**Master Androide - Module MAOA - Sorbonne Université**

# Recherche Opérationnelle et Optimisation Combinatoire

**Cahier de Travaux Manuels Encadrés (TME)**

**Heuristiques et PLNE compacts ou à nombre exponentiel de contraintes (Branch-and-Cut)**

## Table des matières

# 1 Cadre et mise en place

Ces Travaux pratiques ont pour but d'introduire la mise en œuvre de résolution de problèmes d'optimisation combinatoire en utilisant des programmes linéaires en nombres entiers (Mixed Integer Programs, MIPs). Ils sont proposés en langage C++ utilisant la SDL et se basent sur plusieurs outils externes (Graphviz, Lemon et Cplex).

Les exemples et exercices suivants présentent ainsi :

- Quelques exemples très simples d'heuristiques gloutonnes et de méta-heurstiques
- l'utilisation de PLNEs compacts en utilisant l'API "Concert Technology" de Cplex d'IBM
- la mise en œuvre des algorithmes de Branch-and-Cut en utilisant également l'API "Concert Technology" de Cplex d'IBM.

## 1.1 Cadre système

Le système utilisé pour ces TPs est LINUX sans environnement de développement, mais il est possible de l'adaptater à d'autres systèmes. Les principes généraux présentés ici pourraient tout aussi bien être mis en œuvre en utilisant Gurobi à la place de Cplex. Enfin, Cplex comme Gurobi possèdent des interfaces (partielles) dans de nombreux langages (Java, python, langage dédié etc).
Pour utiliser les fichiers d'exemple du TME :

- pour effectuer ce TME à l'UPMC en salle de la PPTI, les librairies et exécutables nécessaires sont déjà dans le répertoire "/Vrac/MAOA/"
- pour installer ce TME sur votre machine linux : suivre l'annexe pour l'installation logicielle pour placer les librairies et exécutables dans un répertoire (par exemple "/home/vous/logiciels").

Dans les deux cas, il faut ajouter une variable système PATHTUTOMIP contenant le répertoire où sont installées les librairies et exécutables.
Le plus souvent il s'agit de :

- éditer le fichier (caché) de votre racine "∼/.bash_profile " à la PPTI (ou "∼/.bashrc" selon les systèmes)
- ajouter à la fin "export PATHTUTOMIP=/Vrac/MAOA" si à la PPTI (ou "export PATHTUTOMIP=/home/vous/logiciel" si chez vous)
- sauvegarder, fermer les xterm et en ouvrir de nouveaux

## 1.2 Mise en place

**Question 1** *Récupérer l'archive de l'énoncé. Décompactez-là sur votre disque.*

L'archive contient plusieurs répertoires :

- `bin` : où seront stockés les codes compilés des programmes (hors SCIP).
- `Instances` : contenant des instances pour les différents problèmes.
- `Graph` : contenant une classe `C_Graph` de manipulation d'un graphe orienté ou non-orienté et des programmes de visualisation des graphes instances et des graphes solutions de plusieurs problèmes. Les algorithmes de cette classe sont basés sur Lemon et la visualisation sur Graphviz ou sur le format SVG.
- `Heuristic` : contenant des heuristiques (très simples) pour le problème du stable maximum et du voyageur de commerce
- `CompactMIP_CPLEX` : contenant des exemples d'utilisation de Concert Technology pour des PLNEs compacts (ce répertoire contient aussi la solution des exercices)
- `B_and_C_CPLEX` : contenant des exemples d'utilisation de Concert Technology pour des PLNEs à nombre exponentiel d'inégalités (ce répertoire contient aussi la solution des exercices)
- `SCIP` : contenant des exemples d'utilisation de SCIP pour des algorithmes de Branch-and-Price (voir TP correspondant)

Il est possible que vous veniez également de télécharger les solutions aux exercices de ce TP... soyez sage et prenez le temps de réfléchir et de faire par vous-même avant de les regarder...

Les programmes s'exécutent en ligne de commande en passant en paramètre le nom du fichier de l'instance SANS extension.

Première partie

# Heuristique et méta-heuristique

## 2 Prise en main de la classe C_Graph : Heurisitique pour le problème du stable maximum

Dans cette section, on s'intéresse au problème du stable maximum pour des poids égaux à 1 pour chaque sommet : ce problème est NP-difficile. Voir le cours MAOA pour une définition. Le but de cette section est de prendre en main la classe C_Graph qui servira dans les exemples de ce TP.

### 2.1 Regardons un peu la class C_Graph

La structure de données pour ces TP est donnée par la classe `C_Graph` et par les fichiers suivants :
- `Graphe.h` et `Graphe.cpp` : contiennent la classe `C_Graph` implémentant une structure de graphe par liste d'adjacence.
- `viewerXX.cpp` : différents visualisateur d'instances ou de solutions.

