EPITA

Optimisation hivernale
Déneiger Montréal

Éléments de recherche opérationnelle

GUENET Ilan
DIA Cheick- tidiane
NARAINEN Kenz
CLAUDEL Antoine
LASSARTESSE Arnaud

---

17 Juin 2020

EPITA

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

1 Qui sommes nous ?

1.1 Équipe

1.1.1 Membres

Kenz	**Arnaud**	**Cheick**	**Ilan**	**Antoine**
Communication Marketing	Développeur	Développeur	Chef de projet	Développeur

1.1.2 Les tâches

Tâches	**Kenz**	**Arnaud**	**Cheick**	**Ilan**	**Antoine**
Chemin du drone		R	R		
Chemin des déneigeuses	R			R	
Les tests	R	R	R	R	R
Autres fonctionnalités				R	R

R : Responsable

1.1.3 Origine du logo

Nous avons décidé d'élaborer un logo assez explicite avec tous les éléments de notre projet sur celui-ci. Le fond est bien évidemment le ville de Montréal sous la neige avec une déneigeuse et son drone. "DrowNe Plow" est un petit jeu de mots avec *drone* et *snow plow*.

1.2 Objectifs

Nous sommes une société de services qui vend des solutions à des entreprises publiques ou privées. Nous avons reçu une demande du conseil municipal de la ville de Montréal pour réaliser une étude dont le but est de minimiser le coût des opérations de déblaiement. Nous cherchons donc à trouver le moyen de déneiger toutes les voies de Montréal à moindre coût, c'est-à-dire en minimisant les trajets des déneigeuses. Cette étude pourrait être envisagée par la ville de Montréal si elle se trouve être efficace.

1.3 Méthode de travail

Nous avons développé la solution itérativement. Une itération correspond à une fonctionnalité. Afin qu'une itération soit validée, il fallait que le programme passe tous les tests : passer les nouveaux tests pour montrer que la nouvelle fonctionnalité répond correctement aux besoins et qu'elle en plus ne casse pas les anciennes itérations. Grâce à un nombre conséquent de tests, il a été facile d'être au courant de tout problème avant de livrer la solution. C'est aussi un gage de qualité.

2 Choix pour le drone

Pour le drone, on utilise un graphe non orienté, étant donné qu'un drone peut se déplacer librement au dessus des routes. La méthode que nous avons utilisé pour rendre le parcours du drone minimal est la suivante. La ville est représentée sous forme de graphe, il est donc possible de la parcourir comme tel. Nous avons utilisé un algorithme pour trouver un cycle eulérien dans le graphe de la ville, permettant un parcours de toutes les arrêtes du graphe. Les arrêtes représentant les routes, le drone parcourera ainsi toute la ville.

Cependant, nous nous sommes rendu compte que cet algorithme ne fonctionnait uniquement pour les graphes eulériens et que les villes n'étaient pas forcément des graphes eulériens. Nous nous sommes donc tournés vers un nouvel algorithme : celui du postier Chinois. Celui-ci permet la transformation d'un graphe non-eulérien en un graphe eulérien, en rajoutant de nouvelles arrêtes au graphe. Les arrêtes ajoutés sont des arrêtes déjà existantes dans le graphe. Ainsi, la sémantique du graphe n'est pas modifiée. Cette technique permet d'assurer que la recherche d'un cycle eulérien ait une solution dans le nouveau graphe. Le drone peut à présent parcourir la plupart des graphes de villes.

```
city: Arcachon, FR, oriented: False
number of vertices: 662
number of edges: 979
time: 0.00001s
solved: True
```

FIGURE 1 – Résultat de l'algorithme pour la ville d'Arcachon (France)

3 Choix pour la déneigeuse

Dans le cas de la déneigeuse, le graphe d'entrée est orienté. En effet, il est de bon sens de dire que les déneigeuses respectent les sens de circulation des routes. De plus, le graphe doit être fortement connexe afin d'y trouver un chemin répondant au problème. Ce chemin doit comme dans le cas du drone passer par chaque arc au moins une fois.

