Viven BERNARD-NICOD 3MICF
Pol-Elouan BRIENT

BE GRAPHES

PROBLEME OUVERT

Sujet :

Pour ce BE, nous avons opté pour le problème des “centres de graphes”, dans lequel nous plaçons k points sur la carte de telle sorte que la pire distance (ou le pire temps) entre un sommet et l’un des points soit minimale.

Globalement, nous cherchons à optimiser :

$$\text{Coût total} = \sum_{v \in V} w(v) \cdot \min_{c \in C} d(v, c)$$

Où :

- V : ensemble des sommets.
- C : ensemble des centres à déterminer.
- $d(v,c)$: distance ou temps entre un sommet $v \in V$ et un centre $c \in C$.
- $w(v)$: poids (ou population) associé à chaque sommet v — actuellement fixé à 1 (chaque sommet a un poids uniforme)

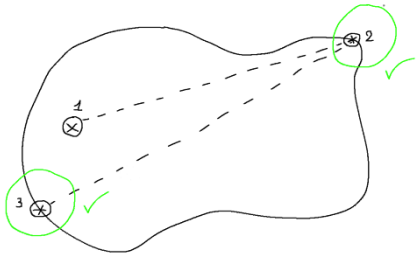
Simplifications :

Afin de résoudre ce problème, nous avons choisi de rester à un niveau de recherche aussi simple que possible. En effet, dans le cadre de cette étude, nous considérons uniquement les graphes fortement connexes comme candidats.

Première idée :

Nous avons d’abord adopté une approche simple, en recherchant un seul centre. L’idée est la suivante : on choisit un sommet aléatoirement dans le graphe, puis on exécute l’algorithme de Dijkstra, avec pour seule condition d’arrêt d’avoir visité tous les nœuds.

Nous récupérons ensuite le sommet le plus éloigné du sommet initial. L’opération est réitérée en repartant de ce nouveau sommet, jusqu’à ce que la distance maximale atteinte ne change plus entre deux itérations. Nous obtenons alors les deux sommets les plus éloignés du graphe. Il ne reste plus qu’à sélectionner le sommet situé au centre du plus court chemin qui les relie.



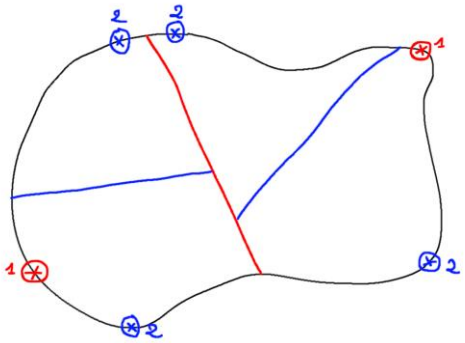
NB : Cette solution, bien que très naïve, permet uniquement d'identifier un centre dit « géographique ». Cependant, elle ne garantit pas que ce centre minimise effectivement la moyenne des distances à tous les autres sommets, et ne satisfait donc pas nécessairement la définition d'un centre de graphe au sens formel.

2ème idée :

Le principe consistait à reprendre l'algorithme précédent, mais cette fois en conservant les extrémités les plus éloignées.

À partir de ces extrémités, nous avons envisagé d'exécuter un algorithme de Dijkstra depuis chacune d'elles. Lorsqu'un nœud est atteint par les deux plus courts chemins, il définit une sorte de « frontière ».

Cette frontière permet alors de diviser le graphe en deux sous-graphes, sur lesquels nous réappliquons la première méthode afin d'en déterminer les centres respectifs.



Cette solution permettait de couvrir toutes les recherches de 2^k centres, puisqu'il suffisait de réitérer la découpe en deux sous-graphes à chaque étape, puis de déterminer leurs centres respectifs.