**Question 2** *Lancer le Makefile du répertoire* `Graph`. *Il compile plusieurs exécutables dans le répertoire* `bin`. *Allez dans le répertoire* `Instances/DIMACS`. *Visualiser des graphes du répertoire avec* `bin/viewerDIMACS`.

**Question 3** *Regarder le fichier* `Graph/viewerDIMACS.cpp`. *Noter qu'il permet*
- *de passer un nom de fichier en ligne de commande*
- *de créer/lire/afficher un graphe non-orienté de type DIMACS.*

### 2.2 Heuristiques gloutonnes et heuristique de descentes itérées

**Question 4** *Aller dans* `Heuristic` *et lancer le Makefile. Il compile les heuristiques de ce répertoire dans* `bin`.

**Question 5** *Exécuter bin/Heuristic_StableSet pour des instances DIMACS : il crée un fichier contenant la solution sous le codage du vecteur d'incidence du stable obtenu. Utiliser le viewer de solutions du stable pour les visualiser.*

**Question 6** *Regarder le fichier* `Heuristic/Heuristic_StableSet`. *Noter qu'il contient :*
- *une heuristique gloutonne*
- *une heuristique dite des descentes itérées.*

*Comprendre ce code afin de vous familiariser avec la classe* `C_Graph`.

**Deuxième partie**

# Formulation PLNE compacte avec CPLEX

# 3 Prise en main de Concert Technology (Cplex) : le problème du stable maximum

## 3.1 Introduction à Concert Technology

Nous nous intéressons ici à un code C++ utilisant les fonctionnalités Concert Technology de Cplex. Il existe plusieurs moyens d'accéder à Cplex, dans des langages très divers. Nous utilisons ici Concert Technology pour sa facilité à se mêler au langage puissant qu'est le C++.

**Question 7** *Aller dans* `CompactMIP_CPLEX/CompactMIP_StableSet` *et exécuter le Makefile. Un exécutable* `CompactMIP_StableSet` *est compilé dans* `bin`*. Utiliser* `CompactMIP_StableSet` *pour résoudre des instances du répertoire d'instance* `Instances/DIMACS`*. Comparer les résultats obtenus avec l'heuristique.*

**Question 8** *Regarder le fichier* `CompactMIP_StableSet.cpp`*. L'entête rappelle la formulation compacte classique de ce problème (que vous pouvez retrouver dans le cours aussi). Regarder ce code qui se compose de sections assez intuitives à lire :*
*- la gestion de l'instance identique à celle de l'heuristique*
*- la création d'un environnement Cplex sous Concert Technology.*
*- la création des variables du PLNE*
*- la création des inégalités par une surcharge d'opérateur qui permet d'écrire assez facilement les inégalités*
*- la création de la fonction objective*
*- la résolution*
*- la récupération de la solution.*

# 4 Exercice : le problème de coloration de graphes

On considère le problème de coloration de graphe (voir Cours MAOA si besoin).

**Question 9** *A partir de la formulation compacte du problème de coloration vue en cours, créer un programme utilisant Concert Technology pour cette formulation. Votre programme doit écrire sur disque une solution sous la forme d'un vecteur où pour chaque sommet, on indique sa couleur entre 0 et $k-1$ où $k$ est le nombre de couleurs (minimal).*

**Solution**
La solution de cet exercice est dans le répertoire CompactMIP_CPLEX. □

**Question 10** *Tester votre code sur des instances DIMACS. Utilisez le viewer de coloration pour visualiser les solutions.*

---

# 5 Exemple de Formulation compacte "MTZ" : le problème du sous-graphe acyclique induit

## 5.1 Définition et formulation compacte MTZ

Soit un graphe orienté $G = (V, A)$ où $V$ est l'ensemble des sommets et $A$ l'ensemble des arcs. Pour simplifier l'écriture, on note ici $ij$ un arc de $A$ allant du sommet $i$ au sommet $j$. On note $n = |V|$ et $m = |A|$.
Un *chemin* de $G$ est une suite d'arcs $P = (i_0i_1, i_1i_2, ..., i_{k-1}i_k)$. Un chemin de $k$ arcs est dit de longueur $k$. Un *circuit* $C$ dans $G$ est un chemin tel que $i_0 = i_k$. On note $V(P)$ (resp. $V(C)$) l'ensemble des sommets impliqués dans un chemin (resp. un circuit).
Un graphe est dit *acyclique* s'il ne contient aucun circuit.
Etant donné $W \subset V$ un sous-ensemble de sommets, on note $A(W)$ l'ensemble des arcs ayant leurs deux extrémités dans $W$. On dit alors que le graphe $(W, A(W))$ est le graphe induit par $W$.
Le **problème du sous-graphe acyclique induit** (PSAI) consiste à déterminer un ensemble de sommets $W$ tel que le sous-graphe $(W, A(W))$ induit par $W$ dans $G$ soit acyclique et tel que $|W|$ soit maximum. Ce problème a été montré comme étant NP-complet.

