Sup Galilée - INFO3 - Université Sorbonne Paris Nord
Recherche Opérationnelle
et Applications (ROA)

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

VERSION Julia+JuMP+CPLEX

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 Julia utilisant le package JuMP qui sert d'interface au logiciel CPLEX. D'autres package Julia seront utilisés pour manipuler des courbes (Plots) ou des graphes.

Les exemples et exercices suivants présentent ainsi :

- Quelques exemples très simples d'heuristiques gloutonnes et de méta-heuristiques
- l'utilisation de PLNEs compacts en utilisant Julia et JuMP
- la mise en œuvre des algorithmes de Branch-and-Cut en utilisant également Julia+JuMP avec derrière le solveur CPLEX

Question 1 *Récupérer l'archive du TME qui se dézipe en créant un répertoire* `TME_MAOA` *contenant des fichiers .jl (langage Julia) et quelques autres fichiers.*
Le sous-répertoire Instances propose des instances pour les différents exemples et exercices.
Les répertoires cités dans le TP seront à partir de la racine du répertoire `TME_MAOA`.

Table des matières

Première partie

Découverte de Julia et de JuMP

1 Cadre et mise en place

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 (C, C++, Java, python, langage dédié etc).

1.1 Le langage Julia

Julia est un langage de programmation conçu au MIT [1]. L'objectif de ce langage est d'être à la fois un langage performant et de haut niveau (d'une facilité de programmation similaire à Python), dans l'objectif d'être utilisé dans des projets scientifiques. Ainsi Julia comporte des bibliothèques rapides (par exemple pour l'algèbre codés en C et en Fortran) ainsi que de nombreux packages, tous en open source, permettant d'en étendre l'utilisation. A terme, les créateurs de Julia ambitionne qu'il remplace Python, mais aussi les langages/logiciels scientifiques comme Matlab, Scilab ou R.
Julia repose sur le principe de "compilation à la volée", en anglais Just-In-Time (JIT) compilation. C'est-à-dire que les lignes de code sont compilées en langage machine par le programme Julia avant d'être exécutée (comme le C) mais en le faisant nouvelle ligne par nouvelle ligne, ce qui permet une utilisation proche de Python mais bien plus rapide que ce dernier.

1.2 Lancer Julia

Similairement aux machines virtuelles (comme Java), Julia est un logiciel qui va compiler et exécuter votre programme écrit en Julia. Ainsi le lancement de Julia et la mise en place des packages requis est un peu longue. Une fois lancé, le logiciel Julia permet par contre de lancer plusieurs programmes dans une configuration chargée auparavant. De plus des fonctions déjà compilées sont déjà connues et ne sont plus à re-compiler...
Attention à l'ordre de compilation : si vous utilisez plusieurs fichiers, il faut faire attention à ce que tous les fichiers soient compilés à partir de leur dernières sauvegardes

1.2.1 Sur les machines de l'institut Galilée

Julia est déjà installé sur les machines de l'institut Galilée sur le compte pierre.fouilhoux.
Il vous suffit de taper

- d'ajouter à la fin de votre .bashrc :

1. The Massachusetts Institute of Technology (MIT) est une des lieux universitaires les moteurs dans l'innovation technonologique

```
export PATH=$PATH:/export/home/users/Enseignants/pierre.fouilhoux/julia-1.6.2/bin/
export JULIA_DEPOT_PATH="/export/home/users/Enseignants/pierre.fouilhoux/.julia-1.6.2"
```

- de relancer votre terminal
- puis d'utiliser `julia-1.6.2` **ET NON** `julia` **tout court !**

Il se peut qu'un long message d'erreur vous parle de "log file" au départ, ce n'est pas grave : il se termine par une phrase vous signalant que ce genre de messages est désactivé pour la suite.
Si vous voulez en savoir plus ou installer Julia chez vous, allez voir l'annexe 6.6.
L'annexe 6.7 propose aussi des idées pour régler quelques bug.

1.3 Un premier programme en Julia

Téléchargez sur le site du TP, le fichier `exemple_1er_pgm.jl` qui propose un premier exemple avec une fonction et une boucle for. Les lignes qui ne sont pas dans une fonction seront exécutées et sont donc une sorte de programme principal.

