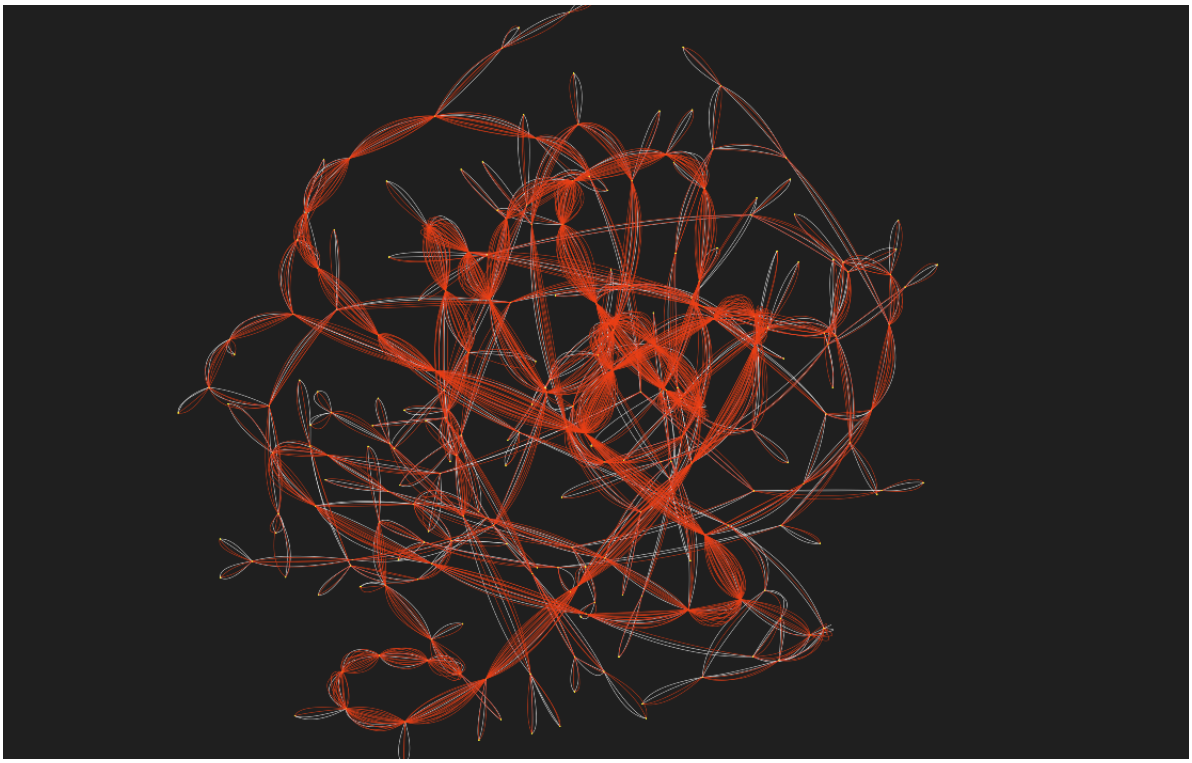
Eléments de recherche opérationnelle

Vincent Saunier **David Chane Yock Nam**
Ahmad Harkous **Félix Huang**

5 Juin 2023



Introduction

Chaque hiver, la ville de Montréal fait face à de nombreux épisodes neigeux. Lors des pires tempêtes, il peut neiger jusqu'à plus d'un mètre. La neige rendant la circulation des voitures, bus et piétons dangereuse, il est donc primordial de déneiger régulièrement les rues. Face à cela, la mairie met à disposition plusieurs engins déneigeurs et des drones permettant de repérer les zones à déneiger.

Problématique

Comment réussir à déneiger la ville en optimisant les déplacements des véhicules de déneigement? Ou encore:
"Comment réduire au minimum leur parcours, tout en maximisant leur zone d'efficacité ?"
L'argent étant le nerf de la guerre, l'objectif est de pouvoir satisfaire la demande (déneigement) , tout en réduisant les coûts (matériel, carburant . . .).