Ce problème possède une structure combinatoire que l'on retrouve dans plusieurs problèmes combinatoires dont les solutions sont des sous-graphes ne pouvant pas contenir de circuits. La notion de sous-graphe acyclique orientés (Directed Acyclic Graph), appelée DAG, se retrouve ainsi dans la gestion de projet, la construction de logique d'inférence,...
On considère des variables binaires $x_i$ $\forall i \in V$ indiquant si le sommet $i$ est pris ou non dans la solution.

Pour cette formulation, on ajoute des variables réelles $u_i$ associées à chaque sommet $i \in V$.
Considérons la formulation $(P_C)$ suivante, dite "formulation MTZ" en référence à la contrainte (1) proposée dans un article de Miller-Tucker-Zemlin.

$$Max \sum_{i \in V} x_i$$

$$u_i - u_j + 1 \leq n(2 - x_i - x_j) \qquad \forall (i,j) \in A, \tag{1}$$

$$1 \leq u_i \leq n \qquad \forall i \in V, \tag{2}$$

$$0 \leq x_i \leq 1 \qquad \forall v \in V \tag{3}$$

$$u_i \in I\!R \qquad \forall i \in V, \tag{4}$$

$$x_i \text{ entier} \qquad \forall i \in V. \tag{5}$$

**Question 11** *Montrer que cette formulation est équivalente au PSAI.*

**Solution**

($\Rightarrow$) Considérons tout d'abord une solution (optimale) $(x, u)$ de la formulation. Par les contraintes (8) d'intégrité, les variables $x$ sont binaires. Notons $W = \{i \in W \mid x_i = 1\}$. On peut prouver que le graphe $H = (W, A(W)$ induit par $W$ est acyclique. En effet, supposons que $H$ contienne un circuit $C$. Notons $i_1, i_2, \ldots, i_k$ les indices des sommets $v_{i_1}, \ldots, v_{i_k}$ consécutifs du circuit $C$. On peut noter que pour chaque arc $v_{i_l} v_{i_{l+1}}$ pour $l \in \{1, \ldots k-1\}$, on a $x_{i_l} = x_{i_{l+1}} = 1$ car les sommets de $C$ sont dans $W$. Ainsi, par l'inégalité MTZ (1) : $u_{i_l} + 1 \leq u_{i_{l+1}}$. On a donc $u_{i_1} + k - 1 \leq u_{i_k}$. Or toujours par l'inégalité MTZ, $u_{i_k} + 1 \leq u_{i_1}$, ce qui est impossible. Donc, $H$ est bien acyclique.

($\Leftarrow$) Considérons à présent un ensemble $W$ induisant un sous-graphe acyclique $H = (W, A(W))$. On veut prouver que $W$ correspond à une solution de $(P_C)$. Pour cela, notons $\chi^W$ le vecteur d'incidence de $W$, *i.e.* $\chi^W_i = 1$ si $i \in W$ et 0 sinon. Il est donc nécessaire de déterminer les valeurs des variables $u$ entre 1 et $n$ puis de vérifier si les inégalités MTZ sont alors bien vérifiées par $(\chi^W, u)$.

Pour déterminer les valeurs $u$, considérons l'algorithme suivant :

$F \leftarrow A(W)$
$u_i \leftarrow 0 \ \forall i \in V$
Tant que $F$ est non vide Faire
  Prendre un plus long chemin $\mu$ dans $F$
  $i_0$ premier sommet de $\mu$
  Si $u_{i_0} = 0$ alors $u_{i_0} \leftarrow 1$
  Dans l'ordre des arcs $ij$ de $\mu$, si $u_j = 0$, $u_j \leftarrow u_i + 1$
FinTantque

Cet algorithme est bien valide, car, comme $H$ est acyclique, on peut en déterminer un plus long chemin. On peut noter qu'à la fin de cet algorithme tous les sommets non isolés de $W$ ont une valeur $u$ non nulle. On fixe alors tous les sommets restants à $u_i \leftarrow 1$.

