lpSolveAPI Package Users Guide

Kjell Konis

Contents

1 Introduction

The lpSolveAPI package provides an R API for the lp_solve library, a *mixed integer linear programming* (MILP) solver with support for pure linear, (mixed) integer/binary, semi-continuous and special ordered sets (SOS) models. The lp_solve library uses the revised simplex method to solve pure linear programs and uses the branch-and-bound algorithm to handle integer variables, semi-continuous variables and special ordered sets.

Note there is separate package called lpSolve available on CRAN that provides a few high-level functions for solving specific types of linear programs. The lpSolve package is based on an older version of lp_solve (5.5.0.8) and does not include the API.

The > shown before each R command is the R prompt. Only the text after > must be entered.

1.1 Installation

Use the following command to install the package from CRAN

```
> install.packages("lpSolveAPI")
```

then load it into your R session.

```
> library(lpSolveAPI)
```

Additionally, development versions of the package can be obtained from the R-Forge lpSolveAPI project website.

`http://lpsolve.r-forge.r-project.org`

1.2 Getting Help

Documentation is provided for each function in the lpSolve package using R's built-in help system. For example, the command

```
> help(add.constraint)
```

will display the documentation for the add.constraint function. Use the following command to list all of the functions provided by the lpSolveAPI package.

```
> ls("package:lpSolveAPI")
```

1.3 Conventions used in the lpSolveAPI Package

The syntax used to program lp_solve via the lpSolveAPI package is different from that normally used in R. The general approach used in the lpSolveAPI package is to create an *lpSolve linear program model object* (LPMO), use the set accessor methods to define a linear program (which will be contained in the LPMO), solve the linear program and finally use the get accessor methods to retrieve elements of the solution. The first argument to almost all of the functions in the lpSolveAPI package is therefore the lpSolve linear program model object that the function is meant to operate on.

A new lpSolve linear program model object with m constraints and n decision variables can be created using the `make.lp` function. For example, the following command creates an lpSolve linear program model object with 3 constraints and 2 decision variables.

```
> my.lp <- make.lp(3, 2)
```

The number of constraints m and the number of decision variables n can be retrieved using the generic `dim` function. Note that the objective function and right-hand-side are not counted in the dimension of the LPMO. Assignment via the `dim` function is not supported; to change the dimension of an LPMO it is necessary to use the `resize.lp` function.

The LPMO is now initialized but contains no data.

```
> my.lp
Model name:
            C1    C2
Minimize     0     0
R1           0     0  free  0
R2           0     0  free  0
R3           0     0  free  0
```

```
Type      Real  Real
Upper      Inf   Inf
Lower        0     0
```

The next step is to use the set accessor methods to define a linear programming problem. Suppose we wish to solve the following linear program

$$\begin{array}{rrcrcl} \text{minimize:} & -2x_1 & - & x_2 & & \\ \text{subject to:} & x_1 & + & 3x_2 & \leq & 4 \\ & x_1 & + & x_2 & \leq & 2 \\ & 2x_1 & + & & \leq & 3 \end{array}$$

with $x_1 \geq 0$ and $x_2 \geq 0$. The best way to build a model in lp_solve is columnwise; that is to set the columns then the objective function then the constraint types and finally the right-hand-side.

```
> set.column(my.lp, 1, c(1, 1, 2))
> set.column(my.lp, 2, c(3, 1, 0))
> set.objfn(my.lp, c(-2, -1))
> set.constr.type(my.lp, rep("<=", 3))
> set.rhs(my.lp, c(4, 2, 3))
> my.lp
```

```
Model name:
            C1    C2
Minimize    -2    -1
R1           1     3  <=  4
R2           1     1  <=  2
R3           2     0  <=  3
Type      Real  Real
Upper      Inf   Inf
Lower        0     0
```

The set accessor methods (the functions with names beginning with the word *set*) operate directly on an lpSolve linear program model object. Do not assign the output of a set accessor method and expect an lpSolve linear program model object. The (invisible) return value of these functions is generally a logical status code indicating whether the function executed sucessfully.

1.4 Caveats

The lpSolveAPI package provides an API for building and solving linear programs that mimics the lp_solve C API. This approach allows much greater flexibility but also has a few caveats. The most important is that the lpSolve linear program model objects created by `make.lp` and `read.lp` are not actually R objects but external pointers to lp_solve 'lprec' structures. R does not know how to deal with these structures. In particular, R cannot duplicate them. Thus you should avoid assigning LPMOs in R code.

