Devoir 1 - Optimisation d'un réseau de télécommunication

Remise le 19/02/2023 **(avant minuit) sur Moodle pour tous les groupes.**

Consignes

- Le devoir doit être fait par groupe de 2 au maximum. Il est fortement recommandé d'être 2.
- Lors de votre soumission sur Moodle, donnez votre rapport en format **PDF** ainsi que vos fichiers de code à la racine d'un seul dossier compressé nommé (`matricule1_matricule2_Devoir1.zip`).
- Indiquez vos noms et matricules en commentaires au dessus des fichiers .py soumis.
- Toutes les consignes générales du cours (interdiction de plagiat, etc.) s'appliquent pour ce devoir.
- Il est permis (et encouragé) de discuter de vos pistes de solution avec les autres groupes. Par contre, il est formellement interdit de reprendre le code d'un autre groupe ou de copier un code déjà existant (StackOverflow ou autre). Tout cas de plagiat sera sanctionné de la note minimale pour le devoir.

Enoncé du devoir

Si vous avez déménagé récemment, vous avez probablement changé votre forfait internet. Après avoir choisi un forfait, vous avez sûrement dû donner votre adresse pour confirmer que le type de forfait que vous vouliez était disponible dans votre nouveau logement. En effet, même si l'on accède généralement à l'internet grâce à des technologies sans fil comme le wifi, le LTE ou le 5G (celui qui transmet la COVID), l'internet dépend encore d'immenses réseaux de câbles. Comme la construction et l'entretien de ces câbles a un coût, les entreprises exploitant ces réseaux doivent faire des choix quant aux parties du territoire où elles vont fournir ces services. Ces entreprises doivent donc atteindre certains clients de manière à optimiser leur rentabilité. L'atteinte de certaines régions (ou clients) permet de tirer un profit ; ainsi, si le réseau ne les atteint pas, l'entreprise perdra ces revenus, mais économisera en contre-partie le coût de construction de réseau pour les atteindre. Il y a donc un certain compromis à faire dans la planification du réseau.

C'est le problème auquel fait face l'entreprise *Belle communication*. En temps normal, la planification de leur réseau est gérée par leur équipe de recherche opérationnelle, mais étant donné la pénurie de main d'œuvre et l'augmentation des coûts en raison de l'inflation, ils se sont tournés vers nous. L'entreprise veut étendre son réseau dans 5 zones géographiques (i.e., les 5 instances du devoir) et vous avez la tâche de leur fournir des solutions qui permettent de minimiser les coûts de ces expansions.

L'objectif de ce devoir est ainsi de concevoir la disposition d'un réseau qui minimise les coûts de construction ainsi que les pénalités associées à l'exclusion de certaines régions. Ces derniers sont référés comme des *noeuds terminaux* dans le réseau. D'autres noeuds, référés comme des *noeuds de transition*, sont également présents dans le réseau mais n'engendrent aucune pénalité s'ils ne sont pas choisis. Par ailleurs, plusieurs arêtes (liens non-dirigés entre les noeuds) indiquent où il est possible pour l'entreprise de construire une connexion. Deux contraintes doivent être prises en compte : (1) le réseau doit être entièrement connecté, et (2) il ne doit y avoir aucun cycle dans le réseau. Autrement dit, le réseau correspond à un *arbre*, au sens mathématique du terme. Pour résoudre ce problème, vous implémenterez une méthode de recherche locale en y ajoutant un mécanisme permettant de sortir des optima locaux. Différentes instances vous sont fournies. Elles sont nommées selon le schéma `reseau_X_N_E_T.txt` où :

1. `X` est le nom de l'instance.
2. `N` est le nombre de clients.
3. `E` est le nombre de connexions potentielles entre les clients.

4. `T` est le nombre de clients terminaux (qui engendrent une pénalité s'ils ne sont pas choisis).

