INFO704 - TP : Livraison de Courier par drone

https://github.com/juliette-bois/tp1-analyse-algo

Sommaire

1. Premier algorithme d'approximation

a. Implémenter l'algorithme

Voir le code

b. Calcul de la complexité de l'algorithme de Kruskal

La complexité est de O(E log E) avec E = nombre d’arrêtes du graphe.
Le graphe étant complet et le nombre de noeuds étant le nombre de villes, on a donc :

$O((N * N - 1)log(N * N - 1))$

c. Conclure que l’algorithme est une 2-approximation

La distance effectuée par n’importe quel tour de livraison est plus grande que le poids d’un arbre couvrant car à la fin, le drone doit revenir à sa base.
De plus, la longueur du cycle obtenu par l’algorithme est plus courte que la longueur du cycle avec répétitions car on va parcourir moins de distance que lors du parcours en profondeur de l’arbre. En reprenant le schéma de la partie 1 du TP, la distance entre les noeuds 0 et 1 est plus courte que la somme des distances entre les noeuds 0 et 3 et 3 et 1.
Enfin, la longueur du cycle avec répétitions est 2 fois plus grande que le poids de l’arbre couvrant minimal car lors d’un parcours en profondeur, on parcourt 2 fois chaque arrête.
On peut donc dire que l’algorithme est une 2-approximation car il génère une solution qui est entre la solution optimum et 2 fois la solution optimum.

---

2. Approche exacte

a. Calculer la complexité de l’algorithme

La complexité de l'algorithme est de :

$O((N^2) * (2^N))$

b. En déduire un ordre de grandeur du nombre maximal de villes possibles

Si le calcul d'une ville prend une microseconde, alors le calcul du meilleur chemin pour 20 points est de 419 430 400 microsecondes soit 419,43 secondes (soit 6,98333 minutes).
Mais pour 30 points, cela représente déjà 966 367 641 600 microsecondes, soit un

peu plus de 11 jours et pour 40 points de 1.7 × 10^15 microsecondes, soit presque 55 ans. Pour 50 villes, le temps de calcul dépasse les 890 siècles. Donc on peut dire que la limite est entre 20 et 30 villes pour rester sur des temps de calcul "raisonnable".

c. Implémenter l’algorithme et tester combien de villes il peut traiter en environ 1 seconde

J'ai testé avec 8, 9 et 10 villes.

```
python3.9 exo2.py examples/Cities8
```

▼ Résultats

> La distance minimale est : 1157.740959369197
> Un cycle possible est : [0, 7, 5, 6, 1, 2, 4, 8, 3, 0]
> --- 0.20141887664794922 seconds ---

```
python3.9 exo2.py examples/Cities9
```

▼ Résultats

> La distance minimale est : 1160.3873493817057
> Un cycle possible est : [0, 7, 5, 6, 1, 2, 4, 8, 3, 9, 0]
> --- 1.7606940269470215 seconds ---

```
python3.9 exo2.py examples/Cities10
```

▼ Résultats

> La distance minimale est : 1176.1352883675124
> Un cycle possible est : [0, 7, 5, 6, 1, 2, 4, 10, 8, 3, 9, 0]
> --- 20.198479175567627 seconds ---

Ainsi, l'algorithme peut traiter environ 8 villes en 0.2 secondes et 9 villes en 1.7 secondes.

d. Comparer les résultats avec l’algorithme précédent

Le résultat ici est forcément le meilleur et le plus optimal.
Cependant, son calcul prend beaucoup de temps et nécessite de petite valeur de part sa complexité.

---

3. Deuxième algorithme d’approximation

a. Implémenter cet algorithme

Une tentative d'implémentation a été faite. Les résultats semblent cependant incohérents : pour 10 villes, on obtient ce résultat

```
La distance minimale est : 2082.501805225093
Un cycle possible est :  [0, 9, 1, 2, 3, 7, 4, 5, 6, 8, 10, 0]
```

Une implémentation en python a été faite à partir du code c++ de l'algorithme minCouplage donné dans le fichier `Simplex.py`.

On identifie bien les sommets de degré impair de l'arbre couvrant de poids minimal, et on fait bien l'union du résultat de l'algorithme minCouplage et de l'arbre.
Cependant il est bien possible que la matrice passée à l'algorithme soit incorrecte ou que l'algorithme pour parcourir le cycle Eulérien soit incorrect également.

b. Comparer les résultats de cet algorithme aux précédents

Il semble difficile de répondre à cette question, vu que nous obtenons des résultats incohérents lors de l'implémentation de l'algorithme précédent.

c. En déduire que cet algorithme est une 3/2-approximation

---

4. Formulation comme un programme linéaire

a. Pourquoi résoudre ce problème linéaire est équivalent à notre problème ?

Les règles que l'on applique sont en adéquation avec le problème. En effet, la première indique qu'on essaye de minimiser la somme des chemins empruntés par le livreur (on multiplie la distance par 0 ou 1 et on laisse le choix au solveur du 0 ou 1)
La seconde ainsi que la troisième règle indiquent qu'on doit avoir exactement un chemin par ville (arrivant et sortant)
La dernière règle indique que pour chaque sous ensemble de ville de taille N (>= 2), on a au plus N - 1 chemin (règle pour aider le solveur à trouver une solution cohérente)
L'ensemble de ces règles permettent de trouver une solution au problème

b. BONUS - Ecrire un programme qui résout le programme linéaire précédent (on pourra utiliser une librairie pour les programmes linéaires).

J'ai utilisé la librairie PuLP : https://pypi.org/project/PuLP/