1.4 Prise en main du logiciel Julia

Il existe trois modes d'utilisation du logiciel Julia

- **en ligne de commande Linux :** `julia exemple_1er_pgm.jl`.
  Ici le programme Julia se lance, compile le programme `exemple_1er_pgm.jl`, l'éxécute, puis Julia s'arrête. Comme le lancement de Julia peut être lent s'il y a des ajouts de package. Ce mode n'est pas à utiliser pour les gros projets.
- **en ligne de commande Julia :** en lançant tout d'abord `julia`. Puis en tapant `include(‘‘exemple_1er_pgm.jl’’)`. Là, Julia a été lancé une fois pour toute, le programme est compilé et exécuté. Julia reste alors actif... ainsi que toutes les fonctions que vous venez de compiler dans `exemple_1er_pgm.jl` (pour vous en convaincre, tapez par exemple : `Affiche_n_fois_une_chaine(5,"Salut")` qui va être exécutée dans la console Julia.
- **Jupyter notebook** : (pour ceux qui sont intéressés, voire l'annexe).

Que choisir en ces trois modes : à vous de voir, mais le choix **en ligne de commande Julia** sera celui utilisé dans la suite du document... Il est très pratique. En particuler, en lançant `julia` votre terminal devient le terminal du logiciel Julia. Les commandes sont donc celle du langage Julia : ce qui permet de lancer des programmes ou des lignes à la volée.

1.5 Un deuxième programme : dessiner des courbes

Le deuxième exmple `exemple_dessiner_courbe.jl` propose 2 fonctions

- la première permet de lire un fichier de données texte et de le charger dans deux tableaux
- la deuxième permet de lancer la première fonction et de créer une courbe dans un fichier pdf

La première ligne de `exemple_dessiner_courbe.jl` est `using Plots` : ce qui permet de signaler que l'on va utiliser le package Plots pour dessiner des courbes...

Si vous êtes sur votre machine : ce package doit être chargé auparavant : lancez les lignes permettant de charger Plots `import Pkg; Pkg.add("Plots")`

Vous pouvez exécuter
`using Plots` Vous pouvez noter que cela prend un certain temps au premier lancement. Il est donc important **en ligne de commande Julia** de le taper au début (ou de lancer une première fois un programme comprenant cette ligne), il sera alors chargé une bonne fois pour toute votre session.

Question 2 *A présent, la commande* `include("exemple_dessiner_courbe.jl")` *arrive à la fin de la compilation... Mais rien ne se passe car ce programme ne contient aucune ligne composant un programme principal.*
Pour lancer le programme, il faut lancer la fonction `Dessiner_courbe` *avec le fichier de votre choix : cela peut se faire donc en* **ligne de commande Julia** *en tapant* `Dessiner_courbe("donnees.txt")` *par exemple.*
Pour cela, le fichier "donnees.txt" est à créer avec un éditeur texte : il doit contenir deux colonnes de chiffres : la 1ere donne les coordonnées x et la deuxième les coordonnées y d'une courbe (les points seront reliés par des droites).

1.6 Mesure du temps

L'exemple suivant pris sur `https://juliapackages.com/p/cputime` permet de comprendre comment mesurer le temps en Julia.
Si vous êtes sur votre machine, il faut ajouter `Pkg.add("CPUTime");`

```
using CPUTime

function Fonction_a_chronométrer()
    x = 0
    for i in 1:10_000_000
        x += i
    end
    sleep(1)
end
```

La fonction exemple `Fonction_a_chronométrer()` contient un temps effectif et un temps de repos forcé avec la commande `sleep`.
La commande `@time @CPUtime Fonction_a_chronométrer()` renvoie alors

```
elapsed CPU time: 0.000174 seconds
  1.002640 seconds (32 allocations: 912 bytes)
```

La macro `@CPUtime` affiche en premier le temps effectif d'utilisation du CPU par la fonction.
La macro `@time` affiche ensuite le temps réel observé de la fonction.
L'un est plus grand que l'autre : il vaut donc mieux utiliser le temps CPU car le temps réel dépend de la charge d'utilisation de votre ordinateur.
En revanche, le temps CPU est difficile à établir sur une machine multi-cœur... car il s'agit souvent d'une fonction du fabriquant de processeur qui cherche à donner le meilleur temps possible... ce qui n'est que rarement le cumul du temps des différents cœur mais plutôt une valeur bien inférieure (une sorte de moyenne, un max ?).