Chaque fichier d'instance contient $E + T + 2$ lignes et a le format suivant.

```
8
11
E 1 2 5
E 1 3 4
E 1 5 7
E 2 4 150
E 3 4 75
E 3 5 2
E 3 6 4
E 5 6 2
E 6 7 2
E 6 8 10
E 7 8 3
T 1 100
T 4 50
T 5 40
T 7 5
T 8 20
```

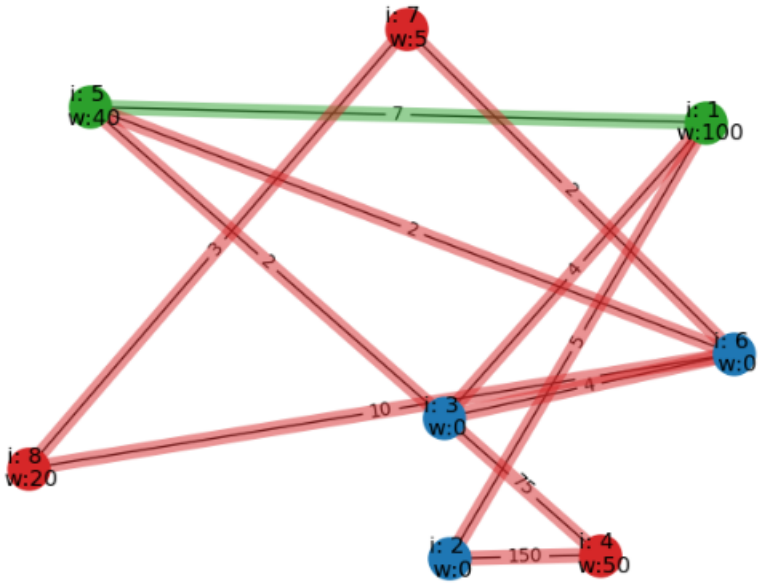
À titre d'exemple, cette instance (`reseau_A_8_11_5.txt`) contient 8 clients, reliés entre eux par 11 arêtes avec 5 noeuds terminaux. Les informations sont les suivantes :

1. La première ligne (N) indique le nombre de clients potentiels (le nombre de noeuds dans le graphe).
2. La deuxième ligne (E) indique le nombre de connexions potentielles (le nombre d'arêtes).
3. Les lignes commençant par un E indiquent des connexions potentielles ainsi que leur coût de construction. Par exemple, la troisième ligne indique que l'ajout d'un lien entre le client 1 et 2 a un coût de 5.
4. Les lignes commençant par un T indiquent les noeuds terminaux ainsi que les coûts de pénalité qui leur sont associés. Par exemple, la dernière ligne indique que l'exclusion du client 8 de la solution résulterait en une pénalité de 20.

Le format attendu d'une solution est un fichier de n lignes, où n représente les arêtes présentes dans la solution. Le format est présenté ci-dessous.

```
n_1 n_2
n_3 n_4
...
n_{N-1} n_{N}
```

Pour rappel, la solution doit être un arbre, c'est-à-dire un graphe sans cycle et n'avoir qu'une seule composante connexe. Pour faciliter la lecture d'une solution, un outil de visualisation vous est fourni. Les noeuds terminaux faisant partie de la solution sont en *vert* et les autres sont *rouges*. Les noeuds non-terminaux sont en *bleu*. Finalement, les arêtes exclues de la solution sont *rouges* et celles qui en font partie sont *noires*. En exécutant la commande `main.py`, un graphe interactif est généré.



Implémentation

Vous avez à votre disposition un projet python. Dans ce projet, une `solution` est un iterable (set ou list) de taille n contenant les arêtes qui sont sélectionnées pour la solution. Les arêtes sont représentées comme un tuple (x, y) avec x et y correspondant aux deux noeuds liés à l'arête. Plusieurs fichiers vous sont fournis :

- `network.py` qui implémente la classe `PCSTP` pour lire les instances, sauvegarder les solutions, les valider et les visualiser.
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme enregistre également votre meilleure solution dans un fichier au format texte et sous la forme d'une image.
- `solver_naive.py` : implémentation d'un solveur naïf qui connecte 2 noeuds terminaux au moyen avec le plus court chemin. Notez également l'utilisation de la librairie networkx que vous êtes autorisés à utiliser (sans aucune obligation).
- `solver_advanced.py` : implémentation de votre méthode de résolution qui sera évaluée pour ce devoir.

