Rapport Éléments de Recherche Opérationnelle

Ferdinand Mom, Marius Dubosc, Nathan Cabasso, Tony Heng

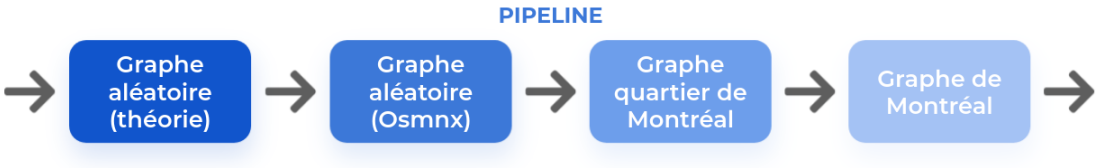
21 juin 2021

1 Introduction

L'enjeu de ce devoir est de concevoir un algorithme capable de parcourir les rues de Montréal, afin de pouvoir les analyser et les déneiger. Le développement d'un tel algorithme fait inévitablement face à un compromis entre efficacité (minimisation de la distance parcourue), et performance (complexité de l'algorithme).
Ce rapport expose nos approches itératives du problème, chacune exploitant les résultats précédents afin d'améliorer le compromis. Ce travail a été fait dans un premier temps sur des graphes de petite taille de notre conception, afin de pouvoir exploiter des algorithmes peu performants. Une fois satisfaits de l'équilibre entre efficacité et performance, nous avons pu appliquer notre algorithme aux cartes de Montréal, en commençant par des quartiers. Ce processus fut appliqué pour le drone, sur des graphes non orientés, puis pour les déneigeuses, sur des graphes orientés.

2 Analyse du terrain - Drone

Dans le cas du drone, la carte peut être représentée sous la forme d'un graphe non orienté, car ce dernier peut se déplacer librement au-dessus des routes sans suivre le sens de circulation. Parcourir le graphe de manière optimale se traduit par la recherche du chemin le plus court traversant l'ensemble du graphe. Dans un graphe non orienté dit eulérien (tous les sommets sont de degré pair), cela correspond à un cycle eulérien (chemin empruntant toutes les arêtes une seule fois). L'idée est donc de rendre artificiellement le graphe eulérien, en "ajoutant" des arêtes aux sommets de degré impair. Ces arêtes ajoutées correspondent en réalité à des routes existantes que l'on s'autorise à parcourir plusieurs fois. Une fois le graphe rendu eulérien, il suffit de chercher son cycle eulérien. Notre plan d'action est de procéder itérativement en partant de graphes de petite taille pour arriver sur des graphes beaucoup plus conséquents.



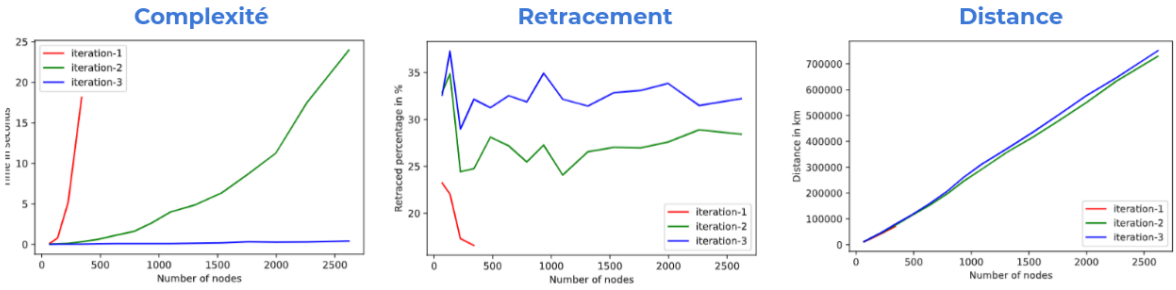
Avant d'aboutir à une solution convenable, nous avons essayé plusieurs approches. L'idée générale est d'***ajouter*** des arêtes entre les sommets de degré impair pour rendre le graphe eulérien. Afin de trouver la combinaison optimale de ces arêtes, on construit un graphe complet avec les sommets de degré impair, pondéré par la distance qui sépare chacun de ces sommets dans le graphe originel, et l'on cherche le *perfect matching* minimisant le poids total. Si cet algorithme nous assure une solution optimale, le temps d'exécution sur un quartier de Montréal est bien trop important (complexité $O(n^3)$), de même que l'espace mémoire requis pour stocker l'ensemble des combinaisons.

Pour pallier cela, on se contente alors de sélectionner, pour chaque sommet de degré impair, le sommet le plus proche parmi ceux non sélectionnés. Si cette approche ne permet pas d'obtenir le chemin optimal, elle permet néanmoins de réduire drastiquement la complexité (($O(n^2)$), tout en fournissant un chemin relativement court. Cela reste cependant long à l'échelle de Montréal.

La solution pour laquelle nous avons alors opté est de sélectionner, pour chaque sommet de degré impair, le premier sommet non sélectionné rencontré dans un parcours en largeur. Cette approche, qui approfondit le principe de la deuxième étape, permet former des paires de sommets dont on soupçonne qu'ils sont proches, et se dispense ainsi de calculer les distances séparant les sommets. Cette grande réduction de complexité permet alors de travailler sur des graphes de tailles variées même si le circuit obtenu n'est pas optimal. Voici une comparaison des différentes approches :

MÉTRIQUES POUR CHAQUE ITÉRATION

- La distance parcourue - en km
- Le taux de retracement - en %
- La complexité - en sec



3 Parcours du terrain - Déneigeuse

Pour la déneigeuse, le problème est similaire à celui du drone, à l'exception que l'on travaille sur un graphe orienté. Ainsi, la notion de graphe eulérien change : pour qu'il soit considéré comme eulérien, il est nécessaire que chaque sommet du graphe ait le même nombre d'arêtes sortantes et entrantes, et que le graphe soit fortement connexe.

