Utilisation de la *callable library* Cplex

Hélène Toussaint, m.a.j. février 2015

Ce document présente brièvement la librairie de **Cplex** ([1]), appelée *Cplex Callable Library,* qui permet d'accéder aux différentes fonctionnalités de **Cplex** depuis un programme écrit en langage C. On ne donne ici qu'une description générale de cette librairie en en présentant les principales fonctions (déclarer, construire et résoudre un programme linéaire / accéder à la solution).

Le manuel de référence est disponible en ligne ([2]) ainsi qu'un tutoriel ([3]). Des exemples de programmes sont également disponibles dans le répertoire d'installation de **Cplex**.

La suite du document montre, étape par étape, l'utilisation des principales fonctions de la *Cplex Callable Library :*

Etape 1 : Déclarer et initialiser l'environnement Cplex

La première chose à faire est d'inclure les fichiers d'entête qui contiennent les déclarations des différentes variables et fonctions **Cplex** :

```
#include <ilcplex/cplex.h>
```

Il faut ensuite déclarer et initialiser une variable pointant sur l'environnement **Cplex**. Cette variable sera passée en paramètre de toutes les fonctions de la librairie :

```
  int status; /* contient le type d'erreur rencontré en cas de problème */
  CPXENVptr env = CPXopenCPLEX (&status); /* ouvre un environnement Cplex */

   if ( !env ) {/*en cas d'erreur...*/
      char  errmsg[1024];
      CPXgeterrorstring (env, status, errmsg);
      fprintf (stderr, "%s", errmsg);
   }
```

On remarquera que les fonctions de la *Callable Library* sont toutes préfixées par CPX (ou CPXX pour les applications 64 bits).

Etape 2 : Déclarer et initialiser le problème linéaire

Une autre variable qui sera omniprésente c'est la variable qui pointe sur le programme linéaire que l'on souhaite résoudre :

```
  CPXLPptr lp = CPXcreateprob (env, &status, "nomDuProb"); /*crée un PL vide*/

  if ( !lp )
    {/* traiter l'erreur...*/}
```

Etape 3 : Construction du problème linéaire

On va construire le PL suivant :

Minimiser $x_1 + x_2 + x_3$
Sous
$x_1 - 6x_3 \geq 3$
$3x_2 + x_3 \geq 4$
$x_1, x_2, x_3 \in \mathbb{N}$

La matrice des contraintes associée est :

$$M = \begin{pmatrix} 1 & 0 & -6 \\ 0 & 3 & 1 \end{pmatrix}$$

Pour construire ce PL il faut déclarer chacune des variables puis construire la matrice des contraintes.

Etape 3.1 : Création des variables (ou colonnes)

On note **`nbVar`** le nombre de variables de notre problème (qui correspond aussi au nombre de colonnes de la matrice des contraintes), dans notre exemple **`nbVar`** = 3.

La création des colonnes se fait à l'aide de la fonction **`CPXnewcols :`**

```
int CPXnewcols (CPXENVptr env, CPXLPptr lp, int nbVar, double *coeff, double *lb,
double *ub, char *ctype, char **colname)
```

qui prend en paramètre :

- `env, lp :` les deux pointeurs définis précédemment
- `nbVar :` le nombre de variables
- `coeff :` un tableau de dimension `nbVar` qui contient les coefficients des variables dans la fonction objectif
- `lb :` un tableau de dimension `nbVar` qui contient la borne inférieure des variables
- `ub :` un tableau de dimension `nbVar` qui contient la borne supérieure des variables
- `ctype :` le type des variables ('B' pour binaire, 'I' pour entier, et 'C' pour réel)
- `colname :` un tableau de `char *` de dimension `nbVar` qui contient le nom des variables

Les vecteurs `lb, ub` et `colname` sont facultatifs : si on passe `NULL` à la place de ces vecteurs, **Cplex** attribue les valeurs par défaut.