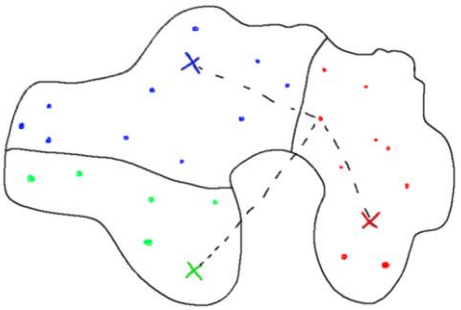
Toutefois, une fois de plus, les centres ainsi trouvés ne garantissaient pas de satisfaire les conditions nécessaires pour être des centres valides.

Il nous fallait donc envisager une autre approche.

Nb : nous avons conservé la methode des Dijkstra en parallèle afin de trouver une frontière pour la suite sur notre algorithme intelligent.

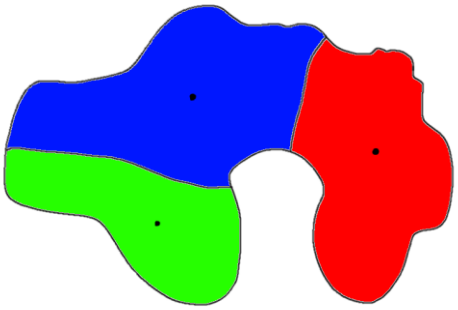
Implémentation finale en bruteforce

• Initialisation:



Le principe est assez simple : nous commençons avec un graphe quelconque et choisissons C nœuds arbitraires sur ce graphe. Pour l'exemple, nous prendrons ici C=3 nœuds, qui représenteront nos centres, correspondant à nos magasins.

• Première itération :

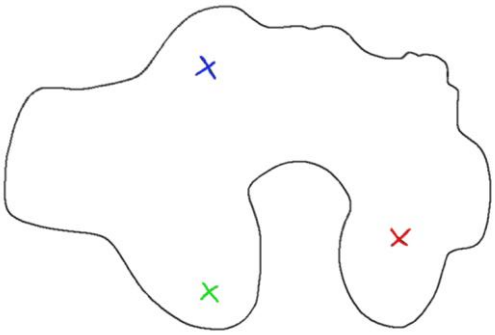


Ensuite, pour chaque point du graphe, nous déterminons quel est le centre le plus proche (en utilisant par exemple un algorithme de Dijkstra ou A*), puis nous l'attribuons au groupe correspondant à ce centre.
Nous obtenons ainsi C groupes.

• Réarrangement :

Après avoir créé nos groupes, nous cherchons, dans chaque groupe, le point central, c'est-à-dire le sommet dont la distance moyenne à tous les autres sommets du sous-graphe associé est la plus faible.

• Répétition :



Nous répétons ce test en boucle jusqu'à ce que la somme des coûts des parcours en largeur des C groupes soit stable, c'est-à-dire égale entre eux à un seuil de tolérance près (plus ou moins strict selon le temps dont on dispose).

Concrètement, à chaque itération, on calcule pour chaque centre c le coût d'aller de ce centre à tous les autres nœuds de son groupe, puis on somme ces coûts.

Le programme s'arrête lorsque les sommes des coûts de tous les groupes sont égales à l'intérieur d'un delta, par exemple 5%.

• Coût :

Cet algorithme, à implémenter, serait assez coûteux et nous en avons pleinement conscience. Nous pensons qu'il est possible d'y apporter des améliorations, mais étant donné que cet algorithme n'est utilisé qu'une seule fois pour déterminer l'emplacement des magasins, ce coût computationnel n'est pas nécessairement un frein majeur.

• Idée d'optimisation :

Cette fois-ci, au lieu d'effectuer une recherche du plus court chemin pour chaque nœud du graphe afin de l'associer au centre le plus proche (nous appelons « centre » un des nœuds choisis comme centre temporaire), nous utilisons un algorithme de Dijkstra dynamique.

