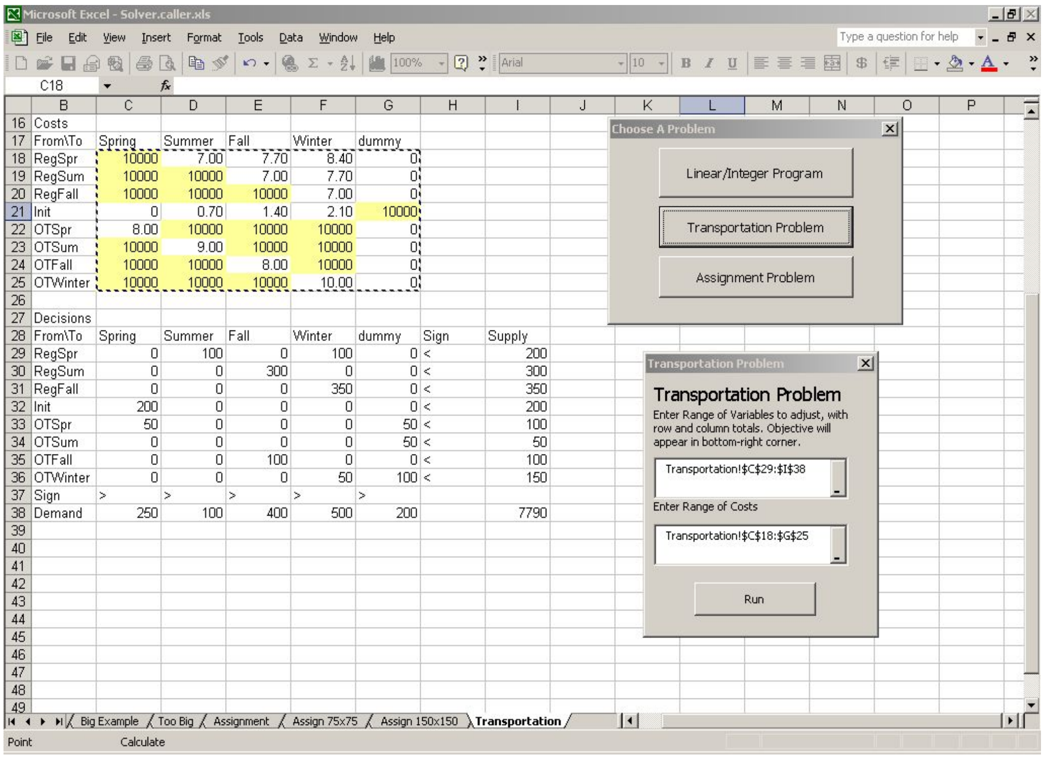
Assignment Problem

Enter Range of Variables to adjust; objective will appear one cell below and to the right

Assignment!$C$26:$Q$40

Enter Range of Costs

Figure 3: Assignment problem example. The upper matrix shows the costs (highlighted cells show the optimal assignments); the lower matrix shows the decision variables. The optimal cost of 28 is visible below and to the right of the lower matrix.



	B	C	D	E	F	G	H	I
16	Costs							
17	From\To	Spring	Summer	Fall	Winter	dummy		
18	RegSpr	10000	7.00	7.70	8.40	0		
19	RegSum	10000	10000	7.00	7.70	0		
20	RegFall	10000	10000	10000	7.00	0		
21	Init	0	0.70	1.40	2.10	10000		
22	OTSpr	8.00	10000	10000	10000	0		
23	OTSum	10000	9.00	10000	10000	0		
24	OTFall	10000	10000	8.00	10000	0		
25	OTWinter	10000	10000	10000	10.00	0		
26								
27	Decisions							
28	From\To	Spring	Summer	Fall	Winter	dummy	Sign	Supply
29	RegSpr	0	100	0	100	0	<	200
30	RegSum	0	0	300	0	0	<	300
31	RegFall	0	0	0	350	0	<	350
32	Init	200	0	0	0	0	<	200
33	OTSpr	50	0	0	0	50	<	100
34	OTSum	0	0	0	0	50	<	50
35	OTFall	0	0	100	0	0	<	100
36	OTWinter	0	0	0	50	100	<	150
37	Sign	>	>	>	>	>		
38	Demand	250	100	400	500	200		7790

Transportation Problem

Enter Range of Variables to adjust, with row and column totals. Objective will appear in bottom-right corner.

