Synthèse de projet

Suite à la découverte du problème, nous nous sommes rendus compte assez facilement que nous étions devant une problématique de graphe. Face à ce genre de situation, de nombreux algorithmes existent d'ores et déjà et nous pouvons choisir celui qui sera le plus adapté à notre situation.

Afin de bien pouvoir expliquer notre point de vue ainsi que la solution mise en place, il nous semble important de dissocier le cas théorique du cas d'application réel.

I. Etude du cas théorique

A. Examen complet du réseau routier

Après étude du sujet, nous remarquons bien que nous sommes confrontés à un problème à caractère eulérien. Plus précisément, ce problème peut être assimilé à celui du postier chinois (**Chinese Postman Problem**) dans le cadre d'un graphe non-orienté. Dans ce genre de cas, nous devons tout d'abord déterminer si le graphe est eulérien. S'il ne l'est pas, c'est à nous de le transformer afin qu'il le devienne. Au cours de nos recherches, nous avons compris que pour rendre eulérien notre graphe, nous devions utiliser l'algorithme de Dijkstra afin de trouver les paires de nœuds impairs les plus optimales. Cela permet de transformer tous les nœuds impairs en nœuds pairs.

Néanmoins, nous avons compris que rendre un graphe eulérien par cette technique était au-dessus de nos capacités. Ainsi, dans le cas théorique, nous sommes partis du principe que les graphes que nous étudions sont tous eulériens.

Maintenant que nous savons que les graphes pris à l'étude vérifient cette condition, il nous suffit de trouver le cycle qui nous permet de visiter toutes leurs arêtes. Pour ce faire, nous avons utilisé l'une des fonctions que nous avons étudiées au cours de l'année en *théorie des graphes* : **find_eulerian_path**.

En sortie de celle-ci, nous obtenons la liste de l'enchaînement des nœuds qui correspond au plus court chemin que nous recherchons.

B. Trajet minimal des appareils de déblaiement

Avant d'attaquer cette partie il est important de rappeler que le problème change légèrement de nature par rapport au cas précédent. En effet, nous nous rendons bien compte, qu'ici, nous allons devoir manipuler des graphes orientés. Néanmoins le but final reste le même, à savoir, trouver le chemin le plus court qui parcourra l'entièreté de notre graphe. Encore en *théorie des graphes*, nous avons étudié plusieurs algorithmes autres que Dijkstra (Bellman-Ford, Floyd-Warshall, ...).

Nous nous sommes penchés dans un premier temps vers Dijkstra car c'est l'algorithme avec lequel nous étions le plus familier. Malheureusement, nous n'avons pas su l'utiliser efficacement pour résoudre la partie théorique du problème.

De ce fait, nous avons préféré nous focaliser davantage sur la création d'une solution fournissant des résultats quantifiables en utilisant des méthodes de recherche plus naïves. C'est pour cela que dans cette partie, vous ne trouverez pas de code apportant une réponse théorique au problème dans le cas de graphes orientés.

II. Etude du cas réel

A. Examen complet du réseau routier

Pour analyser le réseau routier d'une ville ou d'un quartier nous allons donc utiliser un drone. Pour trouver le chemin à parcourir par ce drone nous avons eu dans notre réflexion deux étapes. Une première en générant la liste de tous les plus courts chemins pour chaque nœud du graphe vers tous les autres nœuds de ce dernier. En parcourant cette liste et en récupérant à chaque fois le plus court chemin entre deux points, nous finissons par obtenir une liste contenant le trajet du drone passant par tous les points du graphe.

Le point positif de cette solution était la rapidité. Malheureusement, cette solution couvrait l'ensemble des sommets du graphe et non l'ensemble des arêtes comme nous le recherchions.

Pour palier à ce problème nous avons changé de méthode de parcours. Comme vu dans la partie théorique, nous devons rendre le graphe eulérien. Ceci est possible grâce à la fonction **eulerize()** de networkX. Cette méthode nous permet de répondre efficacement au problème de cette partie même bien qu'elle soit plus lente que la méthode que nous vous avions présentée dans le paragraphe précédent.

