IBM ILOG CPLEX Optimization Studio : Une introduction à OPL et Cplex Studio IDE

Hélène Toussaint, septembre 2016
dernière mise à jour : février 2019

But : apprendre à résoudre un programme linéaire à l'aide d'OPL

Prérequis : avoir une bonne connaissance de la programmation linéaire

Table des matières

1 Introduction

IBM ILOG CPLEX Optimization Studio regroupe un ensemble d'outils pour la programmation mathématique et la programmation par contraintes. Il associe :

- un environnement de développement intégré (*Integrated Development Environment* - IDE) nommé Cplex Studio IDE (sous Windows) ou oplide (sous Linux),
- un langage de modélisation : le langage OPL (*Optimization Programming Language*),
- deux solveurs : IBM ILOG CPLEX pour la programmation mathématique (résolution de programmes linéaires en nombres fractionnaires, mixtes ou entiers et de programmes quadratiques) et IBM ILOG CP Optimizer pour la programmation par contraintes.

Par défaut c'est le solveur CPLEX qui est activé.

Ce document portera uniquement sur l'utilisation de CPLEX via l'IDE Cplex Studio. On abordera les principales fonctionnalités de Cplex Studio IDE et d'OPL à l'aide d'exemples. Les captures d'écran sont réalisées sous Windows avec la version de Cplex Studio IDE livrée avec CPLEX 12.6.

Pour plus de détails, le site d'IBM ([1]) offre une documentation complète sur l'IDE ([2]) et sur OPL ([3]).

2 Le langage OPL (*Optimization Programming Language*)

Le langage utilisé dans Cplex Studio IDE est OPL (*Optimization Programming Language).* Il s'agit d'un langage de modélisation qui permet d'écrire facilement des programmes linéaires (ou quadratiques) grâce à une syntaxe proche de la formulation mathématique. Par ailleurs OPL offre à l'utilisateur la possibilité de séparer le modèle des données, de ce fait un même modèle peut être facilement testé avec différents jeux de données.

OPL fonctionne par projets : pour résoudre un modèle l'utilisateur doit créer un projet OPL dans Cplex Studio IDE qui doit contenir au minimum un fichier "modèle" et un fichier de "configuration d'exécution". En effet chaque projet est constitué de plusieurs types de fichiers :

- un fichier modèle (`.mod`) qui contient le modèle à résoudre,
- un fichier de données (*facultatif*) qui contient les données pour un modèle,
- un fichier de paramètres (`.ops`) (*facultatif*) qui permet de paramétrer le solveur CPLEX,
- un fichier de configuration d'exécution (`.oplproject`) qui indique à l'IDE ce qu'il doit faire quand l'utilisateur demande l'exécution du projet. C'est à dire quel est le modèle à résoudre et quels sont les paramètres et les données (s'il y en a).

La section suivante montre comment créer un projet pour résoudre un petit problème linéaire à 2 variables.

3 Premier exemple : création et résolution d'un modèle à 2 variables et 2 contraintes

3.1 Premiers éléments de syntaxe OPL

OPL connait les types entiers (`int`), entiers positifs (`int+`), flottants (`float`) et flottants positifs (`float+`).

- Les variables de décision se définissent en utilisant le mot clé `dvar` suivi de leur type,
- la fonction objectif est précédée du mot clé "`minimize`" ou "`maximize`" en fonction du sens d'optimisation,
- les contraintes sont dans un bloc entre accolades et précédées des mots clés "`subject to`",
- les principaux opérateurs numériques et logiques sont
    - +, -, *, / pour l'addition, la soustraction, la multiplication et la division,
    - `div` et `mod` pour la division entière et le modulo,
    - <=, >= et == pour les comparaisons,
- les commentaires sont soit entourés de `/* ... */` (comme en C), soit précédé de `//` (comme en C++) s'ils sont sur une seule ligne.

Remarque : la division réelle (/) n'est autorisée que sur des types `float`, pour les entiers il faut utiliser `div`.

Il existe bien d'autres éléments de syntaxe (pour définir des ensembles, des tableaux, etc...) que nous verrons par la suite.

3.2 Création du projet et du modèle

Tout d'abord il faut créer un projet OPL dans lequel on pourra définir notre modèle.
Pour cela dans Cplex Studio il faut cliquer sur `Fichier / Nouveau / Projet OPL`. Une fenêtre s'ouvre : entrez un nom de projet, choisissez son emplacement (dossier parent) et cochez "`Création d'un modèle`" ainsi que "`ajouter une configuration d'exécution par défaut`".