1.7 Pour aller plus loin en Julia

Il y a de nombreux documents synthétiques pour tout comprendre en Julia. En voici certains très utiles :
Tout en une page et en français :
`https://juliadocs.github.io/Julia-Cheat-Sheet/fr`
Un peu plus complet avec un menu pratique :
`https://syl1.gitbook.io/julia-language-a-concise-tutorial`
Et bien sûr la doc officielle
`https://docs.julialang.org/en/v1`

2 Graphes, heuristiques et méta-heuristiqes

Mais si vous êtes sur votre machine : pour obtenir l'ensemble des packages utilisant des graphes : `import Pkg; Pkg.add("Graphs"); Pkg.add("SimpleWeightedGraphs"); Pkg.add("Cairo"); Pkg.add("Compose"); Pkg.add("Fontconfig"); Pkg.add("GraphPlot"); Pkg.add("Colors");`.

Question 3 *Regardez et charger les fichiers `Graphe_Manip.jl` et `Graph_StableSet.jl`.*

2.1 Regardons un peu `Graphe_Manip.jl`

Vous pouvez trouver des fichiers au format DIMACS dans le répertoire `Instances/DIMACS` (ces fichiers sont issus de la librairie d'instances du problème de coloration de graphe).

Question 4 *Lancer la commande de lecture d'un graphe de format DIMACS avec la fonction de lecture.*
Visualiser ces graphes avec la function créant un pdf représentant le graphe.

2.2 Problème du stable : Heuristiques gloutonnes et heuristique de descentes itérées

Dans cette section, on s'intéresse au problème du stable de cardinalité maximale (pour des poids égaux à 1 pour chaque sommet) : ce problème est NP-difficile. Voir le cours ROA pour une définition. Le but de cette section est de prendre en main à la fois les graphes en Julia et aussi les heuristiques/métaheuristiques

Question 5 *Regardons `Graphe_StableSet.jl`. Il contient une heuristique gloutonne et une métatheuristique. Lancer la fonction lançant les deux méthodes l'une derrière l'autre sur quelques instances.*
Un fichier pdf donne le stable obtenu.

Question 6 *Lire attentivement le code pour comprendre ce qu'il fait. N'hésitez pas à écrire l'algorithme à la main etc.*

Deuxième partie

PLNE compact avec le package JuMP

Le langage Julia a donné lieu à de nombreux projets open-source créé un peu partout dans le monde.
Nous allons utiliser le package **JuMP** `https://jump.dev` qui permet

- de modéliser des programmes linéaires (entre autres)
- d'interfacer facilement avec des solveurs linéaires existants comme GLPL, CPLEX, Gurobi, CBC,...

Ainsi Julia et JuMP sont dans ce document uniquement des interfaces pour le solveur CPLEX !
Si vous êtes à la PPTI
Normalement tout est installé.
Si vous voulez installer Cplex sur votre machine allez voir l'annexe 7.

2.3 Un premier exemple de PL en JuMP

Le programme `exemple_PL_cplex.jl` vous propose un premier programme pour manipuler un programme linéaire avec Cplex.

Lisez et exécutez le programme pour comprendre le fonctionnement général.
Quelques remarques :

- la commande `Model()` de JuMP renvoie une variable créant un PL dans le solveur choisi : JuMP est donc une interface avec ce solveur
- plusieurs exécutions de `Model()` renveront autant de modèles : on peut gérer côte à côte plusieurs PL
- les commande commençant par un @ sont des macro-commande Julia conçu pour JuMP, elles lancent une série de fonctions permettant de créer le modèle
- la commande `optimize!()` est une commande JuMP mais elle va en fait lancer l'algorithme d'optimisation du solveur que vous avez choisi plus haut (GLPK ou CPLEX) : ce n'est pas JuMP qui résoud mais bien le solveur. Ainsi les performances vont dépendre du solveur choisi