Exemple :

```
double * coeff = (double *)malloc(nbVar*sizeof(double)); /*coeff des variables dans la
fonction objectif*/
char * ctype = (char*)malloc(nbVar*sizeof(char)); /* type des variables */
double * lb = (double*)malloc(nbVar*sizeof(double)); /* borne inférieure */

for (i = 0; i < nbVar; ++i)
{
  coeff[i] = 1; /* chaque variable a un coefficient 1 dans la fonction objectif */
  ctype[i] = 'I'; /* chaque variable est entière */
  lb[i] = 0; /* chaque variable a une borne inf. = 0 */
}

status = CPXnewcols (env, lp, nbVar, coeff, lb, NULL, ctype, NULL);
if (status)
  printf("echec lors de la creation des variables\n");
```

Etape 3.2 : Création des contraintes (remplissage de la matrice)

La création des contraintes est réalisée par la fonction **`CPXaddrows :`**

```
int CPXaddrows (CPXENVptr env, CPXLPptr lp, int ccnt, int nbRow, int nbVar_non_nul,
double *rhs, char *sense, int *rmatbeg, int *rmatind, double *rmatval, char **colname,
char **rowname)
```

qui prend en paramètre :

- `env, lp :` les deux pointeurs définis précédemment
- `ccnt:` le nombre de nouvelles variables (toujours 0 dans notre cas car on a déjà toutes nos colonnes grâce à `CPXnewcols`)
- `nbRow` : nombre de lignes (= nombre de contraintes)
- `nbVar_non_nul` : nombre de variables avec un coefficient non nul dans la matrice des contraintes
- `rhs` : vecteur de dimension `nbRow`, `rhs`[i] contient le membre de droite de la contrainte i
- `sense` : vecteur de dimension `nbRow`, `sense`[i] contient le sens de la contrainte i : 'L' pour <=, 'E' pour = et 'G' pour >=
- `rmatbeg` : vecteur de dimension `nbRow`, `rmatbeg` contient les indices dans `matval` qui correspondent à des débuts de ligne
- `rmatind` : vecteur de dimension `nbVar_non_nul`, `rmatind` [i] contient le numéro de colonne de la variable dont le coefficient dans la matrice des contraintes est donné par `rmatval` [i]
- `rmatval` : vecteur de dimension `nbVar_non_nul` qui contient tous les coefficients non nuls de la matrice des contraintes
- `colname` : nom des nouvelles colonnes s'il y en a (facultatif)
- `rowname :` un tableau de `char *` qui contient le nom des contraintes (facultatif)

Comme on le constate dans cette fonction, dans **Cplex** la matrice des contraintes est gérée comme une matrice creuse (i.e. matrice contenant un grand nombre de 0). Afin de gagner de l'espace mémoire on ne stocke pas les 0. Ainsi les matrices sont représentées par 3 vecteurs : le premier (`rmatval`) contient les coefficients non nuls (dans l'ordre : coefficients de la première ligne de gauche à droite, puis de la seconde, etc...), le second (`rmatind`) contient les numéros de colonnes des coefficients non nuls (dans le même ordre que précédemment), le dernier (`rmatbeg`) contient les indices dans `rmatval` des coefficients qui sont les premiers de leur ligne.

Exemple :

Soit la matrice des contraintes *M* suivante :

$$M = \begin{pmatrix} 1 & 0 & -6 \\ 0 & 3 & 1 \end{pmatrix}$$

Attention : Les lignes et colonnes commencent à l'indice 0.

Cette matrice est décrite à l'aide des vecteurs `rmatbeg` `rmatind` et `rmatval` de la manière suivante :
`rmatval` = [1 -6 3 1] ; /* les valeurs non nulles */
`rmatind` = [0 2 1 2] ; /* les numéros de colonnes correspondant aux valeurs non nulles*/
`rmatbeg` = [0 2] ; /* les indices correspondant à des débuts de lignes dans `rmatval` */

Exemple d'utilisation de **`CPXaddrows :`**

```
double * rhs = (double*) malloc( NB_ROW * sizeof(double) );
char * sens = (char*) malloc( NB_ROW * sizeof(char) );
double * rmatval = (double*) malloc( NB_NON_NUL * sizeof(double) );
int * rmatbeg = (int*) malloc( NB_ROW * sizeof(int) );
int * rmatind = (int*) malloc( NB_NON_NUL * sizeof(int) );

status = CPXaddrows (env, lp, 0, NB_ROW, NB_NON_NUL, rhs, sens, rmatbeg, rmatind,
rmatval, NULL, NULL);
```

```
if (status)
  printf("echec lors de la creation des contraintes\n");
```

Remarque : on peut utiliser la fonction :

```
int CPXwriteprob(CPXCENVptr env, CPXCLPptr lp, const char * filename_str, const char * filetype_str)
```

pour écrire dans un fichier le programme linéaire que l'on vient de créer. Si le paramètre `filetype_str` est `NULL` alors **Cplex** "devine" le type de fichier à créer en utilisant l'extension du nom de fichier passé en paramètre (`filename_str`).