NB. Si vous oubliez de cocher les cases "`modèle`" ou "`configuration d'exécution par défaut`" vous avez toujours la possibilité de créer les fichiers correspondant dans `Fichier / Nouveau`, même après la création du projet (voir section 7).

Dans le fichier modèle (`.mod`) on entre le modèle suivant (voir capture d'écran Figure 1) :

Minimiser $3x + 2y$ Sous	`dvar float x;` `dvar float y;`

$x - y \geq 5$
$3x + 2y \geq 10$
	`minimize`
	`  3*x + 2*y;`
	`subject to {`
	`  x - y >= 5;`
	`  3*x + 2*y >= 10;`
	`}`

formulation mathématique	*formulation dans le langage OPL*

3.3 Résolution

Pour lancer la résolution il faut faire un clic droit sur "`Configuration d'exécution`" dans l'onglet Projets OPL situé à gauche de la fenêtre principale puis "`exécuter / configuration d'exécution par défaut`". Le bouton *exécuter*  dans la barre d'outils permet de lancer une nouvelle fois la dernière configuration exécutée.

Une fois le modèle résolu, plusieurs informations s'affichent dans les onglets situés en bas de la fenêtre principale (sous le fichier modèle). Par exemple :

- l'onglet "solution" donne des informations sur la solution (optimalité, coût de la fonction objectif, valeur des variables à l'optimalité…),
- l'onglet "journal du moteur" affiche la sortie de CPLEX,
- l'onglet "statistique" montre différentes mesures liées à la résolution (nombre d'itérations du simplexe, nombres de nœuds de branchement…).

Tout en bas à droite de la fenêtre s'affiche le chronomètre qui mesure le temps mis par CPLEX pour résoudre le modèle.

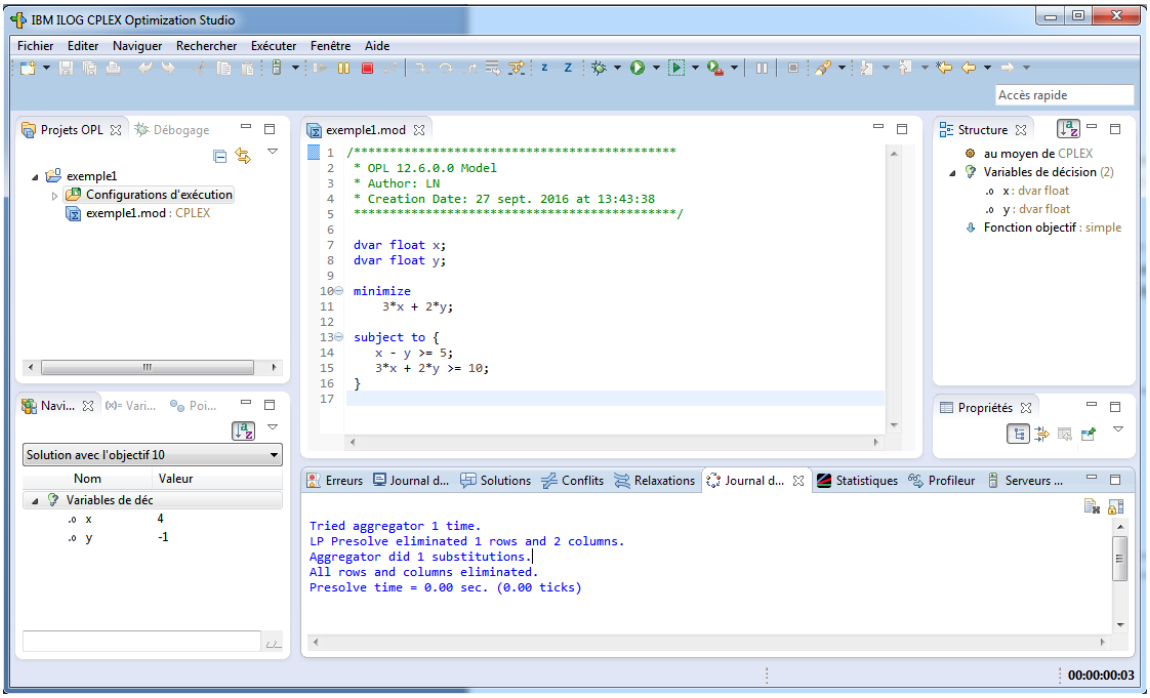

Figure 1 : résolution d'un modèle à 2 variables, 2 contraintes dans Cplex Studio IDE