3 PLNE compact pour le problème du stable

Question 7 *Regardez le modèle PLNE compact dit "aux arêtes" pour le problème du stable de cardinalité maximale dans "`PLNE_compact_StableSet.jl`"*

Question 8 *Résoudre des instances du répertoire d'instance* `Instances/DIMACS`. *Comparer les résultats obtenus avec l'heuristique.*

4 Formulation compacte pour le problème du sous-graphe acyclique induit

4.1 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.

$$
\begin{aligned}
& Max \sum_{i \in V} x_i & & \\
u_i - u_j + 1 \leq n(2 - x_i - x_j) \quad & & \forall (i,j) \in A, & \quad (1) \\
1 \leq u_i \leq n \quad & & \forall i \in V, & \quad (2) \\
0 \leq x_i \leq 1 \quad & & \forall v \in V & \quad (3) \\
u_i \in I\!R \quad & & \forall i \in V, & \quad (4) \\
x_i \text{ entier} \quad & & \forall i \in V. & \quad (5)
\end{aligned}
$$

Question 9 *Montrons 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 χ^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 (χ^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 μ dans F
  i_0 premier sommet de μ
  Si $u_{i_0} = 0$ alors $u_{i_0} \leftarrow 1$
  Dans l'ordre des arcs ij de μ, 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 μ 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 μ' traité avant μ. Et $u_i > u_{i'}$: cela voudrait dire que le sous-chemin de μ avant j est plus long que le sous-chemin de μ' 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).

Question 10 *- il y a une fonction de chargement des instances de graphes orientés du répertoire "Instances/GRA" dans ("Graph_Manip.jl")*
- il y a un visualisateur de solution dans "Graph_Acyclic.jl"
- il y a la formulation dans "PLNE_compact_Acyclic.jl" - Testez-le sur les instances et vous pouvez visualisez les instances (fichiers pdf créés)

Troisième partie

PLNE à nombre exponentiel d'inégalités avec JuMP et CPLEX

5 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.

5.1 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 11 *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.

5.2 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 12 *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 13 *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. □

5.3 Les “Callbacks” de CPLEX en JuMP

CPLEX peut interrompre son déroulement 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 CPLEX 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 :

- `LAZY CONSTRAINT CALLBACK` 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.
- `USER CUT CALLBACK` 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.

5.4 Callback et fonctions de séparation pour le PSAI

Les fichiers `BandC_Acyclic.jl` et `BandC_AcyclicSeperationAlgo.jl` contiennent une implémentation simple d'un algorithme de coupes et branchement pour la formulation non-compacte du PSAI.

Question 14 *Regarder le fichier* `BandC_AcyclicSeperationAlgo`.

Il contient deux fonctions correspondant à deux algorithmes de séparation :
- `ViolatedAcyclic_IntegerSeparation` pour le cas où x est entier
- `ViolatedAcyclic_FractionalSeparation(G,xsep)` 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 15 *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 16 *Regarder le fichier* `BandC_Acyclic.jl`.

Il contient un main similaire à ceux rencontrés dans le cas compact : les inégalités en nombre exponentiel sont activées par l'utilisation de `MOI.set(LP, MOI.XXCallback(), MaFonctionPourXXX)`

qui ajoute un callback au PLNE à résoudre. Le Callback "maFonctionPourXXX" est une fonction codée par nous (utilisateur) que l'on doit écrire avant le `MOI.set` dans la fonction du code principal [2]. Le paramètre "LP" repère le PLNE comme pour les PLNE compacts

Dans la fonction MaFonctionXXX, on reçoit en paramètre d'entrée une variable `c_data` qui contient la valeur de la solution à traiter. Noter que le code de cette fonction est très court : il récupére la solution courante, lance l'algorithme de séparation correspondant qui fournit éventuellement des inégalités violées, puis les ajoute au PL géré par Cplex : pour cela on utilise en fin de fonctions `MOI.submit(LP, MOI.UserCut(cb_data), cst)` ou `MOI.submit(LP, MOI.UserCut(cb_data), cst)` où `cst` contient la nouvelle contrainte qui vient d'être séparée. Ici il y a bien deux callbacks "lazy constraint" et "user cut" qui ont à peu près la même forme mais qui n'utilisent pas les mêmes algorithmes de séparation (on pourrait toutefois mettre le même s'il est efficace dans les deux cas).

