ROI Plug-in NEOS

Hochreiter, Ronald
ron@hochreiter.net

Schwendinger, Florian
FlorianSchwendinger@gmx.at

August 29, 2020

Contents

1 NEOS

The NEOS Server can be used to solve several types of optimization problems. The optimization problems can be uploaded to the server and solved by a chosen solver. More information can be found at https://neos-server.org/neos/.

1.1 Solvers

	LP	MILP	QP	MIQP	QCQP	MIQCQP
AlphaECP			X	X	X	X
ANTIGONE			X	X	X	X
BARON			X	X	X	X
BDMLP	X					
Bonmin			X	X	X	X
Cbc	X	X				
CONOPT			X		X	
Couenne			X	X	X	X
CPLEX	X	X	X	X	X	X
DICOPT			X	X	X	X
FICO-Xpress	X	X				
Ipopt			X		X	
Knitro			X	X	X	X
LINDOGlobal			X	X	X	X
MINOS			X		X	
MOSEK	X	X	X	X	X	X
PATHNLP			X		X	
SBB			X	X	X	X
scip	X	X	X	X	X	X
SNOPT			X		X	

2 ROI.plugin.neos introduction

The R Optimization Infrastructure (**ROI**) plug-in **ROI.plugin.neos** allows to make use of the optimization solvers provided on the NEOS server. Thereby the optimization problem can be formulated directly in R, the optimization problem is sent to the NEOS server and after the problem is solved the solution is fetched from the NEOS server and transformed into the typical **ROI** solution format.

We set `ROI_LOAD_PLUGINS` to `FALSE` so no plugin is loaded automatically. This can speed up the loading of **ROI** if many plugins are installed.

```
Sys.setenv(ROI_LOAD_PLUGINS = FALSE)
library(ROI)

## ROI: R Optimization Infrastructure
## Registered solver plugins:  nlminb.
```

```
## Default solver:  auto.

library(ROI.plugin.neos)
```

2.1 First example

We define the problem like any other optimization problem in **ROI**

```
x <- OP(objective = c(3, 1, 3), maximum = TRUE)
constraints(x) <- L_constraint(L = rbind(c(-1,  2,  1), c( 0,  4, -3),
                                         c( 1, -3,  2)),
                               dir = leq(3), rhs = c(4, 2, 3))
types(x) <- c("I", "C", "I")
```

but instead of solving the problem locally with `"glpk"` or `"lpsolve"` we send the problem to the NEOS server to be solved by the SCIP solver.

```
(s <- ROI_solve(x, solver = "neos", method = "scip"))

## Optimal solution found.
## The objective value is: 2.675000e+01

solution(s)

## [1] 5.00 2.75 3.00
```

Note that `method` is matched after performing the following cleaning function

```
clean <- function(x) tolower(gsub("\\W", "", x))
```

which means

```
clean("SCIP")

## [1] "scip"

clean("scip")

## [1] "scip"

clean("-scip.")

## [1] "scip"

clean("ScIp.")

## [1] "scip"
```

would all select the `"SCIP"` solver.

2.2 Provide a user name

Some solvers (e.g. `"cplex"`) need a working email address, furthermore NEOS gives a higher priority to registered users.

```
ROI_solve(x, solver = "neos", method = "cplex")

## Error in SOLVE(x, cntrl):  CPLEX will not run unless you provide a valid email
address.
```

The username and email address can be provided as control argument.

```
s <- ROI_solve(x, solver = "neos", method = "cplex",
               email = "your_mailaddress@somewhere.net")
```

2.3 Obtain the original solver message

The entire solver message can be obtained as follows.

```
str(solution(s, "msg"))

## List of 5
##  $ solution     : num [1:3] 5 2.75 3
##  $ objval       : num 26.8
##  $ solver_status: num 1
##  $ model_status : num 8
##  $ message      : chr "Executed on prod-exec-4.neos-server.org\n\fGAMS 24.9.2  r64480 Released N
```