4 Deuxième exemple : utilisation de tableaux et lecture dans un fichier de données

4.1 Formulation mathématique du problème

Pour illustrer l'utilisation de tableaux 1D on programme le problème du sac à dos. Dans ce problème on dispose de n objets et d'un sac pouvant supporter un poids maximal P. Chaque objet est associé à un poids et à une valeur et on souhaite remplir le sac de sorte de maximiser la somme des valeurs des objets qu'il contient sans dépasser le poids total autorisé.

On note :

- n le nombre d'objets
- v_i la valeur de l'objet $i, i = 1 \dots n$
- p_i le poids de l'objet $i, i = 1 \dots n$
- P le poids total supporté par le sac
- x_i la variable de décision qui vaut 1 si l'objet i est choisi, 0 sinon

La formulation mathématique est rappelée ci-dessous :

$$\text{Maximiser} \sum_{i=1}^{n} v_i x_i$$

$$\text{Sous}$$

$$\sum_{i=1}^{n} p_i x_i \leq P$$

$$x_i \in \{0,1\}$$

4.2 Lecture dans un fichier de données

Il est recommandé de ne pas initialiser les données directement dans le modèle afin de pouvoir utiliser facilement le modèle avec différents jeux de données. On utilise alors des fichiers de données, qui doivent avoir l'extension `.dat`, et être dans le même projet que le modèle pour lequel ils vont être utilisés.

La lecture dans un fichier de données est facile avec OPL : à la déclaration d'une variable on utilise "`=...;`" pour indiquer que sa valeur est à lire dans un fichier `.dat`. Par exemple pour déclarer un entier n et indiquer à OPL de lire sa valeur dans un fichier de données on écrit dans le fichier modèle (`.mod`) :

```
int n = ...; //déclare un entier et lit sa valeur dans un fichier .dat
```

Dans le fichier .dat on aura :

```
n = 5; //valeur de n
```

Le nom de la variable dans le fichier modèle doit être rigoureusement le même que celui dans le fichier de données. On n'est pas obligé de mettre les variables dans l'ordre de leur déclaration,

cependant l'ordre est important s'il y a une relation de dépendance entre les variables (une variable initialisée à l'aide d'une autre).

4.3 Déclaration d'intervalles et de tableaux

Pour écrire le problème du sac à dos à l'aide d'OPL on a besoin, en plus des éléments de syntaxe vus aux paragraphes précédents, de savoir utiliser les tableaux et les intervalles.

4.3.1 Les intervalles

On crée un intervalle avec le mot clé `range` :

```
range I = 1..10; //définit l'intervalle entier [|1,10|]
range float R = 1.0..100.0; //définit l'intervalle de flottants [1,100]
```

Les intervalles entiers peuvent être utilisés pour :

- Définir des tableaux :

```
int T[I]; // T est un tableau indicé de 1 à 10
```

- Construire des expressions itératives (boucles "pour") :

```
forall(i in I)
 {
      ...
 }
```

- faire des sommes :

```
dvar boolean x[I];
sum( i in I ) x[i] <= 1;
```

Les intervalles entiers ou flottants sont aussi utilisés pour définir le domaine d'une variable de décision :

```
dvar int x in I;
dvar float y in R;
```

On peut également parcourir un intervalle et n'utiliser que les éléments de l'intervalle qui respectent une certaine condition en utilisant l'opérateur de filtrage `:`

```
dvar boolean x[I];

// on suppose qu'on a un tableau tab indicé sur I contenant des entiers, on écrit
la contrainte x[i] <= 1 uniquement si tab[i]=0 :
sum( i in I : tab[i]==0) x[i] <= 1;
```

4.3.2 Les tableaux

On définit un tableau en donnant son type et son domaine d'indexation :

```
range I = 1..10;
int T[I];
//ou directement (sans déclarer l'intervalle au préalable)
int T[1..10];
```

Sur le même principe on peut déclarer un tableau à double indice (i.e. une matrice) :

```
int T[1..2][1..3]; //matrice 2 lignes, 3 colonnes
```