Cette "numérotation" des valeurs $u$ vérifie bien toutes les inégalités MTZ. Cest trivial pour celles correspondant à un arc $ij$ où $x_i + x_j \leq 1$ car l'inégalité devient au pire $u_i - u_j \leq n - 1$

ce qui est toujours vrai. Soit un arc $ij$ avec $x_i + x_j = 2$, c'est-à-dire un arc pris dans $H$. Alors la contrainte MTZ est $u_j \geq u_i + 1$. Cette contrainte est vérifiée par construction pour tous les arcs $ij$ du graphe où $u_j$ était à 0 avant son traitement par l'algorithme. Supposons que $u_j$ est non nul avant le traitement de l'arc $ij$ d'un chemin $\mu$ par l'algorithme et que $u_j < u_i + 1$. Cela veut dire que l'on a numéroté le sommet $j$ en tant que tête d'un arc $i'j$ d'un chemin $\mu'$ traité avant $\mu$. Et $u_i > u_{i'}$ : cela voudrait dire que le sous-chemin de $\mu$ avant $j$ est plus long que le sous-chemin de $\mu'$ avant $j$. Ce qui est impossible.
- Enfin, la fonction objective de $(P_C)$ correspond à la fonction objective de PSAI, la formulation est donc bien équivalente au PSAI. □

Noter que cette formulation est compacte car elle contient $2n$ variables (et seulement $n$ variables binaires) et $m$ inégalités (en dehors des bornes).

### 5.2 Manipulation

Le répertoire `CompactMIP_CPLEX/CompactMIP_Acyclic` contient une implémentation de la formulation MTZ pour le PSAI. Les instances correspondant à ce problème sont dans `Instances/GRA` et possèdent un viewer de graphe et de solution.

**Question 12** *Comme dans les questions précédentes, parcourir, compiler, tester...*

---

## 6 Exercice : le problème du voyageur de commerce

On considère ici le problème du voyageur de commerce euclidien (TSP) : voir cours si besoin.

**Question 13** *Regarder l'heuristique du répertoire* `Heuristic` *pour comprendre comment sont manipulées les instances du TSP. Cette heuristique est composée d'une heuristique gloutonne dite "du plus proche voisin" suivi d'une méthode de descentes stochastiques itérées avec voisinage 2-OPT.*

**Question 14** *Les instances du répertoire* `Instances/TSP` *sont issues de la TSPLIB, benchmark d'instances très utilisées sur le web pour tester des méthodes. Il y a également un viewer correspondant à ces instances : les fichiers produit sont sous format SVG qui sont visualisables par un navigateur web.*

**Question 15** *Coder la formulation MTZ (vue en cours) pour le TSP et la tester sur de petites instances...*

**Solution**
La solution de cet exercice est dans le répertoire CompactMIP_CPLEX. □

**Troisième partie**

# Formulation PLNE à nombre exponentiel d'inégalités avec CPLEX

# 7 Prise en main des algorithmes de Branch&Cut : le problème du sous-graphe acyclique induit

On reprend ici le problème du sous-graphe acyclique pour étudier une formulaion à nombre exponentiel de contraintes.

## 7.1 Formulation non compacte

Considérons la formulation $(P_E)$ suivante :

$$
\begin{aligned}
Max \quad & \sum_{i \in V} x_i & & \\
& \sum_{i \in V(C)} x_i \leq |C| - 1 & \forall C \text{ circuit de } G, & (6) \\
& 0 \leq x_i \leq 1, & \forall i \in V, & (7) \\
& x_i \text{ entier}, & \forall i \in V. & (8)
\end{aligned}
$$

Les inégalités (6) sont dites inégalités de circuit.

**Question 16** *Montrer que cette formulation est équivalente au PSAI.*

**Solution**
Un sous-graphe correspondant à une solution de $(P_E)$ ne peut contenir de circuit et inversement tout vecteur d'incidence d'un ensemble de sommets induisant un sous-graphe acyclique est solution de $(P_E)$. □

Cette formulation $(P_E)$ contient un nombre exponentiel de contraintes par rapport au nombre $n$ de sommets du graphe. Il n'est donc pas possible d'énumérer toutes ces contraintes pour ainsi créer un PLNE à entrer directement dans un solveur.

## 7.2 Méthodes de coupes et Branch-and-Cut

Le déroulement global d'une résolution d'un PLNE par Cplex est un arbre de branchement où dans chaque nœud est résolu la relaxation linéaire du PLNE. Dans le cas d'un PLNE contenant un nombre exponentiel d'inégalités, la résolution de chaque relaxation linéaire de chacun des nœuds est effectuée par une **méthode de coupes** : l'ensemble est alors appelé algorithme de **Branch&Cut**.

L'ajout des inégalités au sein de la méthode de coupes d'un nœud est réalisé par un algorithme de séparation qui, étant donné la valeur d'une solution courante $\tilde{x}$ (a priori non réalisable) détermine si $\tilde{x}$ respecte toutes les inégalités et sinon produit une inégalité violée par $\tilde{x}$.