5.5 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.
Attention CPLEX teste par l'utilisation du callback lazy si cette solution est ou non solution. Si elle est de valeur intéressante, il la 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.
Vous verrez ainsi utiliser votre fonction "lazy contrainte" utilisée en dehors du cadre des itérations de l'algorithme de coupes.

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.

2. On doit pouvoir faire mieux mais cela permet un programme "main" assez lisible.

5.6 Paramètres CPLEX

Le solveur CPLEX possède plusieurs paramètres venant modifier son déroulement.
En JuMP, on peut les activer par la commande suivante placée par exemple juste après la définition du model

```
model = Model(CPLEX.Optimizer)
set_optimizer_attribute(model, "CPXPARAM_Tune_TimeLimit", 600)
```

Ce paramètre permet de fixer le temps total pour le solveur à 10 min.

Quelques paramètres utiles :

- `‘‘CPXPARAM_Tune_TimeLimit’’, x)` : limite le temps total d'exécution du solveur à `x` secondes. Si le solveur a pu calculer sur cette durée une borne inf et une borne sup, il fournit alors une solution réalisable accompagnée d'un écart relatif entre les deux (un gap).
- `‘‘CPXPARAM_MIP_Limits_Nodes’’, y)` : fixe le nombre de nœuds explorés dans l'arbre de branchement à maximum `y` nœuds. Comme pour le paramètre précédent, cela permet parfois d'obtenir une solution réalisable avec une garanti expérimentale.
  Remarque importante : si l'on désire obtenir une borne (inf en minimisation par exemple) par un PLNE, on peut bien entendu lancer le PLNE relaxé (c'est-à-dire avec des variables non entières), mais il est aussi possible de laisser CPLEX s'exécuter uniquement sur la racine en fixant `y` à 0 ; ou même avec un nombre réduit `y` de nœuds ou avec un temps `x` secondes d'éxécution : on obtient alors une meilleure relaxation : on appelle parfois dans la littérature cette borne la “borne CPLEX pour `x` nœuds et/ou `y` secondes”.
- `CPXPARAM_MIP_Tolerances_AbsMIPGap, z)` : fixe à `z`% l'écart relatif (gap) attendu pour le problème : le solveur s'arrête alors dès qu'il a trouvé une solution correspondant à un gap de `z`%.

La liste des paramètres est ici
`https://www.ibm.com/docs/en/icos/20.1.0?topic=parameters-absolute-mip-gap-tolerance`
en utilisant le nom des paramètres correspondant au langage C.

5.7 Tests experimentaux

Question 17 *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
: - tester avec le PLNE compact
- ne mettre que les inégalités Lazy (Branch-and-Cut sans renforcement)
- puis les inégalités User Cut
- puis l'heuristique primale.
Que constatez-vous ?

Quatrième partie

Le problème du voyageur de commerce (TSP)

Cette partie propose un exercice supplémentaire dont la correction sera peu à peu en ligne : voir sur le site.

6 Résolution du problème du voyageur de commerce

6.1 Résolution heuristique

On considère à présent le problème du voyageur de commerce euclidien (TSP) : voir cours si besoin.
On veut utiliser les connaissances vues pour le stable pour construire une heuristique et une métaheuristique pour le problème du voyageur de commerce.
Les instances du répertoire `Instances/TSP` sont issues de la TSPLIB, benchmark d'instances très utilisées sur le web pour tester des méthodes.

