Devoir 2 - Optimisation des soins à domicile

Remise le 26/03/2023 **(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 la soumission sur Moodle, donnez votre rapport en **PDF** ainsi que votre code **commentés** à la racine d'un seul dossier compressé nommé (`matricule1_matricule2_Devoir2.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.

Énoncé du devoir

Le vieillissement de la population amène son lot de défis pour le système de santé, notamment pour les soins à domicile. En raison de leur mobilité réduite, certaines personnes âgées préfèrent recevoir des soins à domicile. Toutefois, ces personnes ne sont pas disponibles toute la journée et il faut donc que les infirmières ou médecins qui les prennent en charge adaptent leur itinéraire en fonction de la disponibilité des patients. Normalement, le gouvernement fait appel à des consultants pour régler ce type de problème, mais étant donné la pénurie de main d'oeuvre et les critiques récentes des consultants dans les affaires gouvernementales, le ministère de la santé fait appel à vous. Vous serez chargés d'optimiser le trajet de Dr. Julian Nguyen, un médecin qui donne des soins à domicile.

L'objectif est de minimiser le temps total nécessaire à la visite de tous ses patients, tout en respectant les disponibilités de ceux-ci.

Formalisation du problème

Soit $G = (N, A)$ un graphe complet, orienté avec $N = \{0, 1, \ldots, n\}$. N est l'ensemble des noeuds, le noeud 0 représente le noeud de départ du Dr Nguyen et A est une matrice $N \times N$ où $a_{ij} \in A$ représente le temps nécessaire pour se rendre au noeud (patient) i en partant du noeud (patient) j. De plus, pour chaque patient, il existe une fenêtre de temps $[l_i, u_i]$ où le patient $i \in N$ est disponible. Cette fenêtre de temps indique que le patient i ne peut être visité avant l_i ou après u_i. On admet un temps d'attente, c'est à dire qu'il est possible d'arriver chez un patient avant le début de sa disponibilité à condition d'attendre qu'il soit disponible. Par exemple, si le patient 1 n'est disponible qu'à compter de 13h, il est possible pour Dr Nguyen de s'y rendre à 12h, à condition de patienter 1 heures de 12h à 13h (par exemple lire le journal avant que le patient ne soit disponible). Il n'est pas possible de quitter un patient i avant le début de sa disponibilité l_i. On suppose également que le temps de traiter un patient (c'est-à-dire le temps où le docteur reste au domicile du patient) est nul.

Ainsi, étant donnée une tournée P, le temps de départ D_{p_k} du patient p_k est $D_{p_k} = \max(A_{p_k}, l_{p_k})$ où $A_{p_k} = D_{p_{k-1}} + a_{p_{k-1}p_k}$. Autrement dit, le départ depuis un patient p_k correspond à la valeur maximale entre : (1) la borne inférieure de sa disponibilité (l_{p_k}) et, (2) la valeur de départ du patient précédent dans le tour ($D_{p_{k-1}}$) plus la durée du trajet pour atteindre le patient actuel ($a_{p_{k-1}p_k}$). Une solution au problème st représentée

par $P = (p_0 = 0, p_1, \ldots, p_n, p_{n+1} = 0)$ où $(p_1, p_2, \ldots, p_n)$ est une permutation des noeuds de $N\backslash\{0\}$. Autrement dit, c'est un cycle hamiltonien où p_k est l'indice du patient visité en k^e position de la tournée. Mathématiquement, on a le modèle suivant :

$$\begin{aligned} &\text{minimize} && D_{p_{n+1}} \\ &\text{subject to} && \sum_{k=0}^{n+1} \omega(p_k) = 0 \end{aligned} \tag{1}$$

où :

$$\omega(p_k) = \begin{cases} 1 & \text{si } A_{p_k} > u_{p_k} \\ 0 & \text{sinon} \end{cases} \quad \text{avec } A_{p_{k+1}} = \max(A_{p_k}, l_{p_k}) + a_{p_k, p_{k+1}}$$

La contrainte $\sum_{k=0}^{n+1} \omega(p_k) = 0$ garantit que toutes les contraintes sur les fenêtres de temps sont respectées. Notez bien qu'une solution valide doit (1) visiter tous les patients une et une seule fois, et (2) respecter les contraintes des fenêtres de temps.