Transportation!$C$29:$I$38

Enter Range of Costs

Transportation!$C$18:$G$25

Figure 4: Transportation problem example. The user supplies signs and values for each row and column constraint (lower matrix). The costs (upper matrix) include shaded cells set to a large value as a penalty. The optimal value of 7790 is visible in the bottom right-hand corner of the constraint matrix.

and the costs of using overtime labor, which differ depending on the season. There is also a carrying charge of \$0.70 for any garment not used in the quarter following its production; so a garment produced in the spring using regular labor has an effective cost of \$8.40 (\$7.00 plus 2 * \$0.70) if sold in the winter. The "Cost" matrix includes some artificially high prices to ensure that nothing is consumed before it is produced, and a "dummy" column reflecting the excess of supply over demand.

In order to use the link, the user enters the sign and value associated with each constraint, as in the screen shot in Figure 4. As with the assignment problem, the $I \times J$ transportation problem is assumed to be a minimization problem, with IJ integer variables and exactly $I + J$ constraints. In the assignment problem it is assumed that every constraint is of the form $\sum_i x_{ij} = 1$ or $\sum_j x_{ij} = 1$; in the transportation problem, the user needs to supply both the direction and the right-hand sides of the constraints. These are visible at the right side and along the bottom of the matrix of decision variables. The link enters the optimal value for the example problem, 7790, into in the bottom-right corner of the upper matrix.

4.4. S examples

In this section we solve two of the same examples using the S interface. The **lpSolve** package can be installed automatically in R with the command `install.packages("lpSolve")` on a computer connected to the Internet. For manual installation in R, or in S-PLUS, the user needs to ensure that the DLL (properly compiled) is available on the system's path.

The integer program from section 4.1 can be run in the following way. Here the ">" is R's prompt and should not be entered by the user.

```
#
# Set up objective.
#
> ex.obj <- c( 592,  381,  273,  55,  48,  37,  23)
#
# The constraint needs to be a matrix.
#
> ex.con <- matrix (c(3534, 2356, 1767, 589, 528, 451, 304), nrow=1)
#
# Create the sign and the right-hand side of the constraint.
#
> ex.sign <- "<="
> ex.rhs <- 119567
#
# Require all seven variables to be integers. In general this
# vector will contain the indices of the integer variables.
#
> ex.int <- 1:7
#
# Solve the linear program. By default print only the objective
#
> lp ("max", ex.obj, ex.con, ex.sign, ex.rhs, int.vec=ex.int)
Success: the objective function is 19979
#
# Solve and print the vector of optimal values.
#
> lp ("max", ex.obj, ex.con, ex.sign, ex.rhs, int.vec=ex.int)$solution
[1] 32  2  1  0  0  0  0
#
# Require only variables 1 and 3 to be integer
#
> lp ("max", ex.obj, ex.con, ex.sign, ex.rhs, int.vec=c(1, 3))$solution
[1] 33.00  1.25  0.00  0.00  0.00  0.00  0.00
```

The transportation problem from section 4.3 is solved in the following way.

```
#
# Set up cost matrix
#
> costs <- matrix (10000, 8, 5); costs[4,1] <- costs[-4,5] <- 0
> costs[1,2] <- costs[2,3] <- costs[3,4] <- 7
> costs[1,3] <- costs[2,4] <- 7.7
> costs[5,1] <- costs[7,3] <- 8; costs[1,4] <- 8.4; costs[6,2] <- 9
> costs[8,4] <- 10; costs[4,2:4] <- c(.7, 1.4, 2.1)
#
# Set up constraint signs and right-hand sides.
#
> row.signs <- rep ("<", 8)
> row.rhs <- c(200, 300, 350, 200, 100, 50, 100, 150)
> col.signs <- rep (">", 5)
> col.rhs <- c(250, 100, 400, 500, 200)
#
# Now run. lp.transport() converts the "solution" into a matrix.
#
> lp.transport (costs, row.signs, row.rhs, col.signs, col.rhs)
Success: the objective function is 7790
> lp.transport (costs, row.signs, row.rhs, col.signs, col.rhs)$solution

     [,1] [,2] [,3] [,4] [,5]
[1,]    0  100    0  100    0
[2,]    0    0  300    0    0
[3,]    0    0    0  350    0
[4,]  200    0    0    0    0
[5,]   50    0    0    0   50
[6,]    0    0    0    0   50
[7,]    0    0  100    0    0
[8,]    0    0    0   50  100
```

4.5. Benchmark timing

In order to examine `lp_solve`'s speed we used some of the problems from the the netlib library (see http://www.netlib.org/lp/data/readme). We were not able to consider the NUOpt package from S-PLUS because we have not purchased this product. Comparisons were made between the `lp_solve` and the Excel **Solver** (for problems with integer variables), and among these two and the `solveLP` and `simplex` functions (from the **linprog** and **boot** packages, respectively) for problems with no integer variables.