Exemple d'utilisation pour créer un fichier de type lp :

```
CPXwriteprob(env, lp, "monProg.lp", NULL);
```

Etape 4 : Lancer l'optimisation

Par défaut **Cplex** minimise la fonction objectif. Pour maximiser il faut appeler la fonction **`CPXchgobjsen`** avec le paramètre **`CPX_MAX`** :

```
CPXchgobjsen(env, lp, CPX_MAX);
```

La fonction **`CPXmipopt`** lance la résolution du PL en nombres entiers et retourne un entier qui indique si l'optimisation c'est bien passée :

```
  status = CPXmipopt (env, lp); /*resolution d'un PL mixte*/
  if ( status )
    printf("echec lors de l'optimisation\n");
```

Pour résoudre un PL fractionnaire utiliser :

```
 int CPXlpopt(CPXCENVptr env, CPXLPptr lp)
```

Etape 5 : Récupérer la solution

On récupère la valeur des différentes variables avec **`CPXgetmipx`** et la valeur de la fonction objectif avec **`CPXgetmipobjval`** :

```
  double * x = (double*)malloc( nbVar * sizeof(double) ); /* valeur des variables */
  double objval; /* valeur de la fonction objectif */

  /* récupère la valeur des variables (pour un PL mixte résolu avec CPXmipopt) */
  CPXgetmipx (env, lp, x, 0, nbVar-1);

  /* récupère la valeur de la fonction objectif (pour un PL mixte résolu avec
CPXmipopt) */
  CPXgetmipobjval(env, lp, &objval);
```

Dans la cas général (PL mixte ou fractionnaire) on peut utiliser :

```
int CPXsolution (CPXCENVptr env,
                 CPXCLPptr lp,
                 int *lpstat_p, /* statut de l'optimisation */
                 double *objval_p, /* valeur fonction objectif */
                 double *x, /* valeur des variables */
                 double *pi, /* valeur des variables duales pour chaque contrainte */
                 double *slack, /* valeur des variables d'écart */
                 double *dj) /* valeur de coûts réduits pour chaque variable */
```

Les 3 derniers paramètres sont facultatifs, on peut passer le pointeur `NULL` si on ne souhaite pas récupérer les valeurs associées.

Etape 6 : Libérer l'environnement

Il faut commencer par désallouer le problème avec la fonction **`CPXfreeprob`** puis quitter l'environnment avec la fonction **`CPXcloseCPLEX :`**

```
if ( lp )
{
  CPXfreeprob (env, &lp);
}

if ( env )
{
  CPXcloseCPLEX (&env);
}
```

Remarque : toutes les fonctions de **Cplex** renvoient un code erreur (qui vaut 0 si aucune erreur n'est détectée). Il n'est pas toujours indiqué dans les exemples afin de ne pas surcharger le code. Cependant dans un programme ce code doit être testé et pris en compte.

Etape 7 : Compilation

Pour compiler il faut indiquer au compilateur le répertoire des fichiers d'entête de **Cplex** ainsi que celui des librairies. Pour cela :

1. à la compilation ajouter l'option `-I` suivi du chemin d'accès vers les fichiers d'entête.

2. à l'édition des liens ajouter cette même option ainsi que l'option `-L` suivi du chemin d'accès vers les librairies et les flags `-lcplex -lm -pthread`

Exemple de makefile :

```
SYSTEM     = x86-64_sles10_4.1
LIBFORMAT  = static_pic
CPLEXDIR   = ILOG/CPLEX_Studio/cplex

CPLEXLIBDIR   = $(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)

CPLEXFLAGS    = -lcplex -lm -pthread
CFLAGS        = -Wall

CC            = gcc
OBJECTS       = main.o #liste de tous les .o a creer a partir des .c
```

```
exec : $(OBJECTS)
	$(CC)  -I$(CPLEXDIR)/include -L$(CPLEXLIBDIR)  -o exe $(OBJECTS) $(CPLEXFLAGS)

.c.o :
	$(CC) $(CFLAGS) -I$(CPLEXDIR)/include -c $< -o $@

clean :
	rm $(OBJECTS) exe
```

Exemple de programme complet :