⚠**Notez bien que le coût à minimiser est le temps final de retour au dépôt (aussi appelé le *makespan*) et non la somme des temps de trajets. Il faut aussi tenir compte du temps perdu à attendre si on arrive trop tôt chez un patient.**

Différentes instances vous sont fournies. Elles sont nommées selon le schéma `X_N.txt` avec X le nom de l'instance, et N le nombre de noeuds dans le problème (incluant le noeud de départ). Chaque fichier d'instance contient $2N+1$ lignes et a le format suivant.

```
N
d(1,1)   d(1,2)   ...   d(1,N)
d(2,1)   d(2,2)   ...   d(2,N)
...
d(N,1)   d(N,2)   ...   d(N,N)

tw(1,1)  tw(1,2)
tw(2,1)  tw(2,2)
..
tw(N,1)  tw(N,2)
```

Ainsi, la première ligne (N) indique le nombre de noeuds dans la tournée à réaliser (le nombre de patients +1 pour le noeud de départ), les N lignes suivantes forment une matrice indiquant les distances entre les noeuds du graphe. La première ligne représente le temps du trajet partant du noeud de départ jusqu'aux autres noeuds; à l'inverse, la première colonne représente le temps du temps partant de chacun des patients jusqu'au noeud de départ. On note deux choses. Notez que les distances sont asymétriques, ce qui correspond à une situation réaliste où un chemin différent doit parfois être pris pour aller d'un point A à B et inversement. Finalement, les N dernières lignes forment une matrice indiquant les bornes inférieures l_i et supérieures u_i où les patients peuvent être visités.

Le format attendu d'une solution est un fichier de $N+2$ lignes : la première ligne contient le nombre de noeuds de l'instance. Ceci sert à valider les solutions obtenues. Ensuite, on a la tournée, qui commence et finit par 0 -le noeud de départ- et contient N noeuds représentant une permutation des noeuds du problème.

```
N
0
p_1
...
p_{N-1}
0
```

A titre d'exemple, l'instance suivante (`A_4.txt`) contient 3 patients (4 noeuds en tenant compte du noeud de départ).

```
4
0 43.0116 36.0555 33.541
53.0116 10 17.0711 21.1803
46.0555 17.0711 10 15
43.541 21.1803 15 10
0 960
43 283
36 276
33 273
```

Une solution possible au problème est la tournée suivante : $\{0,2,1,3,0\}$.

```
4
0
2
1
3
0
```

Pour faciliter la lecture d'une solution, un outil de visualisation vous est fourni. Chaque noeud correspond à un patient (avec l'identifiant du patient) ou au noeud de départ (noeud 0, coloré en vert). Sur chaque arête on peut voir la durée nécessaire pour la parcourir. Les arêtes dans le tour sont colorée en vert. Un exemple est donné à la Figure 1.

Implémentation

Vous avez à votre disposition un projet python. Quatre fichiers vous sont fournis :

- `tsptw.py` qui implémente la classe `TSPTW` pour lire les instances, construire et stocker vos solutions.
- `main.py` vous permettant d'exécuter votre code sur une instance donnée. Ce programme stocke également votre meilleure solution dans un fichier au format texte et sous la forme d'une image.
- `solver_naive.py` qui implémente une résolution aléatoire du problème.
- `solver_advanced.py` qui implémente votre méthode de résolution du problème.

Vous êtes également libres de rajouter d'autres fichiers au besoin. De plus, 5 instances sont mises à votre disposition :

- `A_4.txt` d'optimum connu égal à 117.85. Cette instance ne rapporte aucun point et est donnée pour réaliser vos tests.
- `B_20.txt` d'optimum connu égal à 592.06.