Données & Contraintes

Pour ce qui est des données, la ville possède un arsenal mécanique afin de faire face à l'enneigement. Elle met à disposition des véhicules ainsi qu'un drone aux coûts fixes.
Super Drone :
— Coût fixe : 100 €/jour
— Coût kilométrique : 0.01 €/km
Véhicules :
— Coût fixe : type I : 500€/jour, type II : 800 €/jour
— Coût kilométrique : type I : 1.1 €/km, type II : 1.3 €/km
— Coût horaire les 8 premières heures : type I : 1.1 €/h, type II : 1.3 €/h
— Coût horaire au delà des 8 premières heures : type I : 1.3 €/h, type II : 1.5 €/h
— Vitesse moyenne : type I : 10 km/h, type II : 20 km/h
Il est indiqué aussi que la municipalité aimerait investir dans les véhicules de type II.
On suppose les contraintes suivantes.
-Le drone sert à cartographier les rues. Il doit donc les toutes parcourir au moins une fois
-Les rues sont à double sens. Un camion ne déneige qu'un sens à la fois.
-Contraintes de coût d'utilisation

Choix de Modélisation

Initialement nous avons choisi de modéliser la ville comme un graphe qui a pour arrêtes les routes, et pour noeuds les intersections.
Pour faire cela nous avons utilisé Python aisni que ses bibliothèques.

- Osmnx: extraire le graphe à partir de la carte de la ville
- Networkx: manipulation /opérations sur graphe
- Matplotlib: afficher les jeux de données

1 Modélisation: Cas du Drone

Le problème du drone est semblable au problème du postier chinois. En effet, tout comme le cas du Problème des sept ponts de Königsberg, l'objectif est de passer par tous les ponts (ici rues de la ville) une fois. Nous allons résoudre ce problème en deux étapes :
1-Rendre le graphe Eulérien
2-Effectuer un circuit sur le graphe.

1.1 Transformation du graphe en un graphe eulérien avec des nœuds pairs

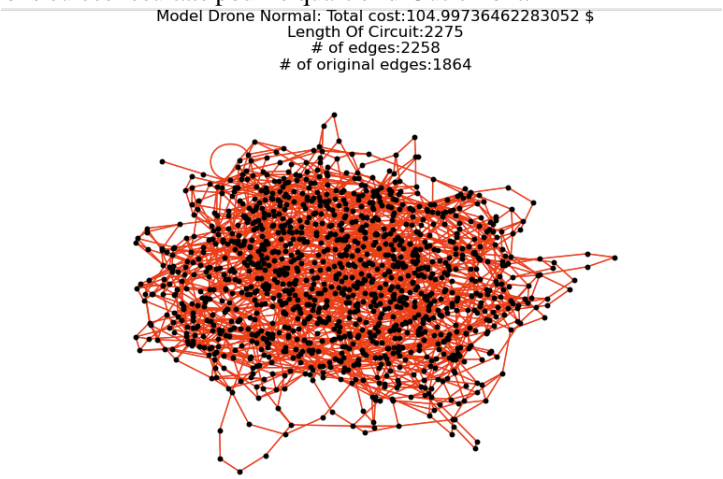
Dans un premier temps, nous avons essayé d'utiliser la fonction nx.eulerize pour transformer le graphe en un graphe eulérien, ceci facilitant ainsi la tâche. Cependant, cette fonction ne nous a pas donné le résultat souhaité avec des nœuds de degré impair. Nous avons donc décidé de mettre en œuvre les étapes manuellement.
A-Identification des nœuds de degré impair :
La première étape consiste à identifier les nœuds du graphe d'origine ayant un degré impair. Un nœud de degré impair signifie qu'il a un nombre impair d'arêtes connectées.
B-Calcul des chemins les plus courts entre les nœuds de degré impair :
Pour chaque paire de nœuds de degré impair, on calcule la distance du plus court chemin entre eux en utilisant **l'algorithme de Dijkstra**. On stocke aussi les distances obtenues dans un dictionnaire avec pour clés les paires de nœuds et valeurs les distances du plus court chemin.
C-Génération d'un graphe complet :
Ensuite, on créé un graphe complet à partir des paires de nœuds de degré impair et de leur distance. La fonction utilisée pour créer ce graphe complet parcourt le dictionnaire généré a l'étape précédente et ajoute des arêtes au graphe, où chaque arête représente une clé du dictionnaire et a pour poids la valeur reliée à cette clé.
Par exemple, si nous avons trois nœuds de degré impair A, B et C, avec les distances du plus court chemin suivantes : distance entre A et B : 5, distance entre A et C : 8, distance entre B et C : 3. Le graphe complet aura les arêtes AB (poids 5), AC (poids 8) et BC (poids 3).
D-Correspondance de poids minimal :
Une fois que le graphe complet est créé, l'algorithme de correspondance de poids minimal est appliqué, il renvoie un dictionaire où les clés sont les paires de nœuds et les valeurs sont les poids. L'objectif est de trouver une correspondance parmi les arêtes du graphe complet qui minimise le poids total. Dans notre code, nous utilisons la fonction *maxweightmatching* de la bibliothèque Python NetworkX pour effectuer cette correspondance. Le code convertit ensuite le dictionnaire en une liste de tuples. Cela permet de garantir l'unicité et éliminer les appariements en double. La liste résultante contient les paires de nœuds uniques dont la somme cumulée donne la distance minimale possible.
E-Finalisation du graphe eulérien :
Nous ajoutons la liste des tuples obtenue à l'étape précédente au graphe original, créant ainsi un graphe augmenté. Maintenant, nous avons un graphe eulérien qui représente le circuit optimal pour le drone qui parcourt toutes les routes de la ville (Toutes les images concernant cette partie sont disponibles dans l'annexe sur les images à la fin du rapport).