Consider the following example. First we create an empty model `x`.

```
> x <- make.lp(2, 2)
```

Then we assign x to y.

```
> y <- x
```

Next we set some columns in x.

```
> set.column(x, 1, c(1, 2))
> set.column(x, 2, c(3, 4))
```

And finally, take a look at y.

```
> y
```

```
Model name:
            C1    C2
Minimize     0     0
R1           1     3  free  0
R2           2     4  free  0
Type      Real  Real
Upper      Inf   Inf
Lower        0     0
```

The changes made to `x` appear in `y` as well. Although `x` and `y` are two distinct objects in R, they both refer to the same external lp_solve 'lprec' structure.

1.5 A Detailed Example

Lets solve the mixed integer linear program

$$\begin{array}{rrcrcrcrcl} \text{minimize:} & 1x_1 & + & 3x_2 & + & 6.24x_3 & + & 0.1x_4 & & \\ \text{subject to:} & & & 78.26x_2 & & & + & 2.9x_4 & \geq & 92.3 \\ & 0.24x_1 & & & + & 11.31x_3 & & & \leq & 14.8 \\ & 12.68x_1 & & & + & 0.08x_3 & + & 0.9x_4 & \geq & 4 \end{array}$$

where x_1 is a real decision variable with a minimum value of 28.6, x_2 is a nonnegative integer-valued decision variable, x_3 is a binary decision variable and x_4 is a real decision variable restricted to the range $[18, 48.98]$.

First, we create an LPMO with 3 constraints and 4 decision variables.

```
> lprec <- make.lp(3, 4)
```

Next we set the values in the first column.

```
> set.column(lprec, 1, c(0, 0.24, 12.68))
```

The lp_solve library is capable of using a sparse vectors to represent the constraints matrix. In the remaining three columns, only the nonzero entries are set.

```
> set.column(lprec, 2, 78.26, indices = 1)
> set.column(lprec, 3, c(11.31, 0.08), indices = 2:3)
> set.column(lprec, 4, c(2.9, 0.9), indices = c(1, 3))
```

Next, we set the objective function, constraint types and right-hand-side.

```
> set.objfn(lprec, c(1, 3, 6.24, 0.1))
> set.constr.type(lprec, c(">=", "<=", ">="))
> set.rhs(lprec, c(92.3, 14.8, 4))
```

By default, all decision variables are created as real with range $[0, \infty)$. Thus we must change the type of the decision variables x_2 and x_3.

```
> set.type(lprec, 2, "integer")
> set.type(lprec, 3, "binary")
```

Setting the type to binary automatically changes the range to $[0, 1]$. We still need to set the range constraints on x_1 and x_4.

```
> set.bounds(lprec, lower = c(28.6, 18), columns = c(1, 4))
> set.bounds(lprec, upper = 48.98, columns = 4)
```

Finally, we name the decision variables and the constraints.

```
> RowNames <- c("THISROW", "THATROW", "LASTROW")
> ColNames <- c("COLONE", "COLTWO", "COLTHREE", "COLFOUR")
> dimnames(lprec) <- list(RowNames, ColNames)
```

Lets take a look at what we have done so far.

```
> lprec
```

```
Model name:
            COLONE    COLTWO  COLTHREE   COLFOUR
Minimize         1         3      6.24       0.1
THISROW          0     78.26         0       2.9  >=  92.3
THATROW       0.24         0     11.31         0  <=  14.8
LASTROW      12.68         0      0.08       0.9  >=     4
Type          Real       Int       Int      Real
Upper          Inf       Inf         1     48.98
Lower         28.6         0         0        18
```

Now we can solve the model and retrieve some results.

```
> solve(lprec)
```

```
[1] 0
```

A return value of 0 indicates that the model was successfully solved.

```
> get.objective(lprec)
```

```
[1] 31.78276
```

```
> get.variables(lprec)
```

```
[1] 28.60000  0.00000  0.00000 31.82759
```

```
> get.constraints(lprec)
```

```
[1]  92.3000   6.8640 391.2928
```

Note that there are some commands that return an answer. For the get accessor functions (the functions with names beginning with the word *get*) the output should be clear. For other functions (e.g., solve), the interpretation of the returned value is described in the documentation. Since solve is generic in R, use the command

```
> help(solve.lpExtPtr)
```

to view the appropriate documentation. The set accessor functions (the functions with names beginning with the word *set*) also have a return value - often a logical value indicating whether the command was successful - that is returned invisibly. Invisible values can be assigned but are not echoed to the console. For example,

```
> status <- add.constraint(lprec, 1:4, ">=", 5)
> status
```

```
[1] TRUE
```

indicates that the operation was successful. Invisible values can also be used in flow control.