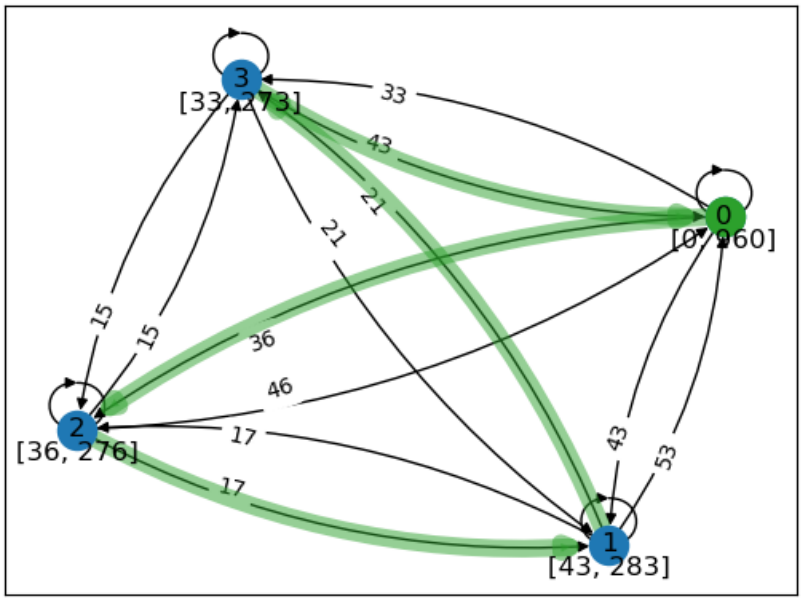

FIGURE 1 – Visualisation d'une solution.

— `C_28.txt` d'optimum connu égal à 837.71.
— `D_60.txt` ; meilleure solution connue : 526.
— `E_100.txt` ; meilleure solution connue : 787.

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

```
python3 main.py --agent=naive --infile=instances/A_4.txt
```

Un fichier `solution.txt` 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 amélioré par au moins une des métaheuristiques suivantes : *recherche tabou*, *algorithmes génétiques*, *beamsearch*, *GRASP*, ou *ant colony optimization*. Vous êtes libres de choisir la métaheuristique que vous souhaitez implémenter.

Comme pour le premier devoir, réfléchissez bien à la définition de votre espace de recherche, de votre voisinage, de la fonction de sélection et d'évaluation. Vous êtes ensuite libres d'apporter n'importe quelle modification pour améliorer les performances de votre solveur, par exemple en combinant votre approche avec une autre métaheuristique ou de réutiliser des idées de votre premier devoir. Une fois construit, votre solveur pourra ensuite être appelé comme suit.

```
python3 main.py --agent=advanced --infile=instances/A_4.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, et vos analyses de complexité. Reportez également les résultats obtenus pour les différentes instances.

⚠**L'utilisation des algorithmes génétiques ou de l'optimisation par colonies de fourmis n'est pas aisée, et vous demandera probablement plus de travail si vous partez sur un de ces options.**

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 B rapporte 1 point si l'optimum est trouvé, et 0 sinon. L'instance C rapporte 2 points si l'optimum est trouvé et diminue progressivement jusqu'à 0 en fonction de la qualité de la solution. Les instances D, E rapportent 2 points si l'optimum est obtenu 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. L'instance cachée (X) rapporte entre 0 et 2 points en fonction d'un seuil raisonnable défini par le chargé de laboratoire. Le temps d'exécution est de 20 minutes par instance. Finalement, 1 point est consacré à l'appréciation générale de votre implémentation (bonne construction, commentaires, etc).
— 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. Un algorithme retournant une solution non cohérente est susceptible de 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 4 instances données prend 1h20. Organisez au mieux votre temps de développement et d'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.