We remove `"\f"` since otherwise it would give an error in **knitr**.

```
cat(gsub("\f", "", solution(s, "msg")$message))

## Executed on prod-exec-4.neos-server.org
## GAMS 24.9.2  r64480 Released Nov 14, 2017 LEX-LEG x86 64bit/Linux 06/06/18 08:54:00 Page 1
## G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
## C o m p i l a t i o n
##
##
##
##
## COMPILATION TIME     =        0.001 SECONDS      3 MB  24.9.2 r64480 LEX-LEG
## GAMS 24.9.2  r64480 Released Nov 14, 2017 LEX-LEG x86 64bit/Linux 06/06/18 08:54:00 Page 2
## G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
## Model Statistics    SOLVE LinearProblem Using MIP From line 48
##
##
## MODEL STATISTICS
##
## BLOCKS OF EQUATIONS           3     SINGLE EQUATIONS            6
## BLOCKS OF VARIABLES           3     SINGLE VARIABLES            6
## NON ZERO ELEMENTS            16     DISCRETE VARIABLES          2
##
##
## GENERATION TIME      =        0.002 SECONDS      4 MB  24.9.2 r64480 LEX-LEG
```

```
##
##
## EXECUTION TIME       =        0.002 SECONDS      4 MB  24.9.2 r64480 LEX-LEG
## GAMS 24.9.2  r64480 Released Nov 14, 2017 LEX-LEG x86 64bit/Linux 06/06/18 08:54:00 Page 3
## G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
## Solution Report     SOLVE LinearProblem Using MIP From line 48
##
##
##               S O L V E      S U M M A R Y
##
##      MODEL   LinearProblem       OBJECTIVE  obj
##      TYPE    MIP                 DIRECTION  MAXIMIZE
##      SOLVER  SCIP                FROM LINE  48
##
## **** SOLVER STATUS     1 Normal Completion
## **** MODEL STATUS      8 Integer Solution
## **** OBJECTIVE VALUE               26.7500
##
##  RESOURCE USAGE, LIMIT          0.002      1000.000
##  ITERATION COUNT, LIMIT         6    2000000000
##
## SCIP             24.9.2 r64480 Released Nov 14, 2017 LEG x86 64bit/Linux
##
##                         LOWER     LEVEL     UPPER
##
## ---- EQU ObjSum          .         .         .
##
## ---- EQU LinLeq
##
##       LOWER     LEVEL     UPPER
##
## R1     -INF      3.500     4.000
## R2     -INF      2.000     2.000
## R3     -INF      2.750     3.000
##
## ---- EQU IntEq
##
##       LOWER     LEVEL     UPPER
##
## C1      .         .         .
## C3      .         .         .
##
##                         LOWER     LEVEL     UPPER
##
## ---- VAR obj            -INF     26.750     +INF
##
## ---- VAR x
##
##       LOWER     LEVEL     UPPER
##
## C1      .        5.000     +INF
## C2      .        2.750     +INF
## C3      .        3.000     +INF
##
## ---- VAR int
##
##       LOWER     LEVEL     UPPER
##
## C1      .        5.000     +INF
## C3      .        3.000     +INF
##
```

```
##
## **** REPORT SUMMARY :        0     NONOPT
##                              0 INFEASIBLE
##                              0  UNBOUNDED
## GAMS 24.9.2  r64480 Released Nov 14, 2017 LEX-LEG x86 64bit/Linux 06/06/18 08:54:00 Page 4
## G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
## E x e c u t i o n
##
##
## ----     52 ---BEGIN.SOLUTION---
##
## ----     52 VARIABLE x.L
##
## C1 5.00000000,    C2 2.75000000,    C3 3.00000000
##
##
## ----     52 ---END.SOLUTION---
##
## **** REPORT FILE SUMMARY
##
## results /var/lib/condor/execute/dir_1203368/results.txt
##
##
## EXECUTION TIME       =        0.000 SECONDS      3 MB  24.9.2 r64480 LEX-LEG
##
##
## USER: Small MUD - 5 User License                     G170411/0001AS-LNX
##       University of Wisconsin-Madison, Computer Sciences Dept.   DC8499
##       License for teaching and research at degree granting institutions
##
##
## **** FILE SUMMARY
##
## Input      /var/lib/condor/execute/dir_1203368/MODEL.gms
## Output     /var/lib/condor/execute/dir_1203368/solve.out
```