Comment avons nous résolu ce problème ? Nous nous sommes basés une nouvelle fois sur un le problème du postier Chinois mais cette fois-ci dans le cas d'un graphe orienté. L'algorithme utilisé est alors différent que celui pour le drone. On va chercher tous les noeuds qu'on pourrait lié afin de rendre le graphe eulérien. Il faut modifier les degrés de certains noeuds. Une fois les paires de noeuds trouvées, on va rechercher les chemins les moins coûteux entre ces paires. Enfin, il faut ajouter à ce graphe tous ces nouveaux arcs. Finalement, notre graphe est devenu eulérien, il ne suffit plus qu'à y chercher un cycle eulérien.

Nous avons choisi d'utiliser cette algorithme parce qu'il respecte la contrainte du respect de la sémantique du graphe. De plus, il permet de résoudre le problème dans tous les cas à une vitesse acceptable.

4 Limites et améliorations

4.1 Limites

Notre algorithme pour déterminer le trajet minimal du drone lors du survol du réseau routier ne supporte pas les graphes trop gros, ou trop complexes. Par exemple, il ne fonctionne pas pour les grandes villes comme Montréal. Cependant, il fonctionne bien pour de plus petites villes.

De plus, les chemins trouvés par nos algorithmes n'offrent malheureusement pas la meilleur solution. En effet, on pourrait à partir d'une autre technique d'eulerization de graphe obtenir des chemins qui coûtent en distance moins cher. Cependant, la performance en sera moindre. Il faudrait trouver un compromis entre performance et efficacité.

Marker	Method	Line	Hits	Avg Time	Runtime	Percent
1 vertices, 0 edges	solve_undirected	bench.py:34	1	0.00002	0.00002	0.00
25 vertices, 184 edges	solve_undirected	bench.py:34	1	0.00360	0.00360	0.00
50 vertices, 676 edges	solve_undirected	bench.py:34	1	0.05727	0.05727	0.07
75 vertices, 1492 edges	solve_undirected	bench.py:34	1	0.27124	0.27124	0.33
100 vertices, 2576 edges	solve_undirected	bench.py:34	1	0.94181	0.94181	1.16
125 vertices, 3967 edges	solve_undirected	bench.py:34	1	2.10396	2.10396	2.59
150 vertices, 5790 edges	solve_undirected	bench.py:34	1	4.63038	4.63038	5.70
175 vertices, 7818 edges	solve_undirected	bench.py:34	1	7.24568	7.24568	8.92
200 vertices, 10156 edges	solve_undirected	bench.py:34	1	14.05275	14.05275	17.30
225 vertices, 12784 edges	solve_undirected	bench.py:34	1	19.53577	19.53577	24.05
250 vertices, 15873 edges	solve_undirected	bench.py:34	1	32.39882	32.39882	39.88

Total runtime: 81.24133

FIGURE 2 – Benchmark pour le drone

Marker	Method	Line	Hits	Avg Time	Runtime	Percent
1 vertices, 0 edges	solve_directed	bench.py:19	1	0.00005	0.00005	0.00
25 vertices, 388 edges	solve_directed	bench.py:19	1	0.00769	0.00769	0.01
50 vertices, 1646 edges	solve_directed	bench.py:19	1	0.08489	0.08489	0.08
75 vertices, 3288 edges	solve_directed	bench.py:19	1	0.35587	0.35587	0.33
100 vertices, 6342 edges	solve_directed	bench.py:19	1	1.17974	1.17974	1.09
125 vertices, 9592 edges	solve_directed	bench.py:19	1	2.85901	2.85901	2.63
150 vertices, 13658 edges	solve_directed	bench.py:19	1	5.69021	5.69021	5.24
175 vertices, 19004 edges	solve_directed	bench.py:19	1	11.01189	11.01189	10.13
200 vertices, 23796 edges	solve_directed	bench.py:19	1	17.82253	17.82253	16.40
225 vertices, 30586 edges	solve_directed	bench.py:19	1	28.94423	28.94423	26.63
250 vertices, 37322 edges	solve_directed	bench.py:19	1	40.72117	40.72117	37.47

Total runtime: 108.67731

FIGURE 3 – Benchmark pour la déneigeuse

4.2 Améliorations

Afin d'améliorer notre solution, nous pourrions utiliser la bibliothèque NetworkX . C'est une bibliothèque Python spécialisée dans l'étude des graphes et des réseaux. Celle-ci propose plusieurs fonctions travaillées et optimisées pour la transformation d'un graphe non eulérien en graphe eulérien notamment, ce qui permettrait à nos algorithmes des résultats plus rapides, plus précis et supportant toute sorte de graphe s'apparentant à une ville.