**Question 17** *Proposer un algorithme de séparation polynomial coupant des solutions entières (non réalisables) pour les inégalités (6) de circuits.*

**Solution**
Comme la variable $x$ est entière, on peut considérer le sous-graphe induit par les sommets $i$ correspondant à une valeur $x_i = 1$. En détectant un circuit dans ce sous-graphe, on détecte une inégalité violée. Il suffit donc de lancer la détection de circuits (au prix $O(n+m)$ d'un parcours en profondeur, dans ce sous-graphe. □

**Question 18** *Proposer un algorithme de séparation polynomial coupant des solutions fractionnaires pour les inégalités (6) de circuits.*

**Solution**
Considérons une solution $x$ associée aux sommets de $G$ : le problème de séparation consiste à déterminer s'il existe une inégalité de circuit violée par $x$ et le cas échéant d'en produire une. Remarquons tout d'abord qu'en posant $\tilde{x} = 1 - x$, la contrainte devient alors $\sum_{i \in V(C)} \tilde{x}_i \geq 1$. Considérons alors un plus petit circuit $\tilde{C}$ selon le poids $\tilde{x}$, on a deux cas :
- soit le cycle $\tilde{C}$ est tel que $\sum_{i \in V(\tilde{C})} \tilde{x}_i < 1$ : on sa alors une inégalité de circuit violée : $\sum_{i \in V(\tilde{C})} x_i \leq |\tilde{C}| - 1$.
- soit un cycle $\tilde{C}$ est tel que $\sum_{i \in V(\tilde{C})} \tilde{x}_i \geq 1$ et dans ce cas, cela signifie qu'il n'y a pas d'inégalité de circuit violée par $x$.

La recherche d'un plus petit circuit se fait facilement :
On pose $w_{ij} = \frac{\tilde{x}_i + \tilde{x_j}}{2} = \frac{2 - x_i - x_j}{2}$ pour tout arc $ij \in A$. On recherche alors, pour tout $i \in V$, un plus court chemin selon $w$ entre $i$ et chaque sommet $j$ prédécesseur de $i$. (Pour cela, on peut rechercher l'arborescence des plus courts chemins depuis $i$ puis exploiter cette arborescence pour tout prédécesseur de $i$). Le plus petit des circuits obtenus correspond au plus petit circuit passant par $i$. Le plus petit de tous donnent le plus petit circuit. □

## 7.3 Les "Callbacks" de Concert Technology

Concert Technology permet d'interrompre le déroulement de Cplex pour nous permettre d'exécuter un algorithme de séparation et d'ajouter les inégalités dans le PL d'un nœud de l'arbre : ces interruptions sont réalisées dans des "Callbacks".

Un **Callback** est une fonction de Concert Technology permettant différentes intéractions : l'ajout d'inégalités comme nous venons de le dire, mais aussi diverses autres interventions comme produire des solutions réalisables par arrondi ou vérifier efficacement qu'une solution est bien valide.

Dans le cas des Callbacks d'ajout d'inégalités, il y a deux sortes de Callbacks suivant la façon dont les inégalités sont utiles à la formulation :

- `ILOLAZYCONSTRAINTCALLBACK` qui correspond aux séparations **exactes** dites "lazy" qui correspondent à des inégalités valides telles que la formulation PLNE serait incomplète sans elles (donc des inégalités nécessaires à la formulation PLNE). Ainsi Cplex utilise cette fonction à chaque fois qu'une solution entière a été produite. Si une variable est demandée à être entière dans la formulation, le "Lazy Callback" n'est appelée **que pour couper une solution entière** et cette séparation doit être exacte.
- `ILOUSERCUTCALLBACK` qui correspond à des séparation **exactes ou heuristiques** d'inégalités dites "user cuts" qui sont des inégalités qui peuvent ne pas être essentielles : elles sont utilisées en **renforcement**. Cplex utilise à sa guise ces séparations de séparation dès qu'il le juge utile (on peut augmenter sa fréquence d'utilisation en changeant des paramètres).

En fait, une séparation "lazy" sera utilisée pour couper **toute** solution entière : c'est de là que vient leur nom "lazy" car il y a bien plus de solutions fractionnaires à couper que de points entiers. Des solutions entiers apparaissent :
- quand on est en train d'explorer un nœud suffisamment profond dans l'arbre
- quand CPLEX tente de savoir si une solution heuristique produite par ses heuristiques primales est valides ou non pour la formulation
- et au hasard des relaxations linéaires qui fournissent des points entiers.
En pratique, les séparations "lazy" sont très souvent utilisées !

Les séparations "user cut" ne sont pas utilisées pour prouver qu'un point est solution (de toute façon, elles ne sont pas utilisées pour couper un point entier). Il s'agit donc d'utiliser cette séparation en **renforcement** :
- pour des inégalités nouvelles capables de renforcer la valeur de relaxation
- pour séparer les mêmes inégalités qu'en "lazy" mais ici avec un rôle non obligatoires de renforcement.
D'autre part même si Cplex est libre d'utiliser ou non les user cuts, elles sont très souvent utilisées. Mais les deux appels ont rarement lieu lors de la même itération.

Notez-bien qu'une même inégalité essentielle à une formulation peut être séparée deux fois :
- une fois par un algorithme de séparation exacte coupant des points entiers et mis en "lazy contraints"
- une fois par un algorithme de séparation (exacte ou non) coupant des points fractionnaires et mis en "user cuts".