Voici son fonctionnement théorique :

- Nous avons une liste de Labels spécifiques à cet algorithme : des LabelsCentered qui ont deux particularitées :
    - Ils enregistrent le sommet d'Origine initial du parcours lorsqu'ils sont visité une première fois.
    - Ils sont considérés comme réellement visité si :
        - Ils sont visité une deuxième fois par un second parcours (provenant donc d'un autre nœud)
        - Ils font parti d'un parcours dont le dernier sommet a été visité deux fois.
- Les labels ainsi considérés comme visités ne peuvent plus être visité par un autre parcours provenant d'un enième nœud central.
- Autrement, l'algorithme fonctionne comme un dijkstra classique avec un binaryHeap associé à chaque nœud central, les LabelsCentered inséré dans les binaryHeap correspondant au nœud central qui l'a atteint.

Nous appelerons par la suite ce Dijkstra adapté à la recherche de centre : DijkstraCentered.

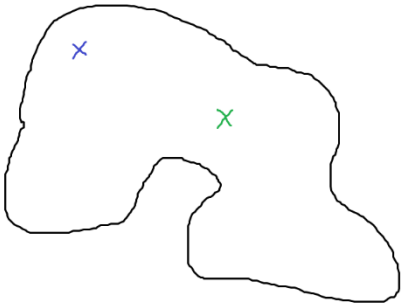
Concernant le fonctionnement de notre nouvelle logique algorithmique « intelligente », nous avons modifié certaines étapes de la recherche en force brute.

Implémentation finale intelligente

- Initialisation :

Nous choisissons arbitrairement C sommets correspondant au nombre de centres recherchés.

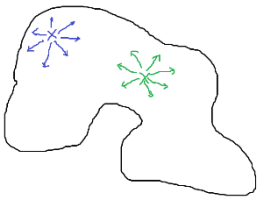
Ici nous avons choisis aléatoirement deux sommets : un vert et un bleu.



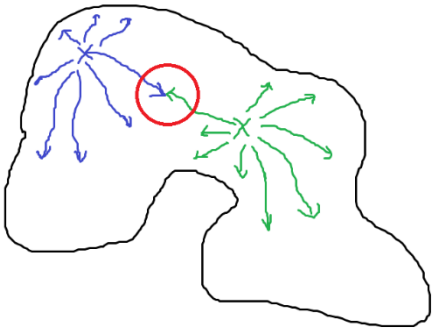
- Première itération :

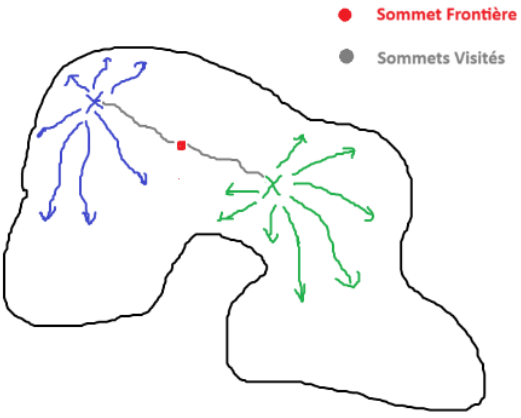
Nous appliquons l'algorithme du DijkstraCentered afin de séparer notre graph en C sous-graphs. Cela se passe donc en plusieurs temps ;

Parcours des sommets en appliquant un Dijkstra sans sommet cible à chaque nœud centraux



Lorsqu'un nœud est atteint par deux parcours, il est considéré comme visité ainsi que tous les nœuds qui constituent le plus court chemin vers ce nœud.

