cplexAPI – Quick Start

Gabriel Gelius-Dietrich

March 15, 2012

1 Introduction

The package *cplexAPI* provides a low level interface to the C API of IBM® ILOG® CPLEX®[1]. The package *cplexAPI* requires a working installation of IBM® ILOG® CPLEX®.

2 Installation

The package *cplexAPI* depends on a working installation of IBM® ILOG® CPLEX®. See `INSTALL` for installation instructions and platform specific details.

3 Usage

At first, load the library.

```
> library(cplexAPI)
```

3.1 Creating and solving a linear optimization problem

In the following, an example linear programming problem will be created and solved:

maximize

$$z = 5x_1 + 4x_2 + 3x_3$$

subject to

$$\begin{aligned} 2x_1 + 3x_2 + x_3 &\leq 5 \\ 4x_1 + x_2 + 2x_3 &\leq 11 \\ 3x_1 + 4x_2 + 2x_3 &\leq 8 \end{aligned}$$

With all variables being non-negative.

[1] IBM® ILOG® CPLEX® version $\geq$ 12.1 from the IBM Academic Initiative
`https://www.ibm.com/developerworks/university/academicinitiative/`

Open a IBM® ILOG® CPLEX® environment.

```
> env <- openEnvCPLEX()
```

Create a problem object.

```
> prob <- initProbCPLEX(env)
```

Assign a name to the problem object.

```
> chgProbNameCPLEX(env, prob, "sample")
```

```
[1] 0
```

Prepare data structures for the problem object. Number of columns and rows.

```
> nc <- 3
> nr <- 3
```

Objective function.

```
> obj <- c(5, 4, 3)
```

Right hand side.

```
> rhs <- c(5, 11, 8)
```

Sense of the right hand side.

```
> sense <- rep("L", 3)
```

Variable lower bounds.

```
> lb <- rep(0, 3)
```

Variable upper bounds.

```
> ub <- rep(CPX_INFBOUND, 3)
```

Column and row names.

```
> cn <- c("x1", "x2", "x3")
> rn <- c("q1", "q2", "q3")
```

The constraint matrix is passed in column major order format. **Be careful here:** all indices start with 0! Begin indices of rows.

```
> beg <- c(0, 3, 6)
```

Number of non-zero elements per row.

```
> cnt <- rep(3, 3)
```

Column indices.

```
> ind <- c(0, 1, 2, 0, 1, 2, 0, 1, 2)
```

Non-zero elements.

```
> val <- c(2, 4, 3, 3, 1, 4, 1, 2, 2)
```

Load problem data.

```
> copyLpwNamesCPLEX(env, prob, nc, nr, CPX_MAX, obj, rhs, sense,
+                   beg, cnt, ind, val, lb, ub, NULL, cn, rn)
```

```
[1] 0
```

Solve the problem using the simplex algorithm.

```
> lpoptCPLEX(env, prob)
```

```
[1] 0
```

Retrieve solution after optimization.

```
> solutionCPLEX(env, prob)
```

```
$lpstat
[1] 1

$objval
[1] 13

$x
[1] 2 0 1

$pi
[1] 1 0 1

$slack
[1] 0 1 0

$dj
[1]  0 -3  0
```

Write the problem to file `prob.lp` in lp format.

```
> writeProbCPLEX(env, prob, "prob.lp")
```

```
[1] 0
```

Read problem from file `prob.lp` in lp format.

```
> lp <- initProbCPLEX(env)
> readCopyProbCPLEX(env, lp, "prob.lp")
```

```
[1] 0
```

Free memory, allacated to the problem object.

```
> delProbCPLEX(env, prob)
```

```
[1] 0
```

Close IBM® ILOG® CPLEX® environment.

```
> closeEnvCPLEX(env)
```

```
[1] 0
```

3.2 Creating and solving a mixed integer programming (MIP) problem

In the following, an example MIP will be created and solved: [2]

maximize

$$z = x_1 + 2x_2 + 3x_3 + x_4$$

subject to

$$\begin{aligned} -x_1 + x_2 + x_3 + 10x_4 &\leq 20 \\ x_1 - 3x_2 + x_3 &\leq 30 \\ x_2 - 3.5x_4 &= 0 \end{aligned}$$

With all variables being non-negative, $x_1 \leq 40$ and $x_4 \in \{2, 3, 4\}$ (x_4 is integer).

Open a IBM® ILOG® CPLEX® environment.

```
> env <- openEnvCPLEX()
```

Create a problem object.

```
> prob <- initProbCPLEX(env, pname = "example")
```

Prepare data structures for the problem object. Number of columns, rows and non-zero elements.

```
> nc <- 4
> nr <- 3
> nz <- 9
```

[2] Taken from IBM® ILOG® CPLEX® example file `mipex1.c`.