Initially we chose the "agg" problem (163 variables, none integer, and 489 constraints) with with to compare our approach with Excel's native solver, but (even though the number of variables in that problem is smaller than the reported maximum of 200) Excel (version 2003) reported that the problem was "too large for Solver to handle." The `lp_solve` package produced Netlib's solution in under a second; `simplex` produced errors and no solution, whereas `solveLP` produced warnings and a solution that was 87% of the optimum. (It must be noted that we programmed routines to convert among the various argument formats, which worked in most cases. Although we can find no error in our routines, we certainly might be responsible for the other packages not completing in some cases.)

For our second test, we used the "adlittle" problem, which has 97 variables (none integer), and 56 constraints (though Netlib reports there are 57). This problem was solved by `lp_solve` and `simplex` in under a second. `solveLP` reported an error. The Excel **Solver** produced a solution in around ten seconds.

As a final test without integer variables, we used the SC205 problem, which has 203 variables and 204 constraints. Because the number of cells is greater than 200, the native Excel **Solver** is unable to solve this problem. The `lp_solve` program, running on the same comparatively powerful desktop computer described above, ran in well under a second both through Excel and through R. In contrast, `simplex` took about four seconds, and the `solveLP` function took about seven seconds.

As an example of a problem with integer variables, we chose the "flugpl" problem from the "miplib" library (see, e.g., `http://miplib.zip.de/miplib3`). This problem has 18 variables, all integer, and 34 constraints. It takes about a second to solve using `lp_solve` (either through Excel or from R) and around thirty seconds on Excel's **Solver**.

5. Conclusion and future development efforts

5.1. Conclusion

This report describes progress in solving linear programs in Excel. The existing **Solver** add-in supplied with Excel handles only small problems and in earlier versions occasionally, as in Example 1 above, produced the wrong answer. Through the C and VBA link described here we can call the `lp_solve` free software from Excel and from S-PLUS and R to find solutions for problems orders of magnitude larger. This allows users to solve moderate to large linear and mixed integer linear programs at no additional cost. In the case of R, this adds a freeware linear program solution capability to a high-quality freeware statistical software environment. A workbook available from the author's web-site demonstrates the link in general linear programs and some functionality specific to transportation and assignment problems.

Although `lp_solve` appears to find the correct solution in the above examples, it cannot be considered validated software.

5.2. Future development

Right now the link is fairly primitive. For example, no constraints other than those on the row and column sums can be added to transportation or assignment problems, and the location of the optimal value cannot be selected by the user. The error-checking is also primitive: if a user enters no text at all into the objective function box, for example, that error will be caught and a reasonable message produced. However, if she enters some text that is not a range at all, the error is not currently caught by the link. Instead, Excel returns a run-time error whose text is largely indecipherable. A number of improvements to the form interface can be made; for example, right now constraints must appear in rows, not in columns.

For the moment the S-PLUS or Excel user needs either to install the DLL into one of the system directories, or to explicitly edit the code to reflect the location of the DLL. S-PLUS has a more intelligent installation mechanism which we have not yet implemented. We are not sure that Excel supports installation in which the user can select the location of the DLL, but the DLL can be used as long as it can be found on the system path. The add-in itself is handled by Excel's built-in add-in manager. The link will continue to be distributed under the same license as the original `lp_solve` software.

More recent versions of the link now support `lp_solve` version 5 which allow calls to other lp_solve functions for, e.g., presolving and the printing of dual values. These features have been added and others will be. The next major phase of the development effort (for R and S-PLUS) will center on using sparse matrix representations so that bigger problems can be solved. Interested users can always find the latest version for R, together with documentation, at the Comprehensive R Archive Network (http://CRAN.R-project.org/); these will require R 2.0.0 or higher. Updated versions of the link for S-PLUS and Excel will be maintained at the author's web site, http://web.nps.navy.mil/.

References

Bronson R (1982). *Operations Research.* Schaum's Outline Series. McGraw-Hill, New York, NY.

Insightful Corporation (2001). *S-PLUS 6 for Windows User's Guide.* Insightful Corporation, Seattle, WA.

Nemhauser G, Wolsey L (1988). *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, NY.

R Development Core Team (2005). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL http://www.R-project.org/.

Walkenbach J (1999). *Microsoft Excel 2000 Power Programming with VBA.* IDG Books, Foster City, CA.

Affiliation:

Samuel E. Buttrey
Code OR/Sb
Department of Operations Research
Naval Postgraduate School
Monterey, CA 93943, United States of America
Telephone: +1/831/656-3035
E-mail: buttrey@nps.edu

Journal of Statistical Software
May 2005, Volume 14, Issue 4.
http://www.jstatsoft.org/

Submitted: 2004-10-08
Accepted: 2005-05-11