On peut noter que, pour une même classe d'inégalités, on a souvent trois algorithmes de séparation :
A : un algorithme exact pour les points entiers
B : un algorithe exact pour les points fractionnaires
C : une heuristique rapide pour les points fractionnaires
Dans ce cas, on utilise :
- si les inégalités sont nécessaires à la formulation, le A en lazy ; puis le C en début de "user cut" ; si le C ne trouve pas d'inégalités, on lance alors le B en dernier recours.
- si les inégalités ne sont pas nécessaires à la formulation : on lance d'abord le C (ou le A s'il est rapide) ; puis le B en dernier recours. L'idée est que, si l'exact est vraiment lent, on peut préférer toujours exécuter l'heuristique avant et ne lancer l'exacte que si l'heuristique n'a pas déterminé d'inégalités violées.

### 7.4 Callback et fonctions de séparation pour le PSAI

Le répertoire `BC_Acyclic` contient une implémentation simple d'un algorithme de coupes et branchement pour la formulation non-compacte du PSAI.

**Question 19** *Regarder le fichier* `le fichier CircuitSeparation.cpp`.

Il contient deux fonctions correspondant à deux algorithmes de séparation :
- `find_ViolatedAcyclicCst_INTEGER` pour le cas où $x$ est entier
- `find_ViolatedAcyclicCst` pour le cas général.
Notez que la seconde fonction, plutôt que rechercher une unique inégalité violée, elle peut retourner plusieurs inégalités violées si elle en trouve (jusqu'à un nombre maximum de 200).

**Question 20** *A quoi sert le tableau "lotohat" ?*

**Solution**
Le tableau lotohat permet de ne pas explorer toujours le graphe dans l'ordre des numéros des sommets afin de mieux couvrir tout le graphe. □

**Question 21** *Regarder le fichier* `BC_Acyclic.cpp`.

Il contient un main similaire à ceux rencontrés dans le cas compact : les inégalités en nombre exponentiel sont activées par l'utilisation de `cplex.use(LazyCircuitSeparation(env,G,x));` qui ajoute un Callback au PLNE à résoudre. Le Callback "LazyCircuitSeparation" est un nom choisi par l'utilisateur au travers d'une macro appelée en haut du fichier (on ne peut pas mieux ranger... pour protester, écrire à IBM).

```
ILOLAZYCONSTRAINTCALLBACK2(LazyCircuitSeparation,
   C_Graph &, G,
```

```
   vector<IloNumVar>&,x
           )
{...}
```

Le nombre 2 siginifie que la macro va construire une fonction de deux paramètres en plus du paramètre habituel "env" : ici nous avons mis le graphe et le vecteur qui nous a permis de stocker les variables de Cplex. On peut ajouter des paramètres si besoin (dans la limite de 7) : le mieux est d'encapsuler les données dans une instance (ici $G$) et les variables dans un nombre limité de tableaux (ici un seul $x$). Le premier champs de la macro est le nom du Callback que l'on désire créer. Le code de cette fonction est très court : il récupére la solution courante, lance la fonction `find_ViolatedAcyclicCst_INTEGER` qui fournit éventuellement des inégalités violées (dans le cas où $x$ est entier), puis les ajoute au PL géré par Cplex.
La même fonction de séparation a été également utilisée pour créer un Callback "user cut" : en effet cette séparation est très rapide et essentielle : les deux callbacks lazy et user cut sont donc ici identiques sauf que "user cut" utilise la fonction `find_ViolatedAcyclicCst`.

### 7.5 Checker

La fonction "lazy" pourrait théoriquement être utilisée par Cplex pour tester si une variable $x$ entière correspond à une solution.
Mais ici, on peut faire autrement : on a ajouté un 3ème Callbacks dit `ILOINCUMBENTCALLBACK` du nom de `CheckSolFeas` : ce Callback est lui aussi ajouté à Cplex par la commande
`cplex.use(CheckSolFeas(env,G,x));`
dans le main. Une solution générée par le PL (lors de la résolution du PL) peut être entière, dans ce cas, elle est dite "incumbent" c'est-à-dire solution potentielle. Pour le PSAI : en effet, il suffit de tester si le graphe est sans circuit : ce qui peut se faire par un simple parcours en profondeur (qui est implémenté dans la classe `C_Graph`). Ainsi Cplex utilisera cette fonction dès qu'une solution entière est produite : c'est le cas si la solution est obtenue lors d'une relaxation, c'est aussi le cas lorsqu'elle est produite par une heuristique primale interne à Cplex.