Question 18 *Créer un fichier* `Graph_TSP.jl` *permettant de lire les instances du répertoire* `Instances/TSP`*. Ces fichiers sont issues de la librairie d'instances de la TSPlib.*
Vous pouvez stocker une instance de la TSP lib en ayant simplement le nombre de points et deux tableaux X et Y portant les coordonnées des points dans un plan.
Pour rappel, vous pouvez trouver sur les sites Julia tout ce qu'il vous faut pour ce TP : Voir les indications en section 1.7
`https://docs.juliaplots.org/latest/` *pour dessiner des points et des traits*
ou `https://juliagraphs.org/Graphs.jl/dev/plotting/` *si vous voulez des dessins plus évolués.*

Question 19 *Visualisez l'instance par un nuage de points (vous pourrez voir que att48 sont les capitales des USA ou France les grandes villes de métropoles etc).*

Question 20 *Coder l'heuristique gloutonne dite "du plus proche voisin" et visualisez votre solution.*

Question 21 *Coder une méthode de descentes stochastiques itérées avec voisinage 2-OPT.*

6.2 PLNE compacts

Question 22 *Coder les trois modèles PLNE compacts (MTZ, flot agrégé, flots désagrégé) vus en cours pour le TSP.*

Question 23 *Résoudre des instances du répertoire d'instance* `Instances/TSP`*. Comparer les résultats obtenus avec votre heuristique.*

6.3 PLNE à nombre exponentiel d'inégalités

Le but de cette section est de c oder la formulation du TSP utilisant les inégalités de coupes (dites de Menger) vue en cours.
Voici cette formulation

$$
\begin{aligned}
& Min \sum_{e \in E} c(e)x(e) \\
& \sum_{e \in \delta(v)} x(e) = 2, \text{ pour tout } u \in V, && (9) \\
& \sum_{e \in \delta(W)} x(e) \geq 2, \text{ pour tout } W \subsetneq V \text{ et } W \neq \emptyset, && (10) \\
& x(e) \geq 0, \text{ pour tout } e \in E \\
& x(e) \in \{0, 1\}, \text{ pour tout } e \in E.
\end{aligned}
$$

Les contraintes (9) de degré sont au nombre de $|V|$ et sont toujours présentes dans la formulation : on les place donc dans le PLNE dés le départ.
En revanche les contraintes (10) sont en nombre exponentiel mais elles sont séparables en temps polynomial !

Notons ici x^* la solution à séparer. On distingue deux cas correspondant aux cas 'LazyConstraint" et 'UserCut" vu dans la partie précédente :

cas entier : si CPLEX active le callback "LazyConstraint", c'est que les composantes x^* sont toutes entières. Pour cela, on peut regarder le graphe H induit par les arêtes ij telle que $x^*_{ij} = 1$. La séparation de ce cas est alors très simple : il suffit de détecter si le graphe H est connexe : en fait on cherche même simplement à détecter si H est un cycle passant par tous les sommets. En effet de part les inégalités (9), H est un ensemble de cycles.

La réponse au problème de séparation se ramène alors à partir d'un sommet u quelconque, de chercher un de ses deux voisins puis d'itérer de voisin en voisin jusqu'à revenir à u. Si on est passé par tous les sommets, cela veut dire que x^* est réalisable (on n'ajoute aucune inégalité). Sinon on a repéré un ensemble de sommets créant une partition des sommets qui est isolée d'autres sommets : ils représentent un ensemble W de sommets correspondant à une coupe (10) violée par x^* : on ajoute alors l'inégalité.

cas fractionnaire : si CPLEX active le callback "UseCut", c'est a priori que des composantes de x^* sont fractionnaires. On doit séparer alors x^* en recherchant s'il existe un ensemble $W \subsetneq V$ et $W \neq \emptyset$ tel que $\sum_{e \in \delta(W)} x(e) < 2$. Si c'est le cas, on ajoute l'inégalité correspondant à W, sinon il n'y a pas d'inégalité violée par x^*.

Pour déterminer un tel ensemble W, on va rechercher ensemble W minimal selon $\sum_{e \in \delta(W)} x(e)$, c'est-à-dire en recherchant une coupe de valeur minimale dans le graphe complet G de l'instance où les distances sont remplacée par les valeurs x^*_{ij} pour chaque arête ij.

Question 24 *Coder dans un premier temps la séparation entière uniquement : elle permet en effet d'obtenir un algorithme de Branch&Bound valide.*
Tester la sur des petites instances.

6.4 Cas fractionnaire pour les inégalités de coupes de Menger

On désire à présent ajouter la séparation des inégalités dans le cas où x^* est fractionnaire (UserCut).

