Viven Bernard-Nicod 3micF Pol-Elouan Brient

BE GRAPHES

PROBLEME OUVERT

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

Nous avons d’abord pensé simple, avec deux centres. Nous avons eu l’idée de prendre un point au hasard sur la carte, faire la distance maximale entre ce point et tous les sommets du graphes. Nous avions alors une extrémité de la carte. Ensuite il suffisait de refaire une distance maximale depuis ce nouveau point et nous avions les 2 centres d’un bout et l’autres de la carte.

// insérer dessin de la réalisation

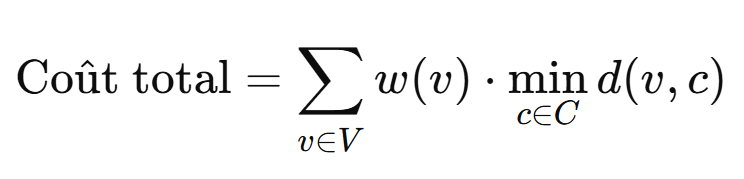
Cette solution ne couvrant que 2 points, il nous fallait trouver une autre solution, nous avons alors pensé à la découpe de carte....

// présenter le truc où la carte est coupée en 2 à chaque fois qui couvre 2^k magasins

Mais cette solution n’était toujours pas la bonne : cette réalisation, en plus de ne couvrir que des puissances de 2 (2^k magasins), couvrait un nombre maximum de zones de la carte sans prendre en compte la densité de population.

Nous nous somme alors rabattus sur une autre solution;

Globalement, nous cherchons à optimiser :



Où V = ensemble de sommets,

C = ensemble des centres à trouver

d(v,c) = distance/temps entre le sommet et le centre

w(v) = population par sommet (non utilisé pour l’instant donc à 1 par défaut => chaque sommet = poids de 1)

J’ai demandé un truc optimal a chat gpt pour ça dcp ça donne ça :

Rapport : Comment réaliser le k-médian sur un graphe de ville avec poids uniforme

----------------------

1. Problème posé

Tu disposes d’un grand graphe représentant une carte urbaine. Chaque **sommet** correspond à une intersection ou un point de la ville. Tu veux placer **k magasins** à certains sommets pour **minimiser la somme des distances** de chaque sommet à son magasin le plus proche.

**Important :** Ici, on considère que chaque sommet a un poids égal à 1, donc la somme des distances revient à minimiser la distance globale parcourue par tous les habitants, en supposant un habitant par sommet.

----------------------

2. Objectif mathématique simplifié

On cherche à choisir un ensemble CCC de kkk sommets (les emplacements des magasins) parmi tous les sommets VVV, de manière à minimiser :

Couˆt=∑v∈Vmin⁡c∈Cd(v,c)\text{Coût} = \sum\_{v \in V} \min\_{c \in C} d(v, c)Couˆt=v∈V∑​c∈Cmin​d(v,c)

où

d(v,c)d(v, c)d(v,c) est la distance ou le temps entre le sommet vvv et un magasin ccc,

CCC est l’ensemble des kkk magasins choisis,

VVV est l’ensemble de tous les sommets.

----------------------

3. Pourquoi ce problème est difficile

Il faut tester de nombreux ensembles possibles de kkk magasins parmi des milliers voire millions de sommets.

Le nombre de combinaisons est gigantesque : impossible d’essayer toutes les solutions.

Donc, on utilise des **méthodes approchées (heuristiques)** pour trouver une bonne solution rapidement.

----------------------

4. Étapes simples pour réaliser le k-médian

Étape 1 : Calculer les distances entre sommets

Tu as déjà les algorithmes pour calculer la distance (ou le temps) entre deux sommets. Cette étape consiste à préparer ou calculer rapidement, pour chaque sommet vvv, la distance vers les magasins potentiels ccc.

----------------------

Étape 2 : Trouver une solution initiale (choisir les premiers magasins)

On commence par sélectionner **k sommets** comme centres initiaux.

Voici une méthode simple et efficace :

Choisir un premier sommet **au hasard** ou le sommet « le plus central ».

Puis, choisir le deuxième magasin comme celui qui est **le plus éloigné** du premier (pour bien répartir).

Ensuite, choisir le troisième magasin comme celui qui est le plus éloigné des deux premiers magasins choisis, et ainsi de suite…

Cette méthode appelée « Farthest-First » assure une bonne couverture initiale.

----------------------

Étape 3 : Affecter chaque sommet à son magasin le plus proche

Pour chaque sommet vvv, tu regardes les distances vers tous les magasins choisis, puis tu choisis le magasin ccc qui minimise d(v,c)d(v, c)d(v,c).

----------------------

Étape 4 : Améliorer la solution par échange (swap)

Une fois la solution initiale fixée, on peut essayer de l’améliorer.

Pour chaque magasin ccc choisi, essaie de le **remplacer** par un autre sommet vvv qui n’est pas un magasin.

Calcule la nouvelle somme des distances après ce remplacement.

Si le remplacement diminue la somme totale, on garde ce changement.

Sinon, on revient à la solution précédente.

On répète ce processus d’échanges jusqu’à ce qu’aucun échange ne puisse améliorer la solution.

----------------------

Étape 5 : Répéter plusieurs fois

Pour éviter de rester bloqué dans une solution qui n’est pas bonne, on peut :

Recommencer plusieurs fois la procédure avec des points de départ différents (différentes solutions initiales).

Garder la meilleure solution trouvée parmi toutes.