### 7.6 Heuristique primale (matheuristique)

Cplex possède des heuristique primales "génériques", c'est-à-dire dans ce cas, des fonctions automatiques d'arrondi d'une solution $x$ fractionnaires. Il teste par l'utilisation du checker si cette solution est ou non solution. Si elle est de valeur intéressante, il l'a conserve.
Dans le déroulement à l'écran du programme, on peut constater qu'à chaque foi que Cplex a réussi à améliorer la solution courante par une heuristique primale, il ajoute une étoile * en début de ligne.

Il est possible d'ajouter soit-même une heuristique primale ad-hoc pour un problème.
Le code de contient une heuristique primale pour le problème du sous-graphe acyclique. Elle

consiste à trier les sommets par valeurs $x$ décroissante et de les ajouter un à un dans cet ordre tant qu'il ne forme pas un circuit. (Remarque : le code fourni est de très mauvaise complexité...).

L'idée d'une telle heurisitique est de fournir très rapidement de nombreuses soltions heuristiques : en effet, elle peut être exécutée à chaque itération de l'algorithme de coupes de chacun des nœuds de l'arbre ! Parmi toutes ces solutions, il y a très souvent la solution optimale... mais il faut dérouler l'arbre jusqu'à tester si elle est ou non optimale par encadrement borne min/borne max.

### 7.7 Tests experimentaux

**Question 22** *Compiler le programme et le tester en comparaison avec la méthode compacte vu auparavant.*
*: Tester le code en activant tour à tous les différentes fonctions*
*: - ne laisser que le checker (Branch-and-Bound pur)*
*- puis les inégalités Lazy (Branch-and-Cut sans renforcement)*
*- puis les inégalités User Cut*
*- puis l'heuristique primale.*
*Que constatez-vous ?*

---

## 8 Exercice : le problème du voyageur de commerce

**Question 23** *Coder la formulation du TSP utilisant les inégalités de coupes (dites de Menger).*

**Solution**
La solution de cet exercice est dans le répertoire B_and_C_CPLEX. □

Pour cela, il faut implémenter un algorithme de séparation des inégalités de coupes, dites aussi "coupes de Menger" car elles sont justifiées par le théorème de Menger. Vous pouvez utiliser dans la classe `C_Graphe` les méthodes permettant de rechercher une coupe de valeur minimale dans un graphe.

**Question 24** *Tester la formulation en comparaison avec la méthode compacte vu auparavant.*

## Quatrième partie

# Annexes

## 9 Annexe : Installation des logiciels sous linux

Nous proposons ici une installation de trois logiciels dans le répertoire "/home/username/logiciels" où username est votre login : GraphViz utilise 9,7MO ; Lemon 4,1MO ; CPLEX 611MO (sans compilation des exemples et en supprimant OPL) ; et si vous désirez étudier la partie Branch-and-Price SCIP 426MO.

### 9.1 Intallation de Graphviz

Graphviz est un outil opensource de visualisation de graphes sous licence EPL (libre pour utilisation commerciale).

Graphviz est ici utilisé à l'intérieur d'une classe Graph pour l'affichage de graph en utilisant en ligne de commande différents exécutables (principalement dot, neato). Vous n'avez pas besoin d'apprendre à utiliser Graphviz pour ces travaux pratiques (si vous désirez apprendre à manipuler cet outil très simple voir la documentation `http://www-desir.lip6.fr/~fouilhoux/documentens.php`)
Vous pouvez tester si le logiciel est installé en testant si neato est bien présent en ligne de commande. Si ce n'est pas le cas et si vous êtes superutilisateur, vous pouvez l'installer très facilement sous linux à partir des systèmes automatiques d'installation de packages. Sinon il est téléchargeables sous `http://www.graphviz.org/` pour différents systèmes. Sous linux, il suffit de télécharger le "source code" de la dernière version stable dans un répertoire temporaire puis taper :
- cd /home/username/logiciels
- mkdir graphivzxxx
- cd "votre repertoire temporaire"
- tar -zxvf graphizxxx.tgz
- cd graphizxxx.tgz
- ./configure --prefix=/home/username/logiciels/graphvizxxx
- make
- make install

Remarque : Il semble qu'un nombre important de warning peut apparaître sans provoquer d'erreurs...

### 9.2 Installation de Lemon

Lemon est une librairie C++ opensource insérée dans le projet COIN-OR en licence libre non-commerciale. Elle se compose de structures manipulant des graphes et propose des implé-

mentations efficaces de plusieurs algorithmes de graphes (plus-courts-chemins, coupe-min,...). Lemon est ici utilisé à l'intérieur d'une classe Graph de manière "encapsulée", c'est-à-dire que l'utilisateur de la classe Graph n'a pas besoin de connaître Lemon.
L'installation de Lemon demande de télécharger le fichier .tgz dans un répertoire temporaire puis, dans ce répertoire, taper :
- cd /home/username/logiciels
- mkdir lemonxxx - cd "votre repertoire temporaire"
- tar -zxvf lemonxxx.tgz
- cd lemonxxx.tgz
- mkdir build
- cd build
- cmake -DCMAKE_INSTALL_PREFIX=/home/logiciels/lemonxxx ..
- make
- make install

### 9.3 Installation de Cplex

• Récupérer votre licence gratuite de Cplex

Pour cela, il faut préalablement avoir créer un compte étudiant sur
`https://ibm.onthehub.com/WebStore/Security/Signin.aspx`
Les membres académiques (chercheurs, enseignants, étudiants) peuvent obtenir une version de Cplex en montrant la preuve de leur appartenance et en s'engageant à ne pas utiliser le logiciel dans un cadre d'entreprise.