Objective function.

```
> obj <- c(1.0, 2.0, 3.0, 1.0)
```

Right hand side.

```
> rhs <- c(20.0, 30.0, 0.0)
```

Sense of the right hand side.

```
> sense <- c("L", "L", "E")
```

Vatiable types.

```
> ctype <- c("C", "C", "C", "I")
```

Variable lower bounds.

```
> lb <- c(0.0, 0.0, 0.0, 2.0)
```

Variable upper bounds.

```
> ub <- c(40.0, CPX_INFBOUND, CPX_INFBOUND, 3.0)
```

The constraint matrix is passed in column major order format. **Be careful here:** all indices start with 0! Begin indices of rows.

```
> beg <- c(0, 2, 5, 7)
```

Number of non-zero elements per row.

```
> cnt <- c(2, 3, 2, 2)
```

Column indices.

```
> ind <- c(0, 1, 0, 1, 2, 0, 1, 0, 2)
```

Non-zero elements.

```
> val <- c(-1.0, 1.0, 1.0, -3.0, 1.0, 1.0, 1.0, 10.0, -3.5)
```

Load problem data.

```
> copyLpCPLEX(env, prob, nc, nr, CPX_MAX, obj, rhs, sense,
+             beg, cnt, ind, val, lb, ub)
```

```
[1] 0
```

Set Variable types.

```
> copyColTypeCPLEX(env, prob, ctype)
```

```
[1] 0
```

Solve the problem using MIP.

```
> mipoptCPLEX(env, prob)

[1] 0
```

Retrieve solution after optimization.

```
> solutionCPLEX(env, prob)

$lpstat
[1] 101

$objval
[1] 122.5

$x
[1] 40.0 10.5 19.5  3.0

$pi
[1] NA

$slack
[1] 0 2 0

$dj
[1] NA
```

Free memory, allacated to the problem object.

```
> delProbCPLEX(env, prob)

[1] 0
```

Close IBM® ILOG® CPLEX® environment.

```
> closeEnvCPLEX(env)

[1] 0
```

3.3 Setting control prarmeters

Open a new environment.

```
> pe <- openEnvCPLEX()
```

All parameters and possible values are described in the IBM® ILOG® CPLEX® documentation. All parameters can be set in *cplexAPI*; the parameters names are the same as in IBM® ILOG® CPLEX®. For example, if one wants to use the debugging routines, the 'messages to screen switch' must be set to 1.

```
> setIntParmCPLEX(pe, CPX_PARAM_SCRIND, CPX_ON)

[1] 0
```

Do not use advanced start information.

```
> setIntParmCPLEX(pe, CPX_PARAM_ADVIND, 0)

[1] 0
```

Lower the feasibility tolerance.

```
> setDblParmCPLEX(pe, CPX_PARAM_EPRHS, 1E-09)

[1] 0
```

Retrieve parameters which are not set at their default values.

```
> (param <- getChgParmCPLEX(pe))

[1] 1001 1016 1035
```

Retrieve names of these parameters.

```
> mapply(getParmNameCPLEX, param, MoreArgs = list(env = pe))

[1] "CPX_PARAM_ADVIND" "CPX_PARAM_EPRHS"  "CPX_PARAM_SCRIND"
```

Close the envoronment.

```
> closeEnvCPLEX(pe)

[1] 0
```

4 Function names

4.1 Searching

The function names in *cplexAPI* are different from the names in IBM® ILOG® CPLEX®, e. g. the function `addColsCPLEX` in *cplexAPI* is called `CPXaddcols` in IBM® ILOG® CPLEX®. The directory `inst/` containes a file `c2r.map` which maps a IBM® ILOG® CPLEX® function name to the corresponding *cplexAPI* function name. Additionally, all man-pages contain an alias to the IBM® ILOG® CPLEX® function name. The call

```
> help("CPXaddcols")
```

will bring up the man-page of `addColsCPLEX`.

4.2 Mapping

The file `c2r.map` in `inst/` maps the *cplexAPI* function names to the orininal IBM® ILOG® CPLEX® function names of its C-API. To use the latter, run

```
> c2r <- system.file(package = "cplexAPI", "c2r.map")
> source(c2r)
```

now either

```
> pr1 <- openEnvCPLEX()
> closeEnvCPLEX(pr1)

[1] 0
```

or the original functions

```
> pr2 <- CPXopenCPLEX()
> CPXcloseCPLEX(pr2)

[1] 0
```

work both. Keep in mind that the mapping only affects the function names not the arguments of a function.