Pour autant nous avons relevé un certain nombre d'améliorations potentielles. Le premier objectif aurait pu être d'optimiser le graphe avant de le rendre eulérien afin de retirer des nœuds de degré impair, les impasses notamment. Nous avions également pensé à améliorer la vitesse d'exécution générale de notre code à l'aide du multithreading. Cela nous aurait également permis d'exécuter notre code sur des zones beaucoup plus importantes que des quartiers. A titre d'exemple, avec l'étude d'un cas précis comme le quartier de LaSalle à Montréal nous avons pu observer des différences de temps conséquentes. En utilisant la première méthode, l'analyse se faisait en 14 secondes alors qu'en utilisant la deuxième, nous mettions 1 minute 20 environ. Ces temps sont des moyennes puisqu'ils dépendent en partie de la machine utilisée pour faire les calculs.

B. Trajet minimal des appareils de déblaiement

Nous allons maintenant aborder le cas d'application réel de la déneigeuse. Comme expliqué précédemment, nous avions pensé à utiliser des algorithmes existants. Il est apparu qu'essayer d'obtenir une solution livrable de cette façon ne pouvait pas fonctionner, les temps de computation nous semblaient beaucoup trop importants. Nous avons donc dû aborder le problème de manière différente. Dans la même dynamique que la partie concernant le drone. Nous avons d'abord considéré une recherche de chemin le plus court, mais nous avons ensuite décidé de le voir comme un problème de couverture de graphe.

Nous commençons donc par calculer l'ensemble des chemins les plus courts entre chaque point de notre graphe en utilisant la bibliothèque OSMnx. Une fois l'ensemble des chemins récupéré nous les pré-sélectionnons. La pré-sélection des vecteurs se fait entièrement en fonction du type de quartier étudié. Plus le quartier est grand, plus la taille minimale de vecteurs de sommet à parcourir sera importante. Après cela, notre but est de trouver les plus longs chemins couvrant le plus de sommets non traités. Pour cela nous utilisons des poids relatifs à la longueur des chemins et à la pertinence de ceux-ci.

Cette méthode nous permet d'obtenir des résultats convenables au niveau de la couverture du graphe et des temps d'exécution, moins de 6 min pour des quartiers de taille importante. Néanmoins un certain nombre de problèmes sont à relever, principalement sur la couverture du graphe. En effet, ayant choisi de couvrir le graphe en ne considérant que les sommets et non les arêtes, nous parcourons, dans la majorité des cas, l'entièreté des sommets sans avoir parcouru l'ensemble des arêtes.

Cela est particulièrement vrai dans des villes ayant de nombreuses avenues perpendiculaires, cas très récurrent dans les villes d'amérique du nord. Le second problème a été le réalisme par rapport aux nombres de déneigeuses nécessaires. Si notre fonction ne parvient pas à trouver de nouveaux sommets de manière efficace, les déneigeuses repasseront de manière significative sur des arêtes déjà visitées.

En plus de cela, les coûts deviennent trop importants en raison du nombre de déneigeuses nécessaires. Pour autant cette solution pourrait être améliorée si des liens entre des chemins étaient trouvés et si les déneigeuses possédaient des zones attitrées.

C. Les coûts

Comme nous l'avons brièvement expliqué précédemment, la problématique de coût reste la grosse faille de notre solution. Prenons comme exemple concret le quartier d' Outremont à Montréal. Dans la mesure où nous ne considérons aucune amélioration de notre code il faudrait 131 engins trottoir et 31 engins route afin de couvrir la totalité du quartier. Avec un budget de 7 millions annuel pour le déneigement de Montréal nous avons pris comme valeur le coût d'un engin trottoir à la journée 60$ et comme valeur le coût d'un engin route à la journée 160$. Ainsi, si le déneigement du quartier d'Outremont se fait en une journée, cela coûterait à la ville 12 820$. Nous considérons ce résultat raisonnable mais sommes conscient qu' il est possible de l'améliorer en perfectionnant nos algorithmes en utilisant par exemple l'algorithme de Karatsuba que nous avons découvert bien trop tard dans nos recherches.