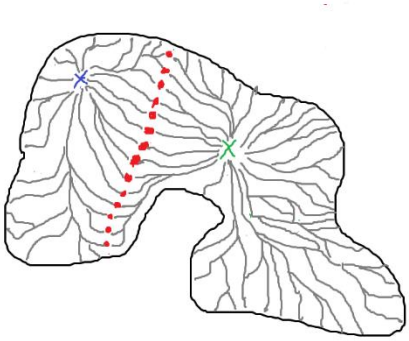


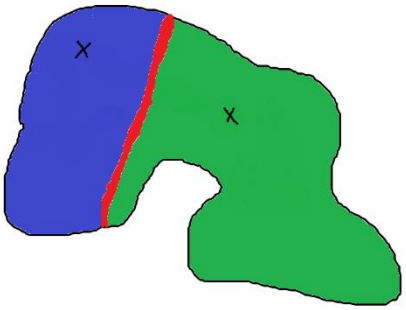


- Découpage en sous-graphs :

La délimitation des frontières de chaque sous-graphe se fait à partir des « nœuds frontières », que l'on supprime temporairement du graphe afin de réaliser un parcours en largeur (BFS) à partir de chaque nœud central.

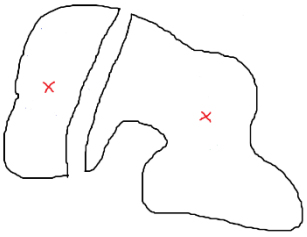
Ainsi, tous les sommets accessibles depuis un nœud central appartiennent au sous-graphe associé à ce nœud.





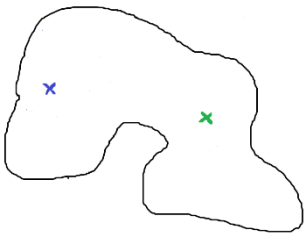
- Recherche des centres :

Recherche du centre réel de chaque sous-graph en methode bruteforce : On effectue un parcours de tout le sous-graphe avec Dijkstra en partant de chaque nœud de celui-ci pour determiner celui qui a la distance moyenne la plus basse vis-à-vis de l'ensemble des autres nœuds du sous-graph.



- Mises à jour :

On recommence à partir de l'étape d'initialisation mais en séléctionnant cette fois ci les C nœuds centraux délimités précédemment.



- Arret :

On arrête d'itérer soit lorsque les nœuds centraux ne changent presque plus, selon une heuristique comparative que l'on peut fixer entre 1 et 5 km à vol d'oiseau en fonction des besoins, soit arbitrairement après un maximum de 5 itérations.

Les différentes complexités

Nous considérons pour les calculs de compléxité que :

- Nous avons un graph V.
- Nous cherchons C centres.
- Il possède N sommets.
- Il possède M arrètes.
- Le degrès moyens de chaque sommet est B.
- La profondeur moyenne d'un chemin optimale est D.

Ainsi nous supposons que la compléxité moyenne de Dijkstra sur notre graph est de $O((M+N) \log N)$ et pour A* de 0(B^D).

• Pour l'algorithme bruteforce :

Nous devons réaliser sur un graph de N sommets, N recherche de plus court chemin avec A star afin de determiner nos différents clusters.

Soit une complexité de 0(N(B^D)).

Supposons que nos cluster donnent des sous-graphs ayant en moyenne :

- N/C sommets
- M/C arrètes.

Nous devons alors réaliser pour chaque sous-graph un Dijkstra complet sur l'ensemble des nœuds du sous graph soit : C * N/C * O(($(N+M)/C$)log(N/C)) ce qui équivaut à O(N*(N+M)*log N). En additionnant nos deux complexités nous obtenons une complexité moyenne de O(max(N*B^D , N(N+ M)logN). Nous observons que cette complexité peut rapidement être élevée en fonction de la profondeur moyenne D.

• Pour l'algorithme intelligent :

Concernant notre algorithme de découpage du graphe en sous-graphes, sa complexité dépend principalement de N et de M. On peut donc estimer sa complexité au même ordre que celle de Dijkstra, soit $O((M+N)\log\ N)$, négligeable face à delle de la recherche de centres. La complexité finale de notre algorithme, incluant la recherche des centres, sera donc de l'ordre de $O(N(M+N)\log\ N)$, ce qui reste toujours équivalent ou meilleur que celle d'une recherche exhaustive en force brute.