On peut initialiser les tableaux soit :

- au moment de la déclaration dans le fichier modèle :

```
int T[1..4] = [50, 14, 19, 20]; // tableau de 4 entiers
float F[1..5] = [0.7, 1.6, 1.5, 2.6, 5.8]; // tableau de 5 flottants
string S[1..2] = ["bonjour", "cplex"];// tableau de 2 chaines de caractères
```

- en lisant les valeurs dans un fichier de données (le nom du tableau doit alors être le même dans le fichier de données et dans le fichier modèle, les éléments du tableau sont séparés par des virgules) :

```
int T[1..5] = ...;
//T = [1, 6, 8, 2, 7]; dans le fichier .dat
int M[1..2][1..3] = ...;
//M = [[11,12,13],[21,22,23]];dans le fichier .dat
```

- en utilisant une expression arithmétique au moment de la déclaration :

```
int T[i in 1..10] = i+1; //T[i] = i+1
int M[i in 1..10][j in 1..10] = 10*i + j; //M[i][j] = 10*i + j
```

- en utilisant des instructions de *preprocessing* avec *IBM ILOG Script* (voir section 5).

4.4 Formulation du problème avec OPL

Pour ce problème on crée un projet OPL (`Fichier / Nouveau / Projet OPL`) et, dans la fenêtre "créer un projet" qui s'ouvre, on coche les cases "`ajouter une configuration d'exécution par défaut`", "`Création d'un modèle`" ainsi que "`Créer des données`".

Dans le fichier modèle, on entre le modèle précédent dans le langage OPL :

```
//lire dans un fichier le nb d'objet et le poids max
int nbObjet = ...;
int poidsMax = ...;
```

```
//déclarer un intervalle d'entiers de 1 à nbObjet
range objets = 1..nbObjet;

//déclarer des tableaux indexés sur les objets,
//ils seront remplis en lisant le fichier de données
int poids[objets] = ...;
int valeur[objets] = ...;

//déclarer les variables de décisions
dvar boolean x[objets];

//modele
maximize
  sum(i in objets) valeur[i] * x[i];

subject to {
       sum( i in objets )
        poids[i] * x[i] <= poidsMax;
}
```

Contenu du fichier modèle (.mod) pour le problème du sac à dos

et dans le fichier de données on entre les données déclarées dans le modèle et suivies de "= ...;" :

```
nbObjet = 5;
poidsMax = 100;

valeur = [6, 10, 5, 4, 3];
poids = [56, 62, 32, 48, 25];
```

Contenu du fichier de données (.dat) pour le problème du sac à dos

5 Les blocs d'exécution (*IBM ILOG Script for OPL*)

Si l'on souhaite prétraiter ou post traiter les données dans le fichier modèle (initialisation via des calculs complexes, affichage de la solution…) on ne peut pas utiliser directement OPL qui sert uniquement à la modélisation. Cependant il existe un langage de script pour OPL appelé *IBM ILOG Script for OPL* (qui est une implémentation de JavaScript). *IBM ILOG Script* peut être utilisé pour :

- Le prétraitement ou le posttraitement des données,
- Le réglage des paramètres du solveur CPLEX (si on n'a pas inclus de fichier de paramètres),
- Le contrôle du flot d'exécution (ce qui permet par exemple d'implémenter des schémas de décomposition type génération de colonnes par exemple, ou de fournir une solution initiale).

Il s'agit d'un langage impératif qui permet d'écrire des affectations, des fonctions, des opérations numériques, des boucles, des conditionnelles…

Il existe trois types de blocs d'exécution :

- les blocs définis par « `execute{…}` » dans le fichier modèle, ils contiennent des instructions pour le pré ou post traitement, et éventuellement des instructions pour le paramétrage du solveur,

- les blocs définis par « `main{…}` » dans le fichier modèle, ils servent à contrôler le flot d'exécution (il y a au plus un bloc main par fichier modèle),
- les blocs définis par « `prepare{…}` » dans le fichier de données, ils permettent de manipuler les données avant de les utiliser dans le fichier modèle.

On donne dans cette section seulement les principaux éléments de syntaxe et les exemples concernent uniquement les blocs d'exécution « `execute{…}` » définis dans le fichier modèle, pour plus de détails voir la documentation d'IBM en ligne ([4]).

