lpsymphony - Integer Linear Programming in R

Vladislav Kim

October 17, 2016

Contents

1 Introduction **2**

2 lpsymphony: Quick Start **2**

3 Integer Linear Programming **4**

3.1 Equivalent and Dual Formulations 4

3.2 Solving the Dual Problem 5

3.3 Methods in Integer Programming 6

1 Introduction

Package **lpsymphony** adapts Symphony[1], an open-source mixed-integer linear programming (MILP) solver, for use in R. This package is largely based on Rsymphony package distributed via CRAN [2]. The point of divergence, however, is the inclusion of sources for POSIX systems and DLLs for Windows, which facilitates the installation of Symphony itself.

2 lpsymphony: Quick Start

The package exports `lpsymphony_solve_LP()` function, which provides a basic interface between the C code of SYMPHONY and R. The user passes a vector with objective function coefficients, a constraint matrix and a vector of applicable inequality signs (i.e. $\leq$, $\geq$, $=$, etc) as parameters. The function returns an optimal mixed or pure integer solution.

A simple example of function maximization would be

$$\max\{3x_1 + x_2 + 3x_3\}$$

subject to

$$\begin{aligned} -x_1 + 2x_2 + x_3 &\leq 4 \\ 4x_2 - 3x_3 &\leq 2 \\ x_1 - 3x_2 + 2x_3 &\leq 3 \\ x_1,\, x_3 &\in \mathbb{Z}^+ \\ x_2 &\in \mathbb{R}^+ \end{aligned}$$

In this simple mixed-integer problem we have 2 variables that are integer-valued (x_1 and x_3) and a rational variable (x_3). Note that in general variables x_i of MILP problem are non-negative and hence a "$+$" subscript on $\mathbb{Z}^+$ and $\mathbb{R}^+$.

To solve this maximization problem using lpsymphony include the package:

```
library("lpsymphony")
```

Define the vector of objective function coefficients. In this case $(3,\ 1,\ 3)$:

```
obj <- c(3, 1, 3)
```

Define the constraint matrix (left-hand side coefficients):

```
mat <- matrix(c(-1, 0, 1, 2, 4, -3, 1, -3, 2), nrow = 3)
# mat <- matrix(c(-1,2,1,0,4,-3,1,-3,2), nrow = 3)
```

Be careful not to mix up rows and columns! Here we create the matrix in the column-major order, i.e. reading off the matrix entries column by column (so vertically, not horizontally). Note that the matrix in the comment

[1] https://projects.coin-or.org/SYMPHONY
[2] http://cran.r-project.org/web/packages/Rsymphony/index.html

is not the same as the problem's constraint matrix. In fact it is its transpose. As explained in 3.1 this is the constraint matrix of the *dual* problem.

Next indicate the inequality (or equality) signs:

```
dir <- c("<=", "<=", "<=")
```

Vector of right-hand side values:

```
rhs <- c(4, 2, 3)
```

This is a maximization problem, thus:

```
max <- TRUE
```

Important: indicate which variables are integer-valued and which are allowed to be rational. Possible options are "I" for integer, "B" for binary, "C" for continuous variable type.

```
types <- c("I", "C", "I")
```

To obtain the optimal solution call `lpsymphony_solve_LP()` function:

```
lpsymphony_solve_LP(obj, mat, dir, rhs, types = types, max = max)
```

```
## $solution
## [1] 5.00 2.75 3.00
##
## $objval
## [1] 26.75
##
## $status
## TM_OPTIMAL_SOLUTION_FOUND
##                         0
```

Setting Lower and Upper Bounds

Another useful argument in lpsymphony is the `bounds` argument. The default lower and upper bounds are zero and infinity, respectively.

