TP3 - *Branch-and-Cut*

Intervenant : Diego Perdigão Martino

Objectifs

— Appliquer un algorithme *Branch-and-Cut* pour résoudre le problème du voyageur de commerce en utilisant l'API *Python* de CPLEX *Optimization Studio* en ajoutant les coupes nécessaires pour obtenir des solutions entières faisables et optimales.

1 Le problème du voyageur de commerce

Soit $G = (N, A)$ un graphe complet (*i.e.*, tous les sommets sont liés deux à deux par une arête) où N correspond au nombre de nœuds et A le nombre d'arcs du graphe. On associe à chaque arc $(i, j) \in A$ un poids c_{ij} représentant le coût de déplacement de i vers j.

Le Problème du Voyageur de Commerce (en anglais, *Traveling Salesman Problem* - TSP) repose sur le graphe G et consiste a déterminer l'ordre de passage dans chaque nœud du graphe (ici, chaque nœud représente un client qui doit être visité) qui minimise le coût total de déplacement tenant en compte les coûts (poids) des arcs. Cet ordre est modélisé par une tournée qui doit commencer par le dépôt (nœud 0), visiter tous les clients une seule fois et revenir au dépôt.

Exemple :

Soit $N = 5$. Une tournée valide pour le problème est donnée par {0, 4, 1, 3, 5, 2, 0} et le coût est exprimé par $c_{0,4} + c_{4,1} + c_{1,3} + c_{3,5} + c_{5,2} + c_{2,0}$. Cependant, les tournées {0, 4, 1, 5, 2, 0} et {0, 4, 1, 3, 5, 1, 2, 0} ne sont pas valides car la première ne contient pas le client 3 et sur la seconde, le client 1 est visité deux fois.

La formulation mathématique du TSP est donnée par les équations ci-dessous.

$$min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$s.t. \quad \sum_{j\in N} x_{ij} = 1 \qquad \forall\, i \in N, i \neq j \tag{2}$$

$$\sum_{i\in N} x_{ij} = 1 \qquad \forall\, j \in N, j \neq i \tag{3}$$

$$x(A(S)) \leq |S| - 1 \qquad \forall\, S \subset N, S \neq \emptyset \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in A \tag{5}$$

La fonction objectif (1) consiste à minimiser le coût total de déplacement. Les contraintes (2) définissent au plus un arc sortant et les contraintes (3) au plus un arc entrant sur chaque nœud du graphe. Les contraintes (4) empêchent la formation des sous tours S (*i.e.,* des tournées isolées qui ne contiennent pas le dépôt) en énumérant tous les cas possibles. Ces contraintes rendent le modèle très difficile à résoudre puisqu'elles nécessitent l'énumération de tous les sous ensembles S qui pourront être éventuellement créés. Ensuite, les variables x_{ij} du problème (5) valent 1 si l'arc (i, j) est sélectionné et 0 sans le cas contraire.

1. Implémentez le modèle donné par les équations (1)-(5) **sans les contraintes** (4).
   Vous pouvez créer une procédure qui permet la lecture d'un fichier de paramètres (instance) qui contient le nombre de clients N ainsi que la matrice de coûts c_{ij} représentant la distance entre chaque pair de nœuds. Cela vous permettra de tester plusieurs instances sans avoir besoin de changer le code à chaque fois.
2. Exécutez le modèle avec l'instance `TSP_small`. Exportez ensuite le modèle et vérifiez que l'ensemble des contraintes ainsi que la fonction objectif sont correctement définis. Résolvez-le. La solution obtenue est-elle réalisable ?

3. Ajoutez les coupes nécessaires pour contourner la non conformité de la solution jusqu'à ce que la solution soit réalisable et optimale. Quelle est la tournée optimale obtenue ?
4. Faites de même pour l'instance `TSP_large`. Quelle est la tournée optimale obtenue ?

(BONUS) Pour aller plus loin...

Utilisez une *LazyConstraintCallback* et créez une fonction auxiliaire qui permet l'identification des sous-tours dans la solution courante pour ajouter ainsi les contraintes nécessaires pour que la solution devient faisable, c'est-à-dire, sans aucun sous-tour. Utilisez les instances `TSP_small` et `TSP_large` pour tester l'algorithme.