2.4 Obtain the model

The optimization model **ROI.plugin.neos** sends to NEOS can be inspected by setting the argument `dry_run` to `TRUE`.

```
model_call <- ROI_solve(x, solver = "neos", method = "mosek",
                        dry_run = TRUE)
cat(as.list(model_call)$xmlstring)
```

```
## <?xml version="1.0" encoding="UTF-8"?>
## <document>
##   <category>milp</category>
##   <solver>MOSEK</solver>
##   <inputMethod>GAMS</inputMethod>
##   <model><![CDATA[Option IntVarUp = 0;
##
## Set i / R1*R0 / ;
## Set j / C1*C3 / ;
## Set jint(j) / C1, C3 / ;
```

```
##
## Parameter objL(j)
## /C1 3
## C2 1
## C3 3/ ;
##
##
## Variables obj;
## Positive Variables x(j);
## Integer Variables int(jint);
##
##
## Equations
##     ObjSum
##     IntEq(jint);
##
## ObjSum .. obj =e= sum(j, x(j) * objL(j)) ;
## IntEq(jint) .. x(jint) =e= int(jint);
##
## Model LinearProblem /all/ ;
##
## Solve LinearProblem using MIP maximizing obj ;
##
## option decimals = 8;
##
## display '---BEGIN.SOLUTION---', x.l, '---END.SOLUTION---';
##
##
## file results /results.txt/;
## results.nw = 0;
## results.nd = 15;
## results.nr = 2;
## results.nz = 0;
## put results;
## put 'solution:'/;
## loop(j, put, x.l(j)/);
## put 'objval:'/;
## put LinearProblem.objval/;
## put 'solver_status:'/;
## put LinearProblem.solvestat/;
## put 'model_status:'/;
## put LinearProblem.modelstat/;]]></model>
##   <options><![CDATA[]]></options>
##   <gdx><![CDATA[]]></gdx>
##   <wantgdx><![CDATA[]]></wantgdx>
##   <wantlog><![CDATA[]]></wantlog>
##   <comments><![CDATA[]]></comments>
## </document>
```

2.5 Asynchronous execution

In some situations it can be advantageous to do the calculations asynchronous. For example you have a rather big optimization problem and otherwise your connection would run into a time out. Asynchronous can be easily performed by setting the parameter `wait` to `FALSE`.

```
neos_job <- ROI_solve(x, solver = "neos", method = "scip", wait = FALSE)
str(neos_job)

## List of 8
##  $ job_number        : int 6571509
##  $ password          : chr "vbGPJahp"
##  $ status            :function ()
##  $ info              :function ()
##  $ final_results     :function ()
##  $ output_file       :function (file_name)
##  $ objective_function:function (x)
##  - attr(*, "class")= chr "'function' 'L_objective' 'Q_objective' 'objective'"
##  $ solution          :function ()
##  - attr(*, "class")= chr "neos_job"

## Make R wait till the job finishes!
while (neos_job$status() != "Done") Sys.sleep(2)
## Obtain the solution from the server and transform it into the typical
## ROI solution object.
(s <- neos_job$solution())

## Optimal solution found.
## The objective value is: 2.675000e+01

solution(s)

## [1] 5.00 2.75 3.00
```

3 Use cases

3.1 A seemingly simple optimization problem

3.1.1 Introduction

We present a seemingly simple optimization problem which is in fact pretty hard to solve and shows that an utilization of the NEOS server through the package **ROI.plugin.neos** makes sense and is easy and fun to use.

We need the following R packages for our demonstration:

```
library(dplyr)
library(ROI)
library(ROI.plugin.glpk)
library(ompr)
```

```
library(ompr.roi)
library(CVXR)
```

3.1.2 Basic problem

Let's consider the following problem from Bertsimas and Freund (2000) where the Magnetron Company manufactures two types of microwave ovens: full-size and compact. Each full-size oven requires 2 hours of general assembly and 2 hours of electronic assembly, whereas each compact oven requires 1 hours of general assembly and 3 hours of electronic assembly. For the current production period, there are 500 hours of general assembly labor and 800 hours of electronic assembly labor available. The company estimates that it can sell up to 220 full-size and 180 compact ovens with an earnings contribution of EUR 120 per full-size oven and EUR 130 per compact oven. Magnetron wants to find a production plan that maximizes earnings!

Of course, this is a standard linear program that can be solved manually or using a modeling language, but let's start with the definition of an abstract meta decision model, which solves our problem:

```
variable F, C
maximize 120F + 130C
subject to
    2F + C <= 500;
    2F + 3C <= 800;
    F <= 220, C <= 180
    F >= 0, C >= 0
```

Solving it manually with **ROI** can be done the following way:

```
lp <- OP(L_objective(c(120, 130), c("full", "compact")),
         L_constraint(L = rbind(c(2, 1), c(2, 3)),
                      dir = c("<=", "<="), rhs = c(500, 800)),
         maximum = TRUE,
         bounds = V_bound(ub = c(220, 180)))
(sol <- ROI_solve(lp, solver = "glpk"))

## Optimal solution found.
## The objective value is: 4.050000e+04

solution(sol)

##    full compact
##     175     150
```

Solving it with a modeling approach like 'ompr' reads as follows:

```
result <- MIPModel() %>%
  add_variable(full, type = "continuous", lb = 0, ub=220) %>%
  add_variable(compact, type = "continuous", lb = 0, ub=180) %>%
  set_objective(120*full + 130*compact, "max") %>%
  add_constraint(2*full + 1*compact <= 500) %>%
```

```
  add_constraint(2*full + 3*compact <= 800) %>%
  solve_model(with_ROI(solver = "glpk"))
get_solution(result, full)

## full
##  175

get_solution(result, compact)

## compact
##     150
```