To change the default bounds pass a list of upper-bound and lower-bound lists (argument `bounds` is a list of lists). For instance, if you wish that all variables in the previous problem are continuous and between 1 and 5, you can

```
bounds <- list(lower = list(ind = c(1,2,3), val = c(1,1,1)),
               upper = list(ind = c(1,2,3), val = c(5,5,5)))
```

Now we need to change types to all continuous variables and pass bounds to `lpsymphony_solve_LP`:

```
types <- c("C", "C", "C")
lpsymphony_solve_LP(obj, mat, dir, rhs, types = types, max = max, bounds = bounds)
```

```
## $solution
## [1] 5.000000 2.857143 3.285714
##
```

```
## $objval
## [1] 27.71429
##
## $status
## TM_OPTIMAL_SOLUTION_FOUND
##                         0
```

It is interesting to see if there are any integer solutions in the interval [1,5] and if they could be naively retrieved from rounding the previous solution:

```
types <- c("I", "I", "I")
lpsymphony_solve_LP(obj, mat, dir, rhs, types = types, max = max, bounds = bounds)
```

```
## $solution
## [1] 5 2 2
##
## $objval
## [1] 23
##
## $status
## TM_OPTIMAL_SOLUTION_FOUND
##                         0
```

As one can see, integer solutions for x_2 and x_3 are **not** simply rounded values of the continuous case. It is true in general that it is impossible to obtain an integer solution by simply rounding the solution up or down.

3 Integer Linear Programming

Given an objective function $z = c^T x$, where c is a vector of objective coefficients, mixed-integer maximization can be formulated as

$$\max \{c^T x \,|\, Ax \leq b,\ x \geq 0,\ x_i \in \mathbb{Z}\}$$

with the constraint matrix A and the right-hand-side vector b. Note that here x_i are all pure integers. However, in geneal some of x_i can be rational numbers.

In a linear programming (LP) relaxation the integer requirement is loosened and variables are allowed to take on continuous values to reach the first feasible solution or to conclude that the linear system doesn't have solutions. Solving an optimization problem by first treating x as floating point numbers and rounding off the final result might produce a good estimate of integer solution. However, finding an **exact** solution with LP relaxation does not produce accurate results in general.

3.1 Equivalent and Dual Formulations

Some care has to be taken in order to differentiate between an equivalent and dual formulations. An equivalent way of stating the above maximization as a minimization problem would be to use the negative of c, vector of

objective coefficients with constraint matrix A and RHS vector b same as above:

$$\min \{-c^T x \,|\, Ax \leq b,\ x \geq 0,\ x_i \in \mathbb{Z}\}$$

Every optimization problem can be viewed from 2 perspectives. The inversion of the *objective* of the problem, not simply max $\longleftrightarrow$ min exchange and sign inversion in the objective function, leads to duality. The original problem then is called a *primal* problem, while inversion of perspective gives rise to a *dual* problem.

A dual problem has a completely different interpretation than the primal and the following mathematical considerations have to be made. For example, the dual of the above maximization task would be

$$\min \{b^T y \,|\, A^T y \geq c,\ y \geq 0,\ y_i \in \mathbb{Z}\}$$

Note that the dual objective function is $b^T y$ with the primal right-hand side vector b acting as the (dual) vector of objective coefficients. The dual constraint matrix is the transpose of the original (primal) constraint matrix A and finally the dual right-hand side vector is the primal vector of objective coefficients c.

3.2 Solving the Dual Problem

Dual vector of objective coefficients is the primal RHS vector

```
obj_dual <- rhs # rhs fom the Quick Start example
```

The dual constraint matrix is the transpose of the primal A

```
mat_dual <- t(mat) # mat from the Quick Start example
```

Dual RHS vector is the primal objective vector

```
rhs_dual <- obj
```

Direction of dual signs is reversed with respect to primal

```
dir_dual <- c(">=", ">=", ">=")
```

Dual of maximization is minimization:

```
max_dual <- FALSE
```

Types can be used from the above program (in this case).

To obtain the optimal solution of the dual problem (minimization):

```
lpsymphony_solve_LP(obj_dual, mat_dual, dir_dual, rhs_dual, types = c("I","C","I"), max = max_dua
```

```
## $solution
## [1] 2 3 5
##
## $objval
## [1] 29
##
## $status
## TM_OPTIMAL_SOLUTION_FOUND
##                         0
```

3.3 Methods in Integer Programming

The earliest approach to integer programming was Gomory's method of cutting planes developed in 1958. The basic principle is to solve the LP relaxation of the problem at the beginning of each iteration step and check whether the solution is integer-valued. If the found solution is integral, the optimum is found. Otherwise, continue by generating new inequalities that usually lead to the tightening (Fig. 1) of the relaxation and then reoptimize. The process stops when all variables in the solution vector are integers. Fig. 1 illustrates schematically how the candidate solution space of the LP shrinks upon addition of Gomory cuts.

