

# INTRODUCTION A CPLEX

A.Lisser

M1 MIAGE

Informatique 2015

# Qu'est-ce que CPLEX ?

- CPLEX est principalement un solveur de PL(NE)
- = Outil (logiciel) pour résoudre des PL(NE)
- Résolution de PL par
  - ✓ Algorithme primal du simplexe
  - ✓ Algorithme dual du simplexe
  - ✓ Algorithme de points intérieurs
- Résolution de PLNE par B&B (B&C)

# *Fonctionnalités principales*

- Possibilité d'utiliser CPLEX via :
  - ✓ Un mode interactif, qui permet
    - ❖ Définition des PL(NE) « à la main »
    - ❖ Définition des PL(NE) via des fichiers de données texte (format LP)
  - ✓ API C++
  - ✓ API Java

MODE INTERACTIF

# Principales commandes du mode interactif

- *help (ou h) : liste les commandes disponibles*
- *help (ou h) nom\_commande : fournit de l'aide sur la commande nom\_commande*
- *enter nom\_pb : saisie de nom\_pb en mode interactif*
- *optimize : résolution du PL*
- *xecute ligne\_commande : exécute ligne\_commande*
- *set : liste les paramètres de CPLEX pour modification*
- *quit : termine CPLEX*

# *Saisir les données d'un problème en mode interactif (format LP)*

- *Principaux mots-clés de la saisie :*
  - ✓ *maximize ou minimize (fonction économique)*
  - ✓ *subject to ou st (contraintes)*
  - ✓ *bounds (bornes individuelles sur les variables)*
  - ✓ *generals ou binaries (variables entières ou 0-1)*
  - ✓ *end (fin saisie)*

# Example

maximize

*Objectif:*  $x_1 + 2x_2 + 3x_3$

st

*Temps:*  $-x_1 + x_2 + x_3 \leq 20$

*Travail:*  $x_1 - 3x_2 + x_3 \leq 30$

bounds

$0 \leq x_1 \leq 40$

$x_2 \leq 100$

generals

$x_1$

$x_2$

end

# Résultat de la saisie

➤ On obtient :

✓ Max  $x_1 + 2x_2 + 3x_3$

✓ Sous contraintes

❖ (Temps)  $-x_1 + x_2 + x_3 \leq 20$

❖ (Travail)  $x_1 - 3x_2 + x_3 \leq 30$

❖ (Borne)  $x_1 \leq 40$

❖ (Borne)  $x_2 \leq 100$

❖ (Positivité)  $x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$

❖ (Intégralité)  $x_1$  entier,  $x_2$  entier



# Valeurs par défaut

- Fonction économique : *obj*
- Contraintes : *c1, c2, ..., cm*
- Bornes des variables :
  - ✓ Borne inf. : 0
  - ✓ Borne sup. : aucune
  - ✓ Les bornes des variables déclarées avec le mot clé *binaries* (variables 0-1) ne devraient jamais être modifiées

# *Afficher des informations*

- *Commande display*
  - ✓ *Affiche une liste des options disponibles*
    - ❖ *iis*
    - ❖ *problem*
    - ❖ *sensitivity*
    - ❖ *settings*
    - ❖ *solution*
- *Commande display nom\_option*
  - ex. (valeurs de toutes les variables) :*
  - display sol var -*

# *Sauvegarder ou lire un PL dans un fichier au format LP*

- *Commande write mon\_fichier.lp*
  - ✓ *écrit le PL dans le fichier « mon\_fichier.lp »*
- *Commande write mon\_fichier.bas*
  - ✓ *écrit la solution optimale du PL résolu (après utilisation de la commande optimize) dans le fichier « mon\_fichier.bas »*
- *Commande read mon\_fichier.lp*
  - ✓ *charge le PL contenu dans le fichier « mon\_fichier.lp »*

# *Modification d'un PL*

- *Commande add*
  - ✓ *Entrée de nouvelles contraintes (comme après st)*
  - ✓ *Si elle est suivie du mot-clé bounds*
    - ❖ *Entrée de nouvelles bornes pour une ou plusieurs variables*
  - ✓ *Fin de la saisie avec le mot-clé end*
- *Commande change*
  - ✓ *Liste les modifications possibles à appliquer au PL*
    - ❖ *Changer les bornes d'une variable, un coefficient, un nom*
    - ❖ *Effacer, etc.*