1.2 Circuit du Drone

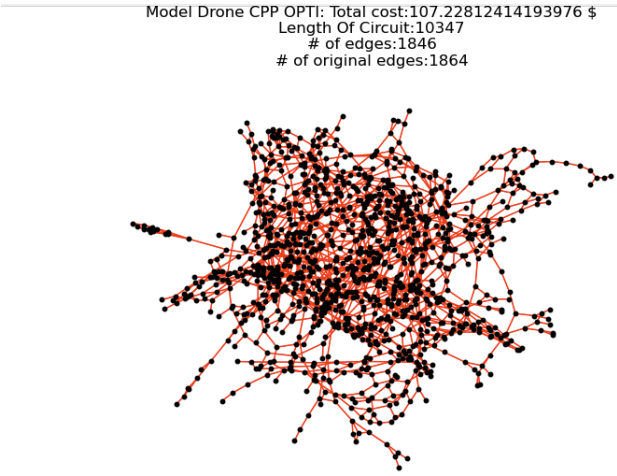
A-Circuit naif eulérien et ses limitations
Commençons par la solution simple mais incomplète.

$$circuit\_eulerien_{naif} = list(nx.eulerian\_circuit(G\_eulerien)) \quad (1)$$

Un circuit eulérien, en particulier, est une sorte de parcours spécial qui visite chaque arête du graph eulerien exactement une fois et revient à son point de départ.
Cependant, tous les graphes ne peuvent pas avoir naturellement un circuit eulérien. Les circuits eulériens existent uniquement dans les graphes où chaque nœud a un degré pair.
Cette approche n'est pas adaptée dans notre cas, car en rendant le graphique de base eulérien, il est probable que nous ajoutions des arêtes qui n'existaient pas dans le graphe original (des rues virtuelles).
Pour bien illustrer le parcours du drone, on a colorié les arêtes parcourues en rouge,les noeuds en noir, et nous avons eu ces résultats pour le quartier d'Outremont:



Nous remarquons que la fonction nx.eulerian_circuit n'est pas parfaite. Il peut y avoir une différence de taille entre le circuit eulérien et le graphe lui-même (Par exemple 17 arêtes de difference pour l'image en dessus).
B-Circuit optimisé , réel
Pour obtenir le circuit réaliste, nous transformons simplement le circuit naïf qui inclut des arêtes inexistantes dans le graphe original en un circuit eulérien en n'utilisant que les arêtes présentes dans le graphe original.
Nous itérons à travers chaque arête du circuit eulérien naïf. Chaque fois que nous rencontrons une arête qui n'existe pas dans le graphe original, nous la remplaçons par la séquence d'arêtes constituant le plus court chemin entre ses nœuds en utilisant le graphe original.



1.3 Coût du Drone

Pour calculer le coût réel du drone (et même des déneigeuses ensuite), nous avons donc utilisé une autre approche avec la fonction geodesic du module geopy, qui permet de calculer la distance entre deux noeud depuis leur coordonnée, et donc la longueur de l'arrête entre eux.
En utilisant la solution naïve, le coût total du parcours du quartier d'Outremont en une journée avec le drone est de 104,9973 euros (comprenant le tarif fixe et le tarif horaire/km de la drone). Cependant, avec la solution optimale et plus réaliste, le coût est légèrement supérieur, soit 107,2281 euros, comme indiqué dans l'image ci-dessus. Cela s'explique par le fait que nous modifions le circuit eulérien initial pour obtenir le circuit réel, ce qui entraîne l'augmentation du nombre d'arêtes du graphe et donc une légère augmentation des coûts. Une estimation du coût pour les 5 quartiers serait alors de 136 euros (7.2281*5).