5.1 Forme générale d'un bloc d'exécution

Un bloc d'exécution est désigné par le mot clé `execute` suivi éventuellement du nom du bloc. Les instructions (dans le langage *IBM ILOG Script*) sont entre accolades. Le point-virgule à la fin de chaque instruction est optionnel si les instructions sont sur des lignes différentes, il est obligatoire si elles se suivent sur une même ligne.

```
execute <nomDuBloc>
{
	instruction1;
	instruction2;
	…
}
```

Un bloc d'exécution placé avant le modèle est pris en compte dans le prétraitement (avant la résolution du modèle par CPLEX). Un bloc placé après le modèle est pris en compte dans le post traitement (après la résolution du modèle).

5.2 Quelques éléments de syntaxe

La syntaxe d'*IBM Script* est proche de la syntaxe des langages procéduraux comme le C.

5.2.1 Variables

Dans *IBM Script*, toutes les variables sont déclarées à l'aide du mot clé `var` quel que soit leur type. Elles sont locales au bloc dans lequel elles ont été déclarées.

5.2.2 Instructions conditionnelles

Les instructions conditionnelles utilisent les mots clés `if / else` :

```
	if (condition)
		{/*instructions1*/}
	else
		{/*instructions2*/}
```

Les accolades sont facultatives s'il n'y a qu'une instruction. La clause `else` peut être omise.

5.2.3 Boucle « Pour »

Les boucles « pour » utilisent le mot clé `for`. Elles peuvent être définies en donnant la valeur initiale du compteur de boucle, le critère d'arrêt et le pas (comme en C) :

```
for (var i = 1; i <= N; i++)
{
      writeln(poids[i]); //poids est un tableau 1D de taille N
}
```

ou elles peuvent être définies directement à l'aide de l'ensemble qu'elles parcourent :

```
for (var i in poids)
{
      writeln(poids[i]); //poids est un tableau 1D
}
```

Remarques :

- le compteur de boucle `i` n'est pas typé, il est simplement déclaré avec le mot clé `var`.
- `writeln()` est une fonction permettant d'afficher à l'écran sur une ligne (inclut le retour à la ligne).

5.2.4 Boucle « While »

On utilise le mot clé `while`. Les instructions sont exécutées tant que la condition est vraie.

```
while (condition)
{
     Instructions ;
}
```

6 Troisième exemple : tableau à 2 dimensions et post traitement des données

Pour illustrer l'utilisation des tableaux à 2 dimensions avec OPL on programme le problème des N reines. Dans ce problème le but est de placer N reines sur un échiquier pondéré de taille $N \times N$, sans qu'elles ne se menacent, et de manière à maximiser la somme des poids des cases occupées par une reine. Deux reines se menacent si elles sont sur une même ligne, une même colonne ou une même diagonale.

Dans l'exemple proposé la taille N de l'échiquier est lue dans un fichier de données. La matrice des poids est initialisée de manière aléatoire avec des entiers compris entre 0 et 99. Le post traitement consiste à afficher à l'écran l'échiquier avec la position des reines. La déclaration des données ainsi que la définition du modèle sont écrits avec le langage OPL tandis que le post traitement est écrit avec *IBM ILOG Script*.

```
 int N = ...; //taille echiquier (lu dans fichier .dat)
 range I = 1..N;

 // poids de chaque case de l'echiquier initialise aleatoirement entre 0 et 99
 int poids[i in I][j in I] = rand(100);
 dvar boolean X[I][I]; //x[i][j] = vrai si on a une reine en (i,j)


//----------------------------MODELE--------------------------------
maximize
  sum(i in I) sum(j in I) poids[i][j] * X[i][j];

subject to
{
	//une reine par ligne
	forall(i in I)
		sum(j in I) X[i][j] == 1;

	//une reine par colonne
	forall(j in I)
		sum(i in I) X[i][j] == 1;

	//une reine max par diagonale ascendante gauche droite, triangle superieur
	forall (i in 2..N)
	  sum (j in 0..i-1) X[i-j][j+1] <= 1;

	//une reine max par diagonale ascendante gauche droite, triangle inferieur
	forall (j in 2..N-1)
	  sum (i in j..N) X[i][j+N-i] <= 1;

	//une reine max par diagonale descendante, triangle superieur
	forall (j in 1..N-1)
	  sum (i in 1..N-j+1) X[i][j+i-1] <= 1;

	//une reine max par diagonale descendante, triangle inferieur
	forall (i in 2..N-1)
	  sum (j in 0..N-i) X[i+j][j+1] <= 1;
}

//----------------------------POST TRAITEMENT--------------------------------
// affichage des reines sur l'échiquier
execute DISPLAY
{
	for(var i in I)
	{
		for(var j in I)
		{
			if (X[i][j] == 1)
				write("| R ");
			else
				write("|   ");
		}
		write("|\n");
	}
}
```

