ED SMI
Année 2012-2013

CS & Optim TP
Mathias Kleiner
mathias.kleiner@ensam.eu

Méthodes et outils d'optimisation

Nous utiliserons tout au long de ces travaux pratiques la suite logicielle IBM/ILOG Optimization Studio. Dans ce logiciel, un problème à résoudre est contenu dans un projet. Un projet offre la possibilité de classer la connaissance en 3 parties : la modélisation du problème (le fichier *.mod*), les données de l'instance (fichier *.dat*), les options de résolution (fichier *.ops*). Les modèles formels des problèmes sont décrits à l'aide d'un langage informatique appelé OPL.
Le logiciel fournit un ensemble de tutoriaux et d'exemples associés permettant de découvrir le fonctionnement d'un projet, le langage OPL, et d'expérimenter/illustrer les différentes méthodes vues en cours. Pour démarrer les tutoriaux qui nous intéressent, il faut se rendre dans la table des matières de l'aide puis dans la partie *IDE and OPL* → *Optimization Programming Language (OPL)* → *Language User's Manual* → *Introduction to OPL* → *A short tour of OPL*. Chaque tutoriel fait référence à des exemples que l'on peut charger par le menu *Fichier* → *Importer* → *Exemple*. Il suffit ensuite de sélectionner l'exemple (on peut filtrer la liste par méthode, par nom, etc.). Notez que l'aide du logiciel est très complète, n'hésitez pas à y rechercher l'information lorsqu'elle n'est pas directement présente dans les tutoriaux.
Pour chacun des exercices qui suivent, observez le comportement du logiciel si vous changez les valeurs, complexifiez le problème ou ajoutez des contraintes qui le rendent plus difficile voire insoluble.

Exercice 1: Programmation linéaire

Suivre le tutoriel *Linear programming : a production planning example*, associé à l'exemple *volsay*.
Sur le même principe, créer un projet pour résoudre le problème *navets et courgettes* vu en cours.

Exercice 2: Programmation linéaire en nombre entiers

Créer un projet pour résoudre le problème *sac à dos* vu en cours.
Suivre le tutoriel *A typical integer program : the knapsack problem* , associé à l'exemple *(multi-)knapsack*.
Importer, executer et comprendre le problème *warehouse* (le problème est décrit dans l'aide).
Créer un projet pour résoudre le problème *work scheduling* vu en cours.

Exercice 3: Programmation par contraintes

Importer, executer et comprendre le problème *color*.
Sur le même principe, créer un projet pour résoudre le problème des *reines* vu en cours.
Suivre le tutoriel *Constraint programming : an inventory matching problem* , associé à l'exemple *steelmill*.