Remarque : afin de mettre l'accent sur les fonctions **Cplex** et de ne pas surcharger le code la gestion des erreurs n'est pas montrée. Le programme pourrait être découpé en sous fonctions afin d'en faciliter la lisibilité et la maintenance :

```c
#include <ilcplex/cplex.h>
#include <stdlib.h>
#include <string.h>

#define NB_VAR 3
#define NB_ROW 2
#define NB_NON_NUL 4

/*
Min x1 + x2 + x3
x1 -        6 x3 >= 3
3 x2  +  x3 >= 4
x1, x2, x3 entiers
*/

int main (int argc, char **argv)
{
    int status, i;
    double objval; /*valeur de la fonction objectif*/

    /* allocation des vecteurs pour la creation des variables */
    double * coeff = (double *) malloc(NB_VAR*sizeof(double));
    char * ctype = (char*) malloc(NB_VAR*sizeof(char));

    /* allocation des vecteurs pour la creation des contraintes */
    double * rhs = (double*) malloc( NB_ROW * sizeof(double) );
    char * sens = (char*) malloc( NB_ROW * sizeof(char) );
    double * rmatval = (double*) malloc( NB_NON_NUL * sizeof(double) );
    int * rmatbeg = (int*) malloc( NB_ROW * sizeof(int) );
    int * rmatind = (int*) malloc( NB_NON_NUL * sizeof(int) );

    /* allocation du vecteur solution */
    double * x = (double*)malloc( NB_VAR * sizeof(double) ); /* valeur des variables */

    /*------------------------------------------------------------*/
    /* 1. initialiser environnement Cplex */
    CPXENVptr env = CPXopenCPLEX (&status);

    /*------------------------------------------------------------*/
    /* 2. creation d'un objet de type "problème Cplex"*/
    CPXLPptr lp = CPXcreateprob (env, &status, "monProbCplex");

    /*------------------------------------------------------------*/
    /* 3. creation des variables et contraintes */
    /* 3.1. creation des variables */
    for (i = 0; i < NB_VAR; ++i)
    {
        coeff[i] = 1; /* coefficient 1 dans la fonction objectif */
        ctype[i] = 'I'; /* variable est entière */
    }

    CPXnewcols (env, lp, NB_VAR, coeff, NULL, NULL, ctype, NULL);
```

```
    /* 3.2. creation des contraintes : remplissage de la matrice des contraintes */
    /* 1ere contrainte :  x1 - 6 x3 >= 3 */
    rhs[0] = 3;
    sens[0] = 'G';
    rmatval[0] = 1; rmatval[1] = -6;
    rmatind[0] = 0; rmatind[1] = 2;
    rmatbeg[0] = 0;

    /* 2sd contrainte :  3 x2  +  x3 >= 4 */
    rhs[1] = 4;
    sens[1] = 'G';
    rmatval[2] = 3; rmatval[3] = 1;
    rmatind[2] = 1; rmatind[3] = 2;
    rmatbeg[1] = 2;

    CPXaddrows (env, lp, 0, NB_ROW, NB_NON_NUL, rhs, sens, rmatbeg, rmatind, rmatval, NULL, NULL);

    /* sauvegarde du PL créé dans un fichier .lp */
    CPXwriteprob( env,  lp, "test.lp", NULL);

    /*------------------------------------------------------------*/
    /* 4. resolution */
    CPXmipopt (env, lp);

    /*------------------------------------------------------------*/
    /* 5. affichage de la solution */
    /* récupère la valeur des variables (pour un PL mixte résolu avec CPXmipopt) */
    CPXgetmipx (env, lp, x, 0, NB_VAR-1);
    /* récupère la valeur de la fonction objectif (pour un PL mixte résolu avec CPXmipopt) */
    CPXgetmipobjval(env, lp, &objval);

    printf (" x1 = %f, x2 = %f, x3 = %f\n", x[0], x[1], x[2]);
    printf ("obj = %f \n", objval);

    /*------------------------------------------------------------*/
    /* 6. libération mémoire */
    free(coeff); free(ctype);
    free(rhs); free(sens); free(rmatval);
    free(rmatbeg); free(rmatind); free(x);

    CPXfreeprob (env, &lp);
    CPXcloseCPLEX (&env);

    return 0;
}
```

Références

[1] Introducing CPLEX, IBM Knowledge Center, http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.6.0/ilog.odms.cplex.help/CPLEX/GettingStarted/topics/preface/preface_synopsis.html?lang=en

[2] CPLEX Callable Library (C API) Reference Manual, IBM Knowledge Center, http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.6.0/ilog.odms.cplex.help/refcallablelibrary/homepageCrefman.html

[3] Tutoriel Callable Library, IBM Knowledge Center, http://www-01.ibm.com/support/knowledgecenter/SSSA5P_12.6.0/ilog.odms.cplex.help/CPLEX/GettingStarted/topics/tutorials/CallableLibrary/synopsis.html