Fichier modèle : contient le modèle et l'affichage

L'affichage est réalisé dans l'onglet "Journal de script" dans la fenêtre située en bas de l'écran (voir Figure 2).

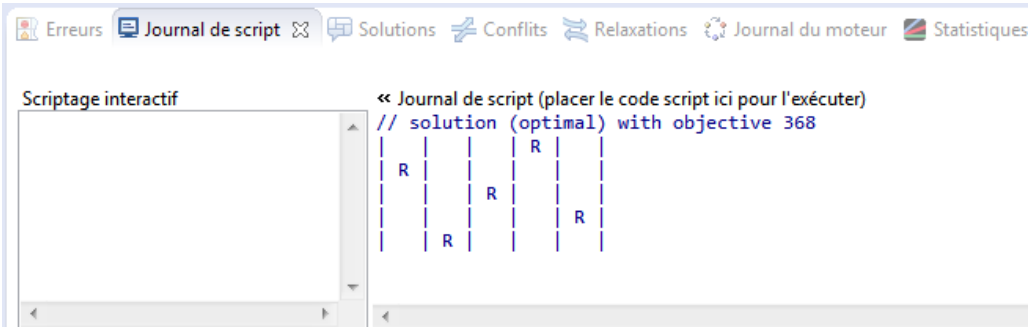

Figure 2 : affichage de la solution

7 Projet OPL et configurations d'exécution

On a vu dans la section 2 qu'OPL fonctionnait par projet, et que chaque projet était constitué de différents types de fichiers. On donne ici quelques informations supplémentaires sur les fichiers de configuration d'exécution et les fichiers de paramètres.

7.1 Les configurations d'exécution

Une configuration d'exécution contient une combinaison de fichiers (modèle, données et paramètres) associés au sein d'un même projet. Elle permet de définir quel modèle on veut exécuter avec quelles données et sous quel paramétrage du solveur. Une configuration d'exécution doit inclure au minimum un fichier modèle. Pour lancer une configuration on fait un clic droit sur son nom (dans le navigateur de projet) puis "exécuter cette configuration" (voir Figure 3).

Attention : quand on clique sur le bouton *exécuter*  , Cplex Studio lance la dernière configuration exécutée quel que soit le projet sélectionné dans le navigateur Projets OPL.

On peut créer une nouvelle configuration d'exécution en faisant un clic droit sur le nom du projet puis "`Nouveau / Configuration d'exécution`". On remplit la nouvelle configuration à l'aide d'un glisser-déposer (*drag-and-drop)* des fichiers souhaités. On peut créer autant de configurations que l'on souhaite (voir Figure 4).