Si vous n'avez pas de compte, cliquez sur Register et choisissez l'option permettant de fournir une adresse reconnue par IBM (par exemple xxx.xxx@upmc.fr). Si vous n'avez pas une telle adresse, vous pouvez aussi choisir l'option `[By submitting proof of your academic affiliation (e.g. a student ID or report card) via upload or fax.]` : dans ce cas munissez d'un scan de votre carte d'étudiant et répondez aux questions.

Une fois obtenu votre compte, vous pouvez taper CPLEX dans la barre de recherche : on vous propose alors "CPLEX Optimization Studio", ajoutez-le à votre panier.
Cliquez ensuite sur votre panier en haut à droite puis télécharger la version qui vous intéresse (en général linux 64, windows ou OSX). (Il est à noter que les linux 32 n'ont plus de version depuis 2017 : pour connaître votre version linux, tapez `uname -a` en ligne de commande.)

• Installation sous Linux

Attention : il faut faire attention à la place disponible sur votre disque. Tapez régulièrement "df -h" ou "quota -h" pour vérifier si vous avez l'espace disponible. Il est signalé ici comment

éviter d'avoir trop de place prise par cplex.

Procédure :
- placer le fichier cplex_studioxxx.linux-x86-64.bin dans "/home/username/logiciels"
- tapez "chmod +x cplex_studioxxx.linux-x86-64.bin" pour rendre le fichier exécutable.
- tapez "./cplex_studioxxx.linux-x86-64.bin"
- suivez les instructions pour choisir la langue et accepter la license
- taper "/home/username/cplex-xxxxx" pour le chemin d'installation
ATTENTION : cette installation va temporairement utiliser 1,6GO sur votre disque.
- une fois installé, si vous désirez réduire l'espace utilisé par Cplex, vous pouvez supprimer le répertoire "opl" ainsi que l'archive.

• Warning indésirable

Il peut arriver par la suite que lors de la compilation des exemples de ces TPs, votre compilateur C++ produise des Warning concernant le code interne de Cplex ! Il est alors très utile de supprimer ces warnings : voir quelques pistes pour cela en Annexe.

### 9.4 Installation de Scip

TBF [1]

## 10 Annexe : Trop de Warnings ?

Les Warnings sont utiles au programmeur... mais quand une série de Warnings apparaît dans des lignes de codes de librairies incluses, cela gène la lecture des messages de compilation. Voici quelques astuces à utiliser avec parcimonie.

### 10.1 Supprimer des Warnings en général

Il peut arriver que de nombreux Warnings apparaîssent dans des lignes de codes des librairies Lemon, Cplex ou Scip utilisez dans ce TP. En général, le nom du Warning est préciser en crochet à la fin de l'affichage : par exemple "[-Wredundant-decls]" ou "[-Wcast-qual]". Il s'agit de Warning activés par défaut dans votre compilateur. Pour cela, ajouter dans votre Makefile les options "-Wno-"suivi du nom du Warning dans les lignes de compilations : par exemple "-Wno-redundant-decls" ou "-Wno-cast-qual".

### 10.2 Supprimer des Warnings dans Cplex

Si vous voyez apparaître ce warning

```
In file included from /home/fouilhoux/logiciels/cplex-12.6.3/cplex/include/ilcplex/ilocplexi.h:982:0,
                 from /home/fouilhoux/logiciels/cplex-12.6.3/cplex/include/ilcplex/ilocplex.h:29,
                 from src/SCIP_pricer_coloring.h:6,
```

1. Très Bientôt Fait

```
                from src/SCIP_pricer_coloring.cpp:1:
/home/fouilhoux/logiciels/cplex-12.6.3/cplex/include/ilcplex/iloparam.h: At global scope:
/home/fouilhoux/logiciels/cplex-12.6.3/cplex/include/ilcplex/iloparam.h:13:8: warning: IloCplex::Param::SolutionTarget is deprecated [-Wdeprecated-declarations]
 struct Param {
```

Editer /home/username/logiciels/cplex-xxxx/cplex/include/ilcplex/iloparam.h et mettre en commentaire la ligne "//CPXDEPRECATED(12060200)"