Vous doit pour cela utiliser un algorithme de recherche d'une coupe minimale dans le graphe G complexe de l'instance en munissant ce graphe des valeurs x^*_{ij} sur chaque arête. Il y a encore peu d'implémentation de ces algorithmes en Julia. Voici un "wrapper" construit depuis le C avec une implémentation efficace de l'algorithme de Boykov-Kolmogorov permettant de résoudre le problème FlotMax/CoupeMin. `https://github.com/Gnimuc/BKMaxflow.jl`
A la PPTI, le code est installé, mais sur votre machine, il faut taper dans Julia

```
Pkg.add(url="https://github.com/Gnimuc/BKMaxflow.jl.git")
```

ATTENTION : il ne faut pas cumuler le `using BKMaxflow` ci-dessous avec un `using Graphs` permettant de manipuler des graphes : en effet, cela provoque un conflit dans les notations (Si vous avez déjà fait un using, il semble nécessaire de sortir et revenir dans Julia... si quelqu'un trouve un autre moyen, merci de le signaler).
Voici l'exemple donné par le site

```
using BKMaxflow

weights, neighbors = create_graph(JuliaImpl{Float64,Int}, 4) # Graphe a 4 sommets

BKMaxflow.add_edge!(weights, neighbors, 1, 2, 1., 1.)
    # Arc de 1 vers 2 avec capacite min et capacite max
BKMaxflow.add_edge!(weights, neighbors, 1, 3, 2., 2.)
BKMaxflow.add_edge!(weights, neighbors, 2, 3, 3., 4.)
BKMaxflow.add_edge!(weights, neighbors, 2, 4, 5., 5.)
BKMaxflow.add_edge!(weights, neighbors, 3, 4, 6., 6.)

# le tableau weight doit etre mis sur 2 lignes pour differencier les capa min et max
w = reshape(weights, 2, :)

# recherche du flot max du sommet 1 au sommet 4
flow, label = boykov_kolmogorov(1, 4, neighbors, w)
```

`Flow` est la valeur du flot max qui est égale à celui de la coupe min.
`label` est un vecteur sur les sommets indiquants `BK_S` ou `BK_T` suivant si le sommet est du

côté de la source s ou du puits t.
Ainsi, on obtient les sommets formant la partition $(W, V \setminus W)$ avec $s \in W$ et $t \notin W$, c'est-à-dire séparant s et t.
Il s'agit d'un graphe orienté : pour traiter un graphe non-orienté, il faut mettre deux arcs aller-retour par arête en mettant les mêmes capacités dessus.

Question 25 *Ajouter la séparation des inégalités dans le cas fractionnaire.*
Vous pouvez également ajouter une heuristique primale (par exemple par un algorithme glouton tentant d'ajouter une à une les arêtes dans l'ordre décroissants des valeurs x^*_{ij} tant qu'elle ne crée pas un sommet de degré 3 et qu'elle ne forme pas un cycle ne passant pas par tous les sommets.
Tester la formulation en comparaison avec la méthode compacte vu auparavant.

6.5 Le Logiciel Concorde

Le logiciel Concordre `https://www.math.uwaterloo.ca/tsp/concorde.html` est un algorithme de Branch&Cut dédié au TSP qui a battu tous les records. Vous pouvez comparer votre code à Concorde... mais vous ne gagnerez pas !
Par contre, si vous avez juste l'idée d'ajouter une contrainte au TPS (par exemple le rendre asymétrique, ajouter des contraintes de précédences immédiates entre villes etc) : vous pouvez adapter votre code, alors qu'il sera bien difficile d'aborder cela avec le code de Concorde.

Cinquième partie

Annexes

Annexe : Julia : installation et autres petites choses

6.6 Si vous voulez l'avoir chez vous : installation de Julia sous Linux

Pour installer Julia sous Linux (sur x86 64 bits, ce qui est le plus courant), récupérer sur `https://julialang.org/downloads` l'archive de Julia `julia-xxxx-linux-x86_64.tar.gz` et décompresser l'archive `tar -zxvf julia-xxx-linux-x86_.tar.gz` à l'emplacement de votre choix, par exemple dans `/home/moi/logiciels`.
Le logiciel `julia` sera alors `/home/moi/logiciels/julia-xxxx/bin/julia`.
Il est utile d'ajouter à votre fichier .bashrc une ligne permettant d'accéder facilement à Julia :
Taper `gedit .bashrc` dans votre répertoire utilisateur.
Ajouter tout à la fin `export PATH=$PATH:/home/moi/logiciels/julia-xxxx/bin`
Fermer et réouvrer votre terminal (ou taper source /bashrc), à partir de maintenant, il vous suffira de taper `julia` pour lancer le programme.

