lpSolveAPI Package Users Guide

Kjell Konis

Contents

1 Introduction

1.1 Installation

Installing the package takes a single command:

```
> install.packages("lpSolve", repos = "http://r-forge.r-project.org")
```

Note that there is also an lpSolve package available on CRAN. The CRAN version of the package is based on lp_solve version 5.5.0.8 and does not include the API. Note

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

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.

1.3 Building and Solving Linear Programs Using the lpSolveAPI R Package

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 one must never assign an existing lpSolve linear program model object in R code.

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

```
> library(lpSolveAPI)
> 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 we made in `x` appear in `y` as well. Although `x` and `y` are two distinct objects in R, they both refer to the same lp_solve 'lprec' structure.

1.4 Learning by Example

```
> lprec <- make.lp(0, 4)
> set.objfn(lprec, c(1, 3, 6.24, 0.1))
> add.constraint(lprec, c(0, 78.26, 0, 2.9), ">=", 92.3)
> add.constraint(lprec, c(0.24, 0, 11.31, 0), "<=", 14.8)
> add.constraint(lprec, c(12.68, 0, 0.08, 0.9), ">=", 4)
> set.bounds(lprec, lower = c(28.6, 18), columns = c(1, 4))
> set.bounds(lprec, upper = 48.98, columns = 4)
> 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      Real      Real      Real
Upper          Inf       Inf       Inf     48.98
Lower         28.6         0         0        18
```

Now lets solve the model.

```
> solve(lprec)
```

```
[1] 0
```

```
> 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 accessor functions (generally named 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 assignment functions (generally named

`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, c(12.68, 0, 0.08, 0.9), ">=",
+     4)
> status
```

```
[1] TRUE
```

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

2 Further Examples

2.1 Solving Dense Mixed Integer/Linear Programs

2.2 Sparse Linear Programs: The Transportation Problem

2.3 Integer Variables: Non-Integer c_{ij}

2.4 Binary Variables: The 8 Queens Problem