# Installation de CPLEX

- Télécharger le programme à l'adresse
  - ✓ [https://www.ibm.com/developerworks/university/software/get\\_software.html](https://www.ibm.com/developerworks/university/software/get_software.html)
- Installer le logiciel
- Télécharger la clé de CPLEX
  - ✓ <https://www.ibm.com/developerworks/university/support/ilog.html>
- Sauvegarder le fichier dans le répertoire c:\ILOG\ILM\
- Créer la variable d'environnement : ILOG\_LICENSE\_FILE (c:\ilog\ilm\access.ilm)

API Java

# Packages Java

- 2 packages Java sont requis pour appeler CPLEX à partir d'un code Java :
  - ✓ `ilog.concert.*`
  - ✓ `ilog.cplex.*`
- Puis, lors de la compilation, l'emplacement de cplex (archive « `cplex.jar` ») doit être précisé au compilateur (javac) avec l'option suivante :
  - `-classpath emplacement_de_cplex.jar`

# Objet Java nécessaire (1) : le modèle

- Créer un objet (une instance) de type **IloCplex**
  - ✓ implémente l'interface **IloMPModeler**, et donc l'interface **IloModeler**
  - ✓ Par exemple : *IloCplex modele = new IloCplex();*
- Tous les appels au solveur vont se faire via cet objet
  - ✓ définition de la fonction économique et des contraintes du modèle à résoudre, résolution, affichage de la solution,...



# Objet Java nécessaire (2) : les variables

- Variables = objets de type
  - ✓ **IloNumVar** (variables quelconques)
  - ✓ **IloIntVar** (variables entières)
- Chacun de ces objets, pour être intégré au modèle courant (*modele*), doit ensuite être défini ainsi :
  - ❖ `var1 = modele.numVar (borne_inf, borne_sup, type);`  
//écriture générique pour une variable var1
  - ❖ `var2 = modele.intVar (borne_inf, borne_sup);`  
//écriture pour une variable var2 entière
  - ❖ `var3 = modele.boolVar();` //écriture pour une variable 0-1 var3

# Remarques sur les variables CPLEX

- Si une variable réelle n'a pas
  - ✓ De borne inf.    `borne_inf = -Double.MAX_VALUE`
  - ✓ De borne sup.    `borne_sup = Double.MAX_VALUE`
    - ❖ Ecriture similaire si variable entière (*Integer*)
- Possibilité de définir des tableaux de variables via  
*`modele.numVarArray` ou `.intVarArray`  
ou `.boolVarArray`*
- Paramètre *`type`* vaut
  - ✓ `IloNumVarType.Int` (variable entière)
  - ✓ `IloNumVarType.Float` (variable réelle)

# Objet Java nécessaire (3) : les expressions

- Expression = combinaison de variables :
  - ✓ Somme, différence, multiplication par des coefficients
- Une expression est un objet de type **IloNumExpr**
  - ✓ **IloLinearNumExpr** pour une expression linéaire, i.e., pour une combinaison linéaire de variables
  - ✓ Une variable est aussi de type **Ilo(Linear)NumExpr**
  - ✓ Pour définir des PL(NE), on pourra se contenter d'objets de type **IloLinearNumExpr**

# Syntaxe des expressions génériques

- Exemple d'une expression *expr* de type **IloNumExpr**

*expr = modele.sum(x1, modele.prod(2.0, x2));*

- Pour chaque opération (+, -, \*, /, etc.), on utilise une instruction dédiée
  - ✓ Opération « somme » : *modele.sum(...)*
  - ✓ Opération « produit » : *modele.prod(...)*
  - ✓ Opération « différence » : *modele.diff(...)*
  - ✓ Opération « négation » : *modele.negative(...)*
  - ✓ Opération « élever au carré » : *modele.square(...)*

# Syntaxe des expressions linéaires

- 1er exemple d'une expression linéaire `lin`
  - ✓ `IloLinearNumExpr lin = modele.scalProd(vectCoeff, var);`
    - ❖ l'instruction `scalProd` fait le produit scalaire du vecteur de coefficients `vectCoeff` et du vecteur des variables `var`
- 2e possibilité (définition itérative par ajout de termes)
  - ✓ `IloLinearNumExpr lin = modele.linearNumExpr();`
  - ✓ Puis `lin.addTerm(coeffi, vari);` //ajout d'un terme

