INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Bensebaa Achraf
Genty Émilie

Rapport BE Graphe
Problème ouvert
3 MIC groupe E

Au cours de ce BE Graphes, nous avons pu utiliser un algorithme des plus courts chemins, afin de résoudre un problème complexe de calcul de trajet. Nous avons ainsi,choisi le problème du marathon : trouver un chemin formant une boucle à partir d'un point de départ O.

Pour cela nous considérons un graphe G(X,E) avec dim X=n et dim E=m, non orienté et connexe. L'objectif est donc de trouver un chemin d'environ 42 km appartenant à ce graphe. Une solution existe si le graphe comporte un point représentant le centre du cercle englobant un octogone de périmètre 42km. De plus, une contrainte nécessaire à la résolution du problème est que le graphe fourni soit suffisamment dense. Nous reviendrons par la suite sur ce point.

De plus, les contraintes sur le circuit pour que le parcours soit accepté sont les suivantes :

- Pas de cycle dans le chemin (ne pas passer 2 fois par le même sommet)
- Le point de départ et le point d'arrivée du marathon doivent se trouver à proximité
- La distance totale du marathon doit être égale à 42 km ± une marge en km

Principe de l'algorithme :
On choisit un point sur le graphe représentant le centre d'un cercle de rayons suffisamment grand pour y inscrire un parcours de 42km. Pour cela la distance séparant ce point du bord doit être supérieure à environ 7km.
Ensuite, on trouve 8 points théoriques positionnés sur ce cercle, formant un octogone inscrit, de 42 km de périmètre. Le parcours du marathon est donné par la concaténation des plus courts chemins entre les points voisins formant le cercle. Afin de déterminer les plus courts chemins, nous avons choisi d'utiliser l'algorithme de Dijkstra.

Pour proposer un algorithme permettant de trouver le parcours du marathon nous aurons besoin d'une fonction annexe noté "AddSommetPlusProche". Cette fonction permet pour un point donné de trouver le sommet le plus proche sur le graphe.
Par la suite on notera (x, y) le sommet de coordonnées x et y.

Fonction annexe :

```
AddSommetPlusProche(Liste L, Sommet s){
        dis_min = infini
        sommetPlusProche = null
        Pour tous les sommets x du graphe :      //complexité  => O(n)
                if (distance(x,s) < dis_min)
                    dis_min = distance(x, s)
                    sommetPlusProche = x
        Add(L, sommetPlusProche)}
```

Algorithme Marathon :

- Initialisation :

```
Soit s un point du graphe dont la distance au bord est d'environ 7km      O(n)
ListeSommet = []      //liste de Node
Parcours = []         //liste d'arc
r = 42 / (8*0.765)    //formule périmètre d'un octogone (0.756 R*8)        O(1)
marge= 1 km           //distance entre départ et arrivée (arbitraire)
```

- Ajout des sommets sur le cercle :

```
//on ajoute des points sur le cercle pour former l'octogone
AddSommetPlusProche(ListeSommet, (s.x, s.y + r))
AddSommetPlusProche(ListeSommet, (s.x, s.y - r))
AddSommetPlusProche(ListeSommet, (s.x + r, s.y))
AddSommetPlusProche(ListeSommet, (s.x - r, s.y))
AddSommetPlusProche(ListeSommet, (s.x + r*cos(π/4), s.y + r*sin(π/4))
AddSommetPlusProche(ListeSommet, (s.x - r*cos(π/4), s.y + r*sin(π/4)))
AddSommetPlusProche(ListeSommet, (s.x + r*cos(π/4), s.y - r*sin(π/4))
AddSommetPlusProche(ListeSommet, (s.x - r*cos(π/4), s.y - r*sin(π/4)))
```

O(n) pour chaque ligne O(n)

- Utilisation dijkstra:

```
For i allant de 0 à taille [Liste Sommet-1] faire
        chemin = Dijkstra(ListeSommet[i], Liste Sommet[i+1]).getPath()
        Add(Parcours, chemin)
end For
if distance(parcours) = 42 ± marge
        retourner Parcours
else
        retourner Erreur("graphe non dense")
```

$O(n^2log(n))$

L'algorithme termine car la boucle for est finie (nombre fini d'itérations) puisqu'elle n'est exécutée que 7 fois. Une fois par chemin séparant 2 points successif de l'octogone.

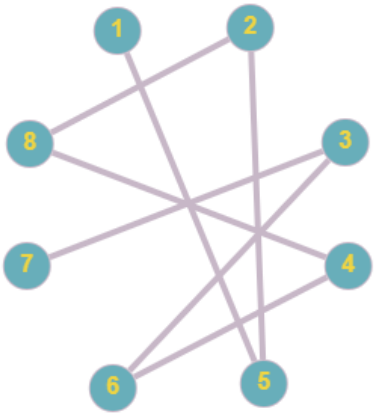
Pour ne pas avoir de cycle dans notre marathon, il ne faut pas réutiliser les sommets utilisés lors d'un parcours réalisé avec Dijkstra sur les sections précédentes.
Pour cela on doit ajouter un label "utilisé" et à la fin de Dijkstra, les sommets marqués (avec le label "marked") doