OPL et l'environnement CPLEX Studio, quelques bases

Cette document a pour vocation à fournir un petit pense-bête concernant les bases du langage OPL.

1 Une brève introduction

OPL signifie *Optimization Programming Language*. Il s'agit d'un langage (un vocabulaire, une syntaxe et une grammaire) informatique permettant de spécifier des problèmes d'optimisation, qui peuvent être ensuite passés à un *solveur* pour résolution. Le fait qu'OPL soit un langage de modélisation implique qu'il n'a pas vocation à être exécuté (comme du C, du Java ou du Python par exemple). En revanche, il permet d'écrire des problèmes d'optimisation dans une syntaxe abstraite que les solveurs peuvent exploiter.

L'interface utilisée pour écrire des problèmes d'optimisation en OPL est l'IBM Ilog CPLEX Optimisation Studio. Il s'agit d'un environnement de développement (IDE) utilisant la plate-forme Eclipse.

CPLEX Studio (qui a porté précédemment d'autres noms, dont Ilog OPL Studio) permet de créer des fichiers selon le langage OPL. Ces fichiers permettent d'exprimer des problèmes :
- d'optimisation linéaire continue,
- d'optimisation linéaire en nombres entiers ou mixtes,
- de programmation par contraintes.

On notera qu'il ne permet pas d'exprimer de problèmes de programmation non-linéaire.

Pour résoudre les problèmes de programmation mathématique (PL et PLNE), CPLEX Studio fait appel au solveur linéaire CPLEX. Pour résoudre les problèmes de programmation par contraintes, il fait appel au programme CP Optimizer. Tous ces programmes sont aujourd'hui inclus dans la suite de logiciels proposée par IBM Ilog.

2 Structure d'un projet OPL

Un projet OPL est constitué de 4 types de fichiers :
- Un fichier projet (.project)
- Un ou plusieurs fichiers de modèles (.mod)
- Un ou plusieurs fichiers facultatifs de données (.dat)
- Un ou plusieurs fichiers facultatifs de paramètres (.ops)
- Un ou plusieurs fichiers de configuration d'exécution (.oplproject)

Les sections suivantes présentent ces différents fichiers dans l'ordre ci-dessus.

3 Le fichier projet

Ce fichier n'a pas réellement vocation à être ouvert dans un éditeur de texte. Il contient l'information permettant à CPLEX Studio d'afficher les différents autres fichiers contenus dans le projet et les liens entre eux. C'est donc le fichier .project qui permet d'afficher la racine d'un projet dans CPLEX Studio.

4 Les fichiers modèle

Les fichiers modèle (.mod) sont le coeur d'OPL. C'est dans un fichier modèle qu'on indique les constantes, les variables de décision, les contraintes et la fonction objectif d'un problème d'optimisation. Il faut donc bien faire l'association :

Un problème d'optimisation = un fichier .mod

Quelques exemples et éléments de syntaxe :

`;`	symbole de fin de déclaration
`//`	symbole de commentaire
`int n=5;`	`n` est un entier qui vaut 5
`float cost[1..n][1..n] = ...;`	`cost` est une matrice de taille `n`×`n` dont les éléments seront spécifiés dans un fichier .dat
`{string} names = ...;`	`names` est un ensemble discret de noms dont les éléments seront spécifiés dans un fichier .dat
`range I=1..n;`	`I` est un intervalle d'entiers de `1` à `n`
`dvar float+ weight;`	`weight` est une variable de décision continue positive
`dvar int ship[1..n];`	`ship` est un vecteur de `n` variables de décision entières
`maximize sum(i,j in I) w[i][j]*ship[i];`	spécifie une fonction objectif
`subject to {ship[1]<=8;}`	spécifie une liste de contraintes entre accolades
`forall(i in 2..n) ship[i]>=3;`	spécifie un ensemble de contraintes indicées par `i`

Quelques opérateurs numériques et logiques :

`+`	addition	`==`	égalité
`-`	soustraction	`<=`	inférieur ou égal
`*`	multiplication	`>=`	supérieur ou égal
`div`	division entière	`mod`	modulo

Le cadre de la programmation par contraintes utilise quelques autres mot-clés, plus souples, incompatibles avec la programmation mathématique (inégalités strictes, opérateur d'implication logique, etc.) qu'on passe sous silence pour l'instant.

Dans une utilisation plus avancée, les fichiers de modèle permettent également de définir des traitements en cours d'optimisation, des post-traitement des données, etc.

5 Les fichiers de données

Lorsqu'on définit un problème d'optimisation dans un fichier modèle, il est souvent préférable de déclarer les constantes du problème mais de ne pas en préciser la valeur en utilisant le symbole "`...`". Ce symbole indique que la valeur de ces constantes sera fournie dans un fichier de données. Cela permet de construire plusieurs jeux de données pour un même fichier modèle.

Quelques exemples :

```
n=3;
cost=[[0.0 1.5 2.3],
      [1.5 0.0 3.7],
      [2.3 3.7 0.0]];
```

Les fichiers de données permettent également de connecter les entrées et les sorties de l'opération d'optimisation à des sources externes de données (feuilles excel ou bases de données SQL par exemple). Un exemple est fourni plus loin.

6 Les fichiers de paramètres

Ces fichiers .ops permettent de paramétrer le solveur d'optimisation (ou de contraintes). Ils permettent notamment de gérer la stratégie de parcours de la recherche arborescente en PLNE, les heuristiques utilisées, les méthodes de construction de coupes, etc. On y accède directement via l'interface graphique de CPLEX Studio.

7 Les fichiers de configuration d'exécution

Ces fichiers .oplproject n'ont, comme les fichiers de projet et les fichiers de paramètres, pas vocation à être ouverts en tant que fichiers textes. On y accède directement via l'interface graphique CPLEX Studio. Un fichier de configuration d'exécution indique précisément "ce qu'il faut faire" dans une exécution donnée d'un solveur. Il précise donc quel modèle utiliser, avec quel fichier de données (si nécessaire) et quel jeu de paramètres (si nécessaire).

8 Les résultats d'optimisation

Une fois un problème spécifié grâce aux fichiers présentés ci-dessus, il est possible d'exécuter le solveur en lui précisant quel fichier de configuration d'exécution utiliser. Pour cela il existe un bouton dans la barre des raccourcis ou, pour ne pas se tromper de configuration d'exécution dans le cas où il y en a plusieurs, il est possible de faire un clic droit sur la configuration d'exécution choisie et de demander l'exécution.

L'interface affiche alors les résultats au fur et à mesure de l'exécution, notamment dans les onglets du bas de la fenêtre, indiquant la solution une fois qu'elle est trouvée (si elle est trouvée), les statistiques d'optimisation, etc. Il est également possible d'explorer les valeurs des variables de décision optimisées dans le "problem brower" sur le côté gauche de la fenêtre.

9 Un exemple de projet OPL pour le problème de p-médiane

On a un ensemble de clients et un ensemble de localisations candidates pour accueillir des entrepôts. Le problème consiste à choisir P entrepôts à ouvrir pour minimiser la somme des distances entrepôts/clients pondérées par des coefficients de priorité.

Soit :
- P le nombre d'entrepôts à ouvrir,
- I l'ensemble de clients,
- J l'ensemble de localisations candidates pour accueillir des entrepôts,
- w_i le coefficient de priorité du client i,
- d_{ij} la distance entre le client i et la localisation j,

les données du problème, et
- $x_j = 1$ si la localisation candidate j est choisie, 0 sinon
- $y_{ij} = 1$ si la demande pour le client i est satisfaite par l'entrepôt j, 0 sinon

les variables de décisions.

La formulation du problème est donc :

$$\min \sum_{i \in I} \sum_{j \in J} w_i d_{ij} y_{ij},$$

sous contraintes :

- $\sum_{j \in J} y_{ij} = 1 \ \ \forall i \in I,$ un client est servi par un seul entrepôt,
- $\sum_{j \in J} x_j = P,$ P entrepôts doivent être construits,
- $y_{ij} \leq x_j, \ \ \forall i \in I, \forall j \in J,$ un entrepôt non construit ne peut servir de client,
- $y_{ij} \in \{0,1\}, x_j \in \{0,1\}, \ \ \forall i \in I, \forall j \in J.$

9.1 Fichier modèle

```
//Data
int P = ...;
{string} Clients = ...;
{string} Local = ...;
int Priorite[Clients] = ...;
float Distance[Clients][Local] = ...;

//Variables
dvar boolean Entrepots[Local];
dvar boolean LivraisonClients[Clients][Local];

//Objectif
Minimize  sum( c in Clients , l in Local )
 Priorite[c]*Distance[c][l]*LivraisonClients[c][l];

//Constraintes
subject to {
        ctClients:
        forall( c in Clients )
            sum( l in Local ) LivraisonClients[c][l] == 1;
        ctNbEntrepot:
        sum( l in Local ) Entrepots[l] == P;
        ctOuvert:
        forall( c in Clients , l in Local )
            LivraisonClients[c][l] <= Entrepots[l];
}
```

9.2 Fichier données

```
P=2;
Clients={"Albert","Julien","Paul","Daniel"};
Local={"Brest","Rennes","Lorient"};
Priorite=[100,80,80,70];
Distance=
[[ 2 , 10 , 50 ],
[ 2, 10, 52 ],
[ 50, 60 , 3],
[ 40 , 60 , 1]];
```