2 Modélisation: Cas des déneigeuses

2.1 Coût des déneigeuses

Le problème du déneigement est semblable à celui du parcours du drone.
La différence est que le sens de circulation doit être pris en compte.
En terme de modélisation on a donc toujours les intersections comme noeuds, mais ceux-ci sont reliés par de arcs entrants et sortants.
On a donc un graphe dirigé, valué. L'enjeu ici est de transformer le graphe de la ville en cette modélisation. Pour cela on procède comme suit:

- rendre le graphe eulérien, si il ne l'est pas ajouter des "dummy nodes"
- rendre le graphe dirigé

Une fois ce processus terminé, on peut à la manière du drone (même formule algébrique de coût) déterminer un montant.
Nous avons donc obtenu une première solution. Néanmoins, elle n'est pas efficace.
Une solution qu'on a trouvé était de segmenter le graphe en plusieurs sous-graphes.
Une fois le graphe découpé en sous-ensemble on peut lancer plusieurs machines, une sur chaque secteur.
On peut maintenant essayer plusieurs découpage des quartiers et comparer les coûts.
Cela ne nous amnène pas à la solution optimale, mais on peut itérativement s'en rapprocher.

3 Conclusion

Pour conclure ce rapport et par la même occasion le projet, on peut dire que nous avons réussi à trouver une solution pour déneiger la ville, bien qu'il y a matière à l'améliorer. Nous avons manqué d'organisation. Nous aurions pu, sur la fin, optimiser correctement le parcours.

4 Liens Annexes

4.1 Lien vers repo

Le lien github vers la racine du projet:
https://github.com/ERO1-TEAM-8/submission.
Celui-ci comprend :

1. Un **README** qui explique toute la démarche d'installation et de manipulation du projet
2. circuit_drone_comp/ et circuit_snow_removal/ sont des dossiers d'images d'illustrations (images + gif).
3. Le code réside dans les dossiers drone/ et snowremoval/.

4.2 Références

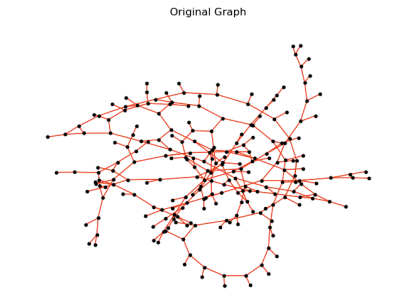
Ces articles ne sont pas cités dans ce rapport mais nous ont grandement aidés pour le projet. On se permet donc de les citer.

1. Leila Hajibabai and Yanfeng Ouyang
   Dynamic Snow Plow Fleet Management under Uncertain Demand and Service Disruption
   https://ieeexplore.ieee.org/ielaam/6979/7553592/7412707-aam.pdf
2. Joris Kinable, Willem-Jan van Hoeve & Stephen F.Smith
   Snow plow route optimization: A constraint programming approach
   https://www.tandfonline.com/doi/full/10.1080/24725854.2020.1831713

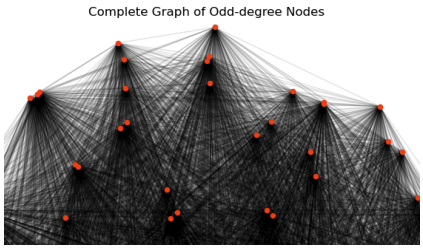
5 Sources

- Documentation Networkx:
  https://networkx.org/documentation/stable/reference/index.html
- Documentation OSMNX: https://osmnx.readthedocs.io/en/stable/
- Problème du chemin eulérien
  https://fr.wikipedia.org/wiki/Graphe_eulÃľrien

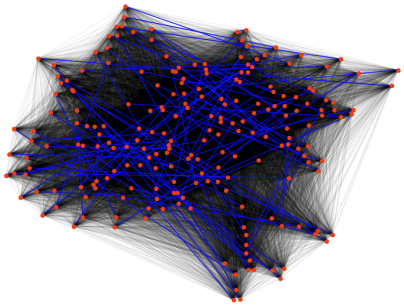
6 Images: Etapes : Rendre le graphe Eulerien sur la ville de Leynhac, France



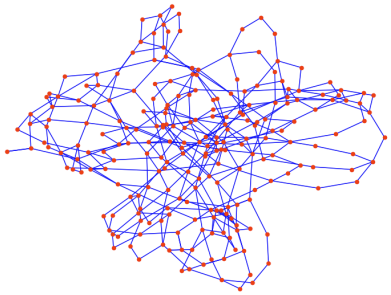
(a) 1-Graph d'origine



(b) 2-Graphe complet avec les noeuds impaires



(c) 3-Min weight sur le graphe complet



(d) 4- Graphe augmenté eulerien

Figure 1: Etapes de transformations du graphe.

nx.eulerise incorrect :