6.7 Quelques bugs résolus

Il se peut que vous ayez des bug étranges avec des affichages signalant une erreur du type

```
Warning: The dynamically loaded GMP library (v"4.3.2" with __gmp_bits_per_limb == 64)
does not correspond to the compile time version (v"6.2.0" with __gmp_bits_per_limb == 64).
Please rebuild Julia.
Base.GMP gmp.jl:100
```

Cela provient de la variable système `LD_LIBRARY_PATH` qui est très certainement modifiée dans votre `/.bashrc`.
Si vous ôtez les lignes correspondant à la modification de `LD_LIBRARY_PATH` cela devrait régler le problème... mais cela peut poser problème dans d'autre module !
Une façon plus locale est de taper à chaque fois `export LD_LIBRARY_PATH=""`
juste après avoir ouvert un terminal pour commencer vos exercices ou projet en Julia !

Annexe : Jupyter Notebook

Il faut avoir préalablement installer Jupyter notebook. Puis taper dans Julia :

```
import Pkg
Pkg.add("IJulia")
notebook()
```

7 Julia avec Cplex

Obtenir une licence académique CPLEX

Pour cela :

- créer un compte IBMid sur la base de votre addresse universitaire
  (attention la validation par mail avec envoi de code est lente... or il y a un délai entre les 2) `https://www.ibm.com/account/reg/fr-fr/signup?formid=urx-19776`
- aller sur internet à l'ACADEMIC INITIATIVE d'IBM (et non à la verson free qui est limitée) `https://www.ibm.com/academic/home`
- loguez vous et singalez que vous êtes dans le cadre de l'ACADEMIC INITIATIVE

- aller à `https://www.ibm.com/academic/technology/data-science`
  Puis sur l'onglet de milieu de page Software
  A gauche, il y a alors un quart de page CPLEX Cliquer sur Download (Il y a des browsers qui peuvent rester bloqué... si trop de protection par exemple (firefox), tenter sur des browsers plus "ouverts" (chromium-browser par exemple)
- vous arrivez alors à la page de téléchargement **en haut, il y a des choix dans un onglet entre IBM Download Director ou http : prendre http ! ! ! !**
- Choisir votre version (linux, windows, MacOs...) et download
- Prévoyez un emplacement pour installer Cplex pour lequel vous aurez le chemin car il servira à JuMP par la suite
- Sur linux, il s'agit d'un binaire à lancer qui se dézipera et installera Cplex (penser à rendre le binaire exécutable si besoin avec `chmod +x`

Julia, JuMP et Cplex

Si vous êtes sur votre machine, pour utiliser JuMP, il faut bien entendu taper une fois au début

```
import Pkg; Pkg.add(''JuMP'')
```

Pour que JuMP ne soit pas qu'un outil vide d'algorithme de résolution, il faut aussi ajouter un solveur : par exemple ici ce sera CPLEX.

Pour utiliser votre licence CPLEX :

Il faut auparavant avoir installer CPLEX en payant une licence (très élevée) ou en ayant une licence académique (voir annexe).

Par exemple vous pouvez le mettre dans le répertoire `/home/moi/logiciels/CPLEX_studioxxx/cplex`.

Puis définir une variable Julia vers le répertoire contenant CPLEX.

```
ENV["CPLEX_STUDIO_BINARIES"] =
"/home/moi/logiciels/CPLEX_studioxxx/cplex/bin/x86-64_linux"
Pkg.add("CPLEX")
Pkg.build("CPLEX")
```

Au début d'un programme, n'oubliez pas `using CPLEX`.

Par exemple, Pour utiliser la licence CPLEX installée à la PPTI `ENV["CPLEX_STUDIO_BINARIES"] = "/Vrac/MAOA/cplex/bin/x86-64_linux"`