3.1.3 Extension

It was assumed that the prices of full-size and compact microwave ovens are set so that the resulting unit contributions to earnings are EUR 120 and EUR 130 per oven for full-size and compact microwave ovens, respectively. As it turns out, the unit earnings contribution of EUR 120 per oven for full-size ovens derives from the fact that Magnetron has set the price of a full-size oven to be EUR 270, and the variable production cost of a full-size oven is EUR 150 (and so the unit contribution to earnings is $120 = 270 - 150$. Also, the unit earnings contribution of EUR 130 per oven for compact ovens derives from the fact that Magnetron has set the price of a compact oven to be EUR 230, and the variable production cost of a compact oven is EUR 100 (and so the unit contribution to earnings is EUR $130 = 230 - 100$. As a next step in the marketing/production planning process, the company would like to determine the optimal combination of prices and production levels to maximize the overall contribution to earnings.

The changes in the prices of ovens will result in changes in demand. Suppose that Magnetron has estimated that the demand for their ovens is related to the prices they set as follows, i.e. $D_F = 490 - P_F$ and $D_C = 640 - 2P_C$ where D_F and D_C are the demands for full-size and compact ovens, and P_F and P_C are the respective prices set by Magnetron for full-size and compact ovens. It is obvious that that when $P_F = 270$ and $P_C = 230$, the demands are as specified in the linear model.

The resulting meta-model looks like this:

```
variable F, C, P_F, P_C
maximize F(P_F - 150) + C(P_C - 100)
subject to:
    2F + C <= 500
    2F + 3C <= 800
    F <= 490 - P_F
    C <= 640 - 2P_C
    F, C, PF, PC >= 0
```

If we want to solve this nonlinear optimization model to determine the optimal pricing and production strategy, we actually run into a problem. This seemingly easy extension is actually problematic. First we try to solve this problem by making use of the **CVXR** package.

```
full <- Variable(1)
compact <- Variable(1)
```

```
p_full <- Variable(1)
p_compact <- Variable(1)

objective <- Maximize(full * (p_full - 150) + compact * (p_compact - 100))

## Warning in full * (p_full - 150): Forming a non-convex expression (affine) *
(affine)
## Warning in compact * (p_compact - 100): Forming a non-convex expression
(affine) * (affine)

constr <- list(2 * full + compact <= 500,
    2 * full + 3 * compact <= 800, full <= 490 - p_full,
    compact <= 640 - 2 * p_compact, full >= 0, compact >= 0,
    p_full >= 0, p_compact >= 0)
magnetron <- Problem(objective, constr)
cvxr_sol <- solve(magnetron)

## Error in CVXR::psolve(a, b, ...): Problem does not follow DCP rules.
```

Here we see that since the problem is non-convex it cannot be solved by **CVXR**.

Especially in the case of quadratic non-convex optimization problems **ROI.plugin.neos** comes in handy, as we just need to formulate the model in **ROI** and can directly send it off to the NEOS server as shown below.

```
library(slam)
Q <- simple_triplet_matrix(i = 1:4, j = c(2, 1, 4, 3), rep(1, 4))
as.matrix(Q)

##      [,1] [,2] [,3] [,4]
## [1,]    0    1    0    0
## [2,]    1    0    0    0
## [3,]    0    0    0    1
## [4,]    0    0    1    0

var_names <- c("full", "price_full", "compact", "price_compact")
o <- OP(
    Q_objective(Q = Q, L = c(-150, 0, -100, 0),  names = var_names),
    L_constraint(rbind(c(2, 0, 1, 0), c(2, 0, 3, 0),
                       c(1, 1, 0, 0), c(0, 0, 1, 2)),
                 dir = leq(4), rhs = c(500, 800, 490, 640)),
    maximum = TRUE)
```

On the NEOS server there exist several options to solve non-convex quadratic problems. In this example we make use of the BARON solver.

```
(sol <- ROI_solve(o, solver = "neos", method = "BARON"))

## Optimal solution found.
## The objective value is: 5.128182e+04
```

```
solution(sol)

##          full    price_full       compact price_compact
##      151.8182      338.1818      165.4545      237.2727
```

References

Dimitris Bertsimas and Robert M. Freund. *Data, Models, and Decisions: The Fundamentals of Management Science*. South-Western College Publishing, 1st edition, 2000.