# Objet Java nécessaire (4) : les contraintes

- Contraintes = objets de type **IloRange**
- Une contrainte *contr* se définit à partir d'une expression *expr*. Ex. :

**IloRange** *contr* = *modele.range(borne\_inf, expr, borne\_sup);*

- On peut aussi utiliser

❖ **IloRange** *le* = *modele.le(expr, borne\_sup);*

*//contrainte en*

❖ **IloRange** *ge* = *modele.ge(expr, borne\_inf);*

*//contrainte en*

❖ **IloRange** *eq* = *modele.eq(expr, 2nd\_membre);*

*//contrainte en =*

# Dernière étape : définir le modèle

- Ajout d'une contrainte `contr` au modèle `modele` :
  - ✓ `modele.add(contr)`
    - ❖ */\* `contr`, de type `IloRange`, a été définie par `modele.range`, `modele.le`, `modele.ge` ou `modele.eq` \*/*
- *Écriture condensée* (définition + ajout au modèle)
  - ✓ `modele.addLe(linExpr, borne_sup);`
    - ❖ *contrainte en*
  - ✓ `modele.addGe(linExpr, borne_inf);`
    - ❖ *contrainte en*
  - ✓ `modele.addEq(linExpr, 2nd_membre);`
    - ❖ *contrainte en =*

# Définition du modèle

- Ajout d'une fonction économique (expression) *lin*
  - ✓ `modele.addMaximize(lin);`
  - ✓ `modele.addMinimize(lin);`
- Principales commandes pour la résolution :
  - ✓ Lancer la résolution : `modele.solve();`



# Résolution du modèle et récupération de la solution

- ✓ Récupérer la valeur optimale
  - ❖ `modele.getObjValue();`
- ✓ Récupérer la solution optimale :
  - ❖ `modele.getValue(var1);`  
valeur de la variable `var1` à l'optimum
  - ❖ `modele.getValues(tableau_variables);`  
valeurs optimales
- ✓ Pour plus d'information sur l'issue de la résolution : `modele.getStatus();`
  - ❖ opt. non borné, problème infaisable, etc.
- ✓ Refermer l'accès au solveur à la fin : `modele.end();`

# Récapitulatif (1)

- Définir le modèle : objet `IloCplex`
  - ✓ `IloCplex modele = new IloCplex();`
- Définir les variables : objets `IloNumVar` et `IloIntVar`
  - ✓ `IloNumVar var1 = modele.numVar(borne_inf, borne_sup, IloNumVarType.Float);`
  - ✓ `IloIntVar var1 = modele.numVar(borne_inf, borne_sup, IloNumVarType.Int);`
  - ✓ `IloIntVar var1 = modele.intVar(borne_inf, borne_sup);`
  - ✓ `IloIntVar var1 = modele.boolVar();`

## Récapitulatif (2)

- Définir une expression linéaire `linExpr`
  - ✓ `IloLinearNumExpr linExpr = modele.linearNumExpr();`
  - ✓ `linExpr.addTerm(coeffi, vari);`
- Ajouter une contrainte (expression `linExpr`)
  - ✓ `modele.addLe(linExpr, borne_sup); // contr.`
  - ✓ `modele.addGe(linExpr, borne_inf); //contr.`
  - ✓ `modele.addEq(linExpr, 2nd_membre);/ /contr. =`
- Ajouter une fonction économique (expression `lin`)
  - ✓ `modele.addMaximize(lin);`
  - ✓ `modele.addMinimize(lin);`

# Récapitulatif (3)

- Résolution : `modele.solve();`
- Récupération de la valeur optimale
  - ✓ `modele.getObjValue();`
- Récupération de la solution optimale
  - ✓ `modele.getValue(var1);`
  - ✓ `modele.getValues(tab_vars);`
- Informations suppl.
  - ✓ `modele.getStatus();`
- Refermer l'accès au solveur : `modele.end();`

# Utilisation d'Eclipse

- Ajouter cplex.jar dans default package->add external jar
- Run configuration
  - ✓ Arguments->-  
Djava.library.path=« c:\....\cplex\bin »