APP ERO

Casse Germain, Gosselin Colas, Huynh Kévin, Kaci Quentin

Juillet 2020

Table des matières

1 Introduction

Pour ce projet, notre groupe était composé au départ de 5 personnes mais dû au fait que l'un d'entre nous ne devait pas rattraper le module de maths (redoublant), nous avons fait le projet à 4: Casse Germain, Gosselin Colas, Huynh Kévin et Kaci Quentin.

2 Les solutions utilisées pour le projet

2.1 Cas général

Dans le cas général, trouver le chemin le plus optimal en passant par toutes les arêtes revient à chercher un cycle eulérien dans le graphe.
Pour le cas d'un graphe déjà eulérien de base, il suffit juste de chercher un cycle: nous avons utilisé l'algorithme de Hierholzer.
Pour le cas contraire, il faut rendre le graphe eulérien tout en minimisant le coût du cycle eulérien parcourant le graphe. Afin de calculer le chemin minimal entre tous les noeuds non équilibrés (cela dépend du cas orienté ou non-orienté) du graphe nous utilisons l'algorithme de Floyd-Warshall .

2.2 Le drone

Afin de résoudre ce problème pour le drone, nous nous plaçons dans un graphe non orienté.
Un graphe non orienté est eulérien si tous ses noeuds sont de dégré pair.
Notre objectif est donc de transformer tous les noeuds de dégré impair en degré pair en les couplant deux à deux entre eux avec un coût minimal (en dupliquant des arêtes déjà existantes).
Avant d'effectuer cette étape, nous avons prélablement doublé les arêtes de tous les noeuds possédant une seule arête (transformer les noeuds de degré impair en pair) puisque ces derniers correspondent à des impasses dans lesquelles il faut obligatoirement pouvoir ressortir. Cette méthode permet ainsi d'optimiser notre algorithme de couplage minimal puisqu'on réduit le nombre de noeuds de degré impair.

La solution est d'utiliser le "Edmond's algorithm" afin de trouver le couplage minimal dans un graphe quelconque. Cependant nous ne sommes pas parvenu à implémenter cet algorithme; celui-ci étant complexe. Nous avons donc utiliser la bibliothèque networkx afin de pouvoir implémenter la première vision que nous avions de l'algorithme. Ce processus est executé dans la fonction optimized_solve du module snowymontreal.

Notre première vraie implémentation consistait à calculer tous les couplages possibles des noeuds de dégré impair (brute force) puis de retenir le couplage minimal. Cependant, cette solution était beaucoup trop lente à l'exécution et en consommation mémoire. (de l'ordre du n!)

Après de nombreuses recherches, nous avons trouvé un moyen d'approximer la solution du problème en utilisant les arbres couvrants minimaux. Notre solution finale est donc de calculer l'arbre couvrant minimal à partir du graphe complet de tous les noeuds de degré impair. Puis de trouver un couplage maximal dans celui-ci. Cependant il arrive que des noeuds soient isolés dans le couplage maximal de l'arbre couvrant minimal. Pour cela, il faut alors réitérer le processus avec les noeuds isolés restants jusqu'à ne plus avoir de noeuds de degré impair à traiter. Cette solution ne donne donc pas le couplage minimal mais une approximation de celui-ci. De plus, cette méthode est beaucoup plus optimisée que la méthode brute force présentée précédemment. Une fois que nous avons obtenu le couplage minimal, nous devons rajouter les arêtes qui sont incluses dans le plus court chemin entre eux. Le résultat obtenu dans la fonction solve ne correspond par conséquent pas au chemin minimal mais à une approximation de celui-ci.

2.3 La déneigeuse

Contrairement au drone, nous nous plaçons dans ce cas dans un graphe orienté. Le problème est différent du fait qu'un graphe orienté est eulérien si chaque noeud a le même degré entrant que sortant.

Notre solution est donc de relier tous les noeuds nécessitant des arêtes entrantes supplémentaires aux noeuds nécessitant des arêtes sortantes supplémentaires. Nous devons donc créer un graphe biparti en mettant d'un côté les noeuds nécessitant des arêtes entrantes et de l'autre ceux nécessitant des arêtes sortantes. À partir de ce graphe, nous devons trouver le couplage minimal. Ce problème est plus simple à résoudre que dans le cas d'un graphe quelconque.

Dans un premier temps, nous avons utilisé, comme pour le cas du drone, une méthode brute force afin de trouver le couplage minimum. Cependant, nous avons trouvé le "Hungarian algorithm" qui permet de trouver le couplage minimal dans un graphe biparti de manière beaucoup plus optimisée. Une fois que nous avons obtenu le couplage minimal, nous devons enfin rajouter les arêtes qui sont incluses dans le plus court chemin entre eux.

2.4 L'interfaçage avec OSMnx

Afin de pouvoir utiliser notre moteur sur une grande échelle notamment pour notre cas dans la ville de Montréal, nous avons utilisé les fonctions optimisées de la bibliothèque networkx afin de résoudre notre problème.
Nous avons plusieurs moteurs pour notre solution que nous avons détaillés dans le README.
De plus, dans notre produit final, nous permettons à l'utilisateur de rentrer les coordonnées de départ de la déneigeuse ou du drone qu'il veut mettre en marche et nous affichons par la suite le trajet minimal qui parcourt l'ensemble des routes de la zone sélectionnée.

3 Les potentielles améliorations

Dans le cadre de la déneigeuse, nous aurions pu utiliser des heuristiques afin d'approximer le résultat et donc gagner en complexité sur l'exécution de l'algorithme Dans le cadre du drone, nous avons déjà détaillé la méthode permettant de trouver le chemin minimum exact tout en restant sur une complexité acceptable: $O(n^3)$. Cependant, il existe également des méthodes heuristiques pour améliorer cet algorithme (notamment celle de la bibliothèque networkx).

4 Positionnement du point de vue d'une société

Etant une société militant pour l'environnement, et afin de minimiser la pollution produite par nos déneigeuses, nous avons opté pour cette solution. En effet, cette dernière utilise tout d'abord un drone qui permet de cadrer toute une zone et nous permet par la suite, grâce à ce programme, de pouvoir trouver le chemin le plus optimal afin de déneiger une zone tout en polluant le moins. Certes, les drones demandent un coût mais à long terme une rentabilité sera visible tout en protégeant au maximum l'espace dans lequel nous vivons.

5 Conclusion

Pour conclure, nous avons rapidement identifié ce problème comme étant le Chinese Postman Problem, ce qui nous a aiguillé sur les algorithmes et les méthodes à utiliser. Nous avons dans un premier temps utilisé des méthodes "brute force" pour nos algorithmes afin de savoir si notre algorithme global allait dans la bonne direction. Nous avons par la suite optimisé nos algorithmes afin de monter en charge sur les graphes que nous pouvions prendre en entrée. Pour finir, nos algorithmes n'étant tout de même pas assez efficaces afin de prendre en charge des graphes comme celui de la ville de Montreal, nous avons dû utiliser la bibliothèque netwokx qui possède des algorithmes utilisant des heuristiques pour résoudre ce problème efficacement et pouvoir proposer un produit final cohérent.

References

[1] Optimization on Chinese Postman Problem
http://brooksandrew.github.io/simpleblog/articles/intro-to-graph-optimization-solving-cpp/

[2] Directed Chinese Postman Problem,
https://www-m9.ma.tum.de/graph-algorithms/directed-chinese-postman/index_ en.html