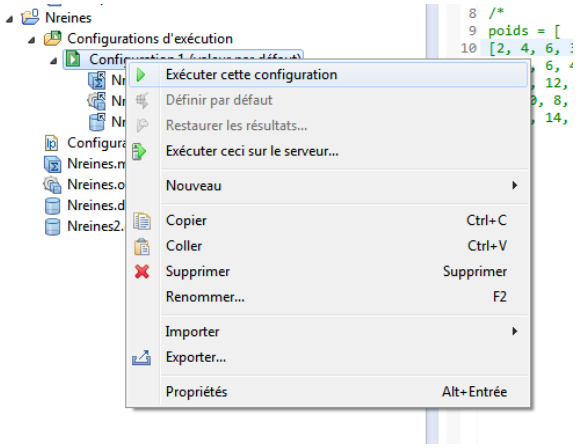

Figure 3 : choix de la configuration à exécuter

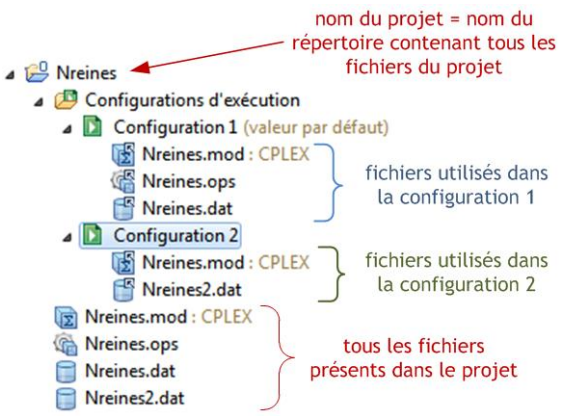

Figure 4 : exemple de projet avec 2 configurations d'exécution

7.2 Fichier de paramètres (.ops)

Si on souhaite paramétrer le solveur, le plus facile est d'ajouter un fichier de paramètres à la configuration d'exécution courante (clic droit sur la configuration puis "`Nouveau / Paramètres`"). Lorsqu'on double clic sur ce fichier, une fenêtre s'ouvre et permet d'explorer un grand nombre de paramètres liés au solveur et de fixer leur valeur (voir Figure 5).

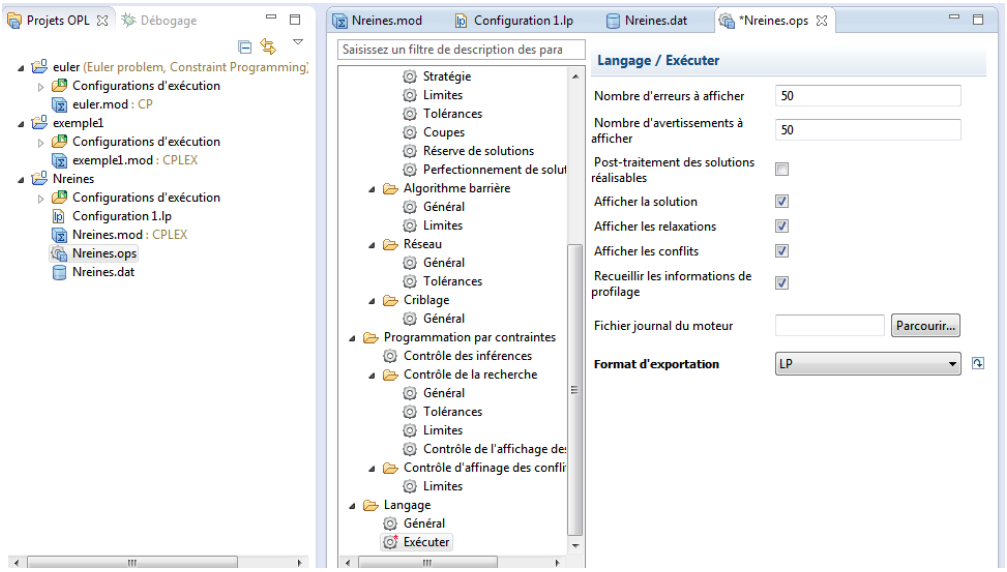

Figure 5 : exemple de fichier de paramètres

Le fichier de paramètres permet également de définir un format d'exportation pour le modèle (dans l'onglet "`Langage / Exécuter`"). Dans l'exemple de la Figure 5 on a choisi le format lp. Un fichier .lp sera donc automatiquement généré au moment de l'exécution du modèle associé.

Les noms de variables dans le fichier lp sont les mêmes que dans OPL. On peut également donner un nom aux contraintes, il suffit de mettre le nom souhaité suivi de ":" devant la contrainte. Par exemple, on peut définir les deux premiers jeux de contraintes de l'exemple des N reines de la manière suivante :

```
forall(i in I)
      Lig:          sum(j in I) X[i][j] == 1;

forall(j in I)
      Col:          sum(i in I) X[i][j] == 1;
```

8 Références

[1] IBM, IBM Knowledge Center, https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html

[2] IBM, IBM ILOG CPLEX Optimization Studio Getting Started with the IDE, https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/gsoplide.pdf

[3] IBM, IBM ILOG CPLEX Optimization Studio OPL Language Reference Manual (V12 Release 6), http://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.studio.help/pdf/opl_langref.pdf

[4] IBM, IBM Knowledge Center, Utilisation d'IBM ILOG Script for OPL, http://www.ibm.com/support/knowledgecenter/fr/SSSA5P_12.6.2/ilog.odms.ide.help/OPL_Studio/usroplide/topics/opl_ide_script.html