----------------------

5. Pourquoi cette approche marche bien avec poids uniformes

Comme chaque sommet compte pour 1, les zones où les sommets sont très nombreux (par exemple, un centre-ville dense) vont naturellement avoir plus de « poids » dans la somme totale des distances.

Donc, les magasins auront tendance à se concentrer là où il y a beaucoup de sommets.

Ce comportement correspond bien à ce qu’on veut : desservir en priorité les zones très peuplées.

----------------------

6. Ce que tu obtiens

Un placement de kkk magasins répartis sur ta carte, qui minimise la somme des distances entre chaque sommet et son magasin le plus proche.

Un découpage naturel de ta ville en kkk zones d’influence, chacune associée à un magasin.

La garantie d’une bonne couverture globale, sans que certains sommets soient excessivement éloignés de leur magasin.

----------------------

7. Conseils pratiques

Le calcul des distances est la partie la plus lourde, mais comme tu l’as déjà, tu peux t’appuyer dessus.

Ne cherche pas la solution optimale exacte, elle est trop coûteuse en temps.

Utilise l’algorithme simple d’**initialisation + amélioration par échanges**.

Recommence plusieurs fois avec des départs différents pour éviter de rester bloqué.

----------------------

8. En résumé

**Prépare tes données** : distances entre sommets, k nombre de magasins.

**Choisis k magasins initiaux** (méthode Farthest-First ou aléatoire).

**Attribue chaque sommet à son magasin le plus proche**.

**Teste des échanges** de magasins pour améliorer la somme des distances.

**Répète avec différentes initialisations** pour obtenir la meilleure solution.

----------------------

Avec la version plus optimisée :

Comment calculer localement la variation de coût lors d’un échange dans k-médian

----------------------

Situation de départ

* Tu as déjà une solution avec kkk magasins placés sur certains sommets.
* Pour chaque sommet vvv, tu connais :
  + Le magasin **le plus proche** cvc\_vcv​,
  + La distance correspondante d(v,cv)d(v, c\_v)d(v,cv​).

----------------------

Objectif

Tu veux tester un échange :  
**Remplacer un magasin coldc\_{old}cold​ par un candidat cnewc\_{new}cnew​** (un sommet non magasin),  
et savoir si ça réduit la somme totale des distances.

----------------------

Étapes simples pour calculer la variation de coût

1. Pour chaque sommet vvv, regarde si coldc\_{old}cold​ est son magasin actuel le plus proche.

* Si **non** :
  + Sa distance au magasin le plus proche ne changera **pas**, car cet échange ne le concerne pas directement.
  + Donc, son coût reste le même.
* Si **oui** :
  + Ce sommet vvv était attaché à coldc\_{old}cold​.
  + Il faut maintenant voir s’il est plus proche de cnewc\_{new}cnew​ ou d’un autre magasin (parmi les magasins restants sauf coldc\_{old}cold​).

----------------------

2. Calcul des distances pour vvv attaché à coldc\_{old}cold​

Calcule la distance d(v,cnew)d(v, c\_{new})d(v,cnew​).

Trouve la distance minimale entre vvv et tous les autres magasins différents de coldc\_{old}cold​ (car coldc\_{old}cold​ sera remplacé).

La nouvelle distance pour vvv sera donc :

dnouveau(v)=min⁡(d(v,cnew),min⁡c∈Cc≠coldd(v,c))d\_{\text{nouveau}}(v) = \min \big( d(v, c\_{new}), \min\_{\substack{c \in C \\ c \neq c\_{old}}} d(v, c) \big)dnouveau​(v)=min(d(v,cnew​),c∈Cc=cold​​min​d(v,c))

----------------------

3. Variation locale du coût pour vvv

Avant échange, coût partiel : d(v,cold)d(v, c\_{old})d(v,cold​).

Après échange, coût partiel : dnouveau(v)d\_{\text{nouveau}}(v)dnouveau​(v).

Variation pour vvv :

Δv=dnouveau(v)−d(v,cold)\Delta\_v = d\_{\text{nouveau}}(v) - d(v, c\_{old})Δv​=dnouveau​(v)−d(v,cold​)

----------------------

4. Somme des variations

Pour chaque sommet vvv attaché à coldc\_{old}cold​, calcule Δv\Delta\_vΔv​.

Additionne toutes ces variations ∑Δv\sum \Delta\_v∑Δv​.

Cette somme est la **variation totale** du coût si tu remplaces coldc\_{old}cold​ par cnewc\_{new}cnew​.

----------------------

5. Décision finale

Si ∑Δv<0\sum \Delta\_v < 0∑Δv​<0, l’échange améliore la solution (diminue le coût total), tu peux l’accepter.

Sinon, tu rejette cet échange.

----------------------

Pourquoi cette méthode est rapide

Tu ne reviens pas sur les sommets attachés à d’autres magasins, tu n’étudies que ceux affectés à coldc\_{old}cold​.

Le nombre de sommets concernés est souvent bien plus petit que l’ensemble total.

Tu n’as pas à recalculer la distance pour chaque sommet à chaque échange.

Tu utilises la mémoire de qui est actuellement affecté à quel magasin et leur distance.

----------------------

Astuce pratique

Pour chaque magasin ccc, conserve la liste des sommets qui lui sont attachés.

Quand tu testes un échange pour coldc\_{old}cold​, parcours seulement cette liste.

Ça réduit considérablement le travail.

J’ai pas exactement tout compris mais ça peut aider si on a pas d’autres solution