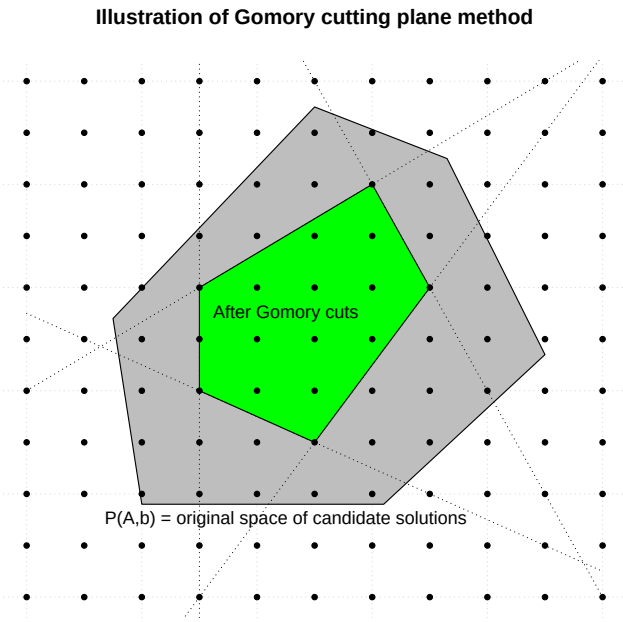

Figure 1: Gomory cuts reduce the number of candidate solutions. P(A,b) is a polyhedron defined by the initial set of constraints $Ax \leq b$. The black dots indicate integers; dotted lines are Gomory cuts

Modern integer linear programming solvers, however, do not solely utilize cutting plane algorithm to solve MILP problems. Symphony uses primarily **branch and bound** and **branch and cut** methods.

Branch and bound is a tree search algorithm that divides the initial problem into subproblems (Fig. 2) and eliminates invalid solutions by local and global bound checking. Different variants of branch and bound exist. Among implementation details are the following aspects: choice of the next tree node to be processed, the way bounds are calculated, the way the tree branches out, etc.

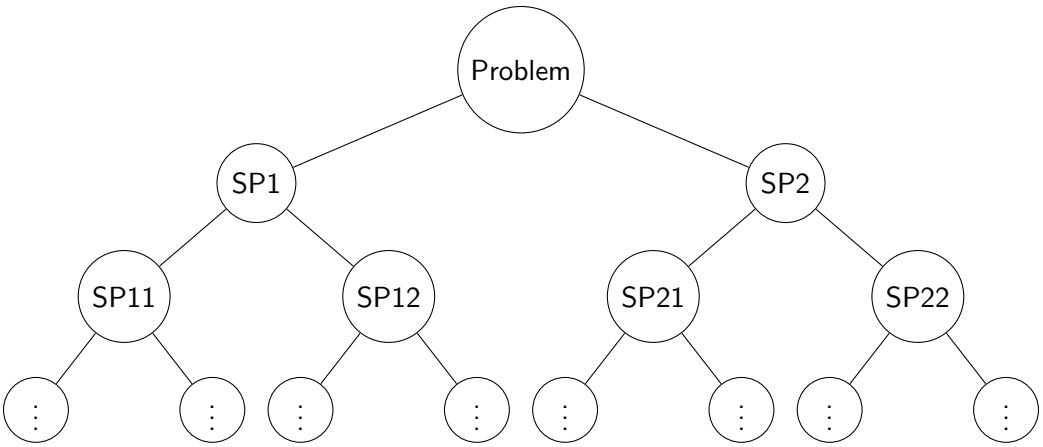

Figure 2: A sketch of a search tree generated by branch and bound. SP stands for subproblem

Branch and cut is a modification of branch and bound, in which cutting plane method is used to tighten LP relaxations. With branch and cut one can generate cuts at the root of the tree (i.e. only in the beginning) or at every iteration of branch and bound. Cut generation helps reduce the size of the tree and speeds up branch and bound.