Notez bien qu'un vérificateur de solutions est disponible. Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 5 instances sont mises à votre disposition :

- `reseau_A_8_11_5.txt` : pour tester votre implémentation (optimum : 63)
- `reseau_B_64_192_64.txt` : utilisé pour évaluer votre solveur (optimum : 3908)
- `reseau_C_640_1280_160.txt` : utilisé pour évaluer votre solveur (optimum : 39315)
- `reseau_D_640_4135_50.txt` : utilisé pour évaluer votre solveur (optimum : 10675)
- `reseau_E_640_40896_160.txt` : utilisé pour évaluer votre solveur (solution à battre : 29697)

Votre code sera également évalué sur une instance cachée (`reseau_X`), de taille légèrement inférieure à la dernière. Pour vérifier que tout fonctionne bien, vous pouvez exécuter le solveur naïf comme suit.

```
python3 main.py --agent=naive --infile=instances/reseau_A_8_11_5.txt
```

Un fichier `solution.csv` et une image `visualization.png` seront générés.

Production à réaliser

Vous devez compléter le fichier `solver_advanced.py` avec votre méthode de résolution. Au minimum, votre solveur doit contenir un algorithme de recherche locale. C'est à dire que votre algorithme va partir d'une solution complète et l'améliorer petit à petit via des mouvements locaux. Réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous devez également intégrer un mécanisme de votre choix pour vous s'échapper des minima locaux. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=instances/reseau_A_8_11_5.txt
```

Un rapport **succinct** (2 pages de contenu, sans compter la page de garde, figures, et références) doit également être fourni. Dans ce dernier, vous devez présenter votre algorithme de résolution, vos choix de conceptions, vos analyses de complexité. Reportez également les résultats obtenus pour les différentes instances.

Critères d'évaluation

L'évaluation portera sur la qualité du rapport et du code fournis, ainsi que sur les performances de votre solveur sur les différentes instances. Concrètement, la répartition des points (sur 20) est la suivante :

— 10 points sur 20 sont attribués à l'évaluation de votre solveur. L'instance A rapporte 2 point si l'optimum est trouvé, et 0 sinon. Les instances B, C, D, et E rapportent 2 points si l'optimum est trouvé et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Si aucun groupe ne trouve l'optimum pour ces instances, la solution du meilleur groupe est prise comme référence des 2 points. Le temps d'exécution est de 30 minutes par instance.
— 10 points sur 20 sont attribués pour le rapport. Pour ce dernier, les critères sont la qualité générale, la qualité des explications, et le détail des choix de conception.
— 2 points bonus seront attribués au groupe ayant le meilleur résultat pour l'instance cachée (X).

⚠Il est attendu que vos algorithmes retournent une solution et un coût correct. Par exemple, il est interdit de modifier artificiellement les contraintes entre les noeuds ou autre. Un algorithme retournant une solution non cohérente est susceptible de ne recevoir aucun point.

Conseils

Voici quelques conseils pour le mener le devoir à bien :

1. Tirez le meilleur parti des séances de laboratoire encadrées afin de demander des conseils.
2. Inspirez vous des techniques vues au cours, et ajoutez-y vos propres idées.
3. Tenez compte du fait que l'exécution de vos algorithmes peut demander un temps considérable. Dès lors, ne vous prenez pas à la dernière minute pour réaliser vos expériences.
4. Bien que court dans l'absolu, prenez garde au temps d'exécution. Exécuter votre algorithme sur les instances données prend 2h (sans l'instance secrète et l'instance A). Organisez au mieux votre temps de développement/évaluation.

Remise

Vous remettrez sur Moodle une archive zip nommée `matricule1_matricule2_Devoir1` contenant :

- Votre code commenté au complet.
- Vos solutions aux différentes instances nommées `solutionX` où X est le nom de l'instance.
- Votre rapport de présentation de votre méthode et de vos résultats, d'au maximum 2 pages.