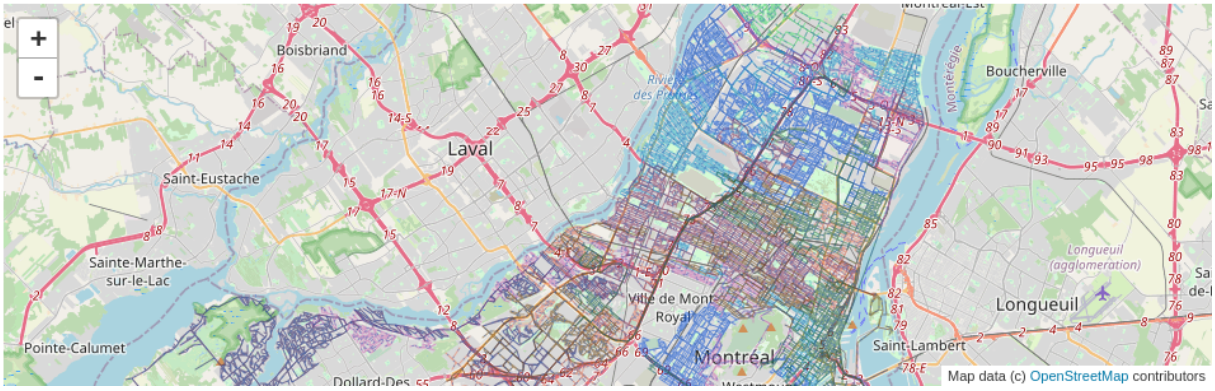
Afin de transformer le graphe de Montréal en graphe eulérien, nous travaillons sur la plus grosse composante fortement connexe afin d'exclure les zones non accessibles de la ville du processus de déneigement. Ensuite nous calculons la liste des sommets nécessitant d'ajouter des arêtes entrantes ou sortantes ainsi que leur nombre, que nous séparons dans deux ensembles différents.

Il existe ensuite plusieurs stratégies pour associer chaque sommet entrant à un sommet sortant. Du fait de notre expérience avec le drone, nous avons commencé par chercher une solution non optimale, mais rapide, en utilisant un parcours en largeur. Nous pensions à l'origine que la complexité $O(n^3)$ du *perfect matching* serait un problème.

Finalement, nous nous sommes rendu compte qu'il était possible d'implémenter un algorithme de *perfect matching* avec une complexité $O(n^2 log(n))$. L'idée est de former un graphe biparti des sommets selon leur déficit d'arêtes (entrantes ou sortantes), puis de former une matrice de coûts (distances séparant chaque sommet), et d'y chercher la plus petite somme. Il ne nous restait plus qu'à optimiser le calcul des distances entre les sommets entrants et sortants, ainsi que l'usage de la mémoire. Nous avons donc fait en sorte de calculer et conserver uniquement les informations nécessaires à la construction de la matrice de coûts.

Afin de profiter au maximum des performances de la machine sur laquelle notre algorithme tourne, nous avons décidé de diviser le travail de calcul de la matrice de coûts (*single source Dijkstra*) sur plusieurs processus. Cela permet de diviser le temps d'exécution par le nombre de cœurs du processeur de la machine sur lequel l'algorithme tourne.

Au-lieu de diviser la carte de Montréal, nous avons décidé de décomposer le circuit d'une seule déneigeuse sur tout Montréal pour les attribuer à d'autres déneigeuses. Cela permet d'éviter le problème des routes non déneigées lors de la division de la carte de Montréal.



Circuit généré sur Montréal avec 10 déneigeuses

4 Proposition d'un modèle de côut

Nous proposons un modèle de coût qui se base sur le nombre de déneigeuses voulues pour une distance à déneiger. Avec ces critères, nous pouvons estimer la durée du service de déneigement (plus il y a de déneigeuses, moins long sera le processus de déneigement), et proposer une offre.

Variables :
- n : nombre de déneigeuses
- v : vitesse d'une déneigeuse
- t : temps par déneigeuse
- t_{tot} : temps total
- d_{tot} : distance totale
- n_{day} : nombre de jours
- p_{rent} : coût de location d'une déneigeuse
- p_{day} : prix pour un jour
- p_{tot} : prix total du service

Calcul du modèle de coût :
- $t = d_{tot}/v$
- $t_{tot} = t/n$
- $n_{day} = t_{tot}/24$
- $p_{day} = p_{rent} * n$
- $p_{tot} = p_{day} * n_{day}$

5 Limites et améliorations

Nous avons vu à chaque itération les avantages, mais aussi les désavantages : temps d'exécutions élevé, grande complexité spatiale, résultats non convenables. La plupart des désavantages ont été corrigés aux itérations suivantes, mais certains d'entre eux persistent.

5.1 Limites

Nous avons besoin d'un graphe fortement connexe pour pouvoir calculer le chemin optimal, de ce fait, certaines routes ne sont pas prises en compte et les impasses sont retirées.

5.2 Améliorations

Pour faire face à cette limite, nous pouvons diviser Montréal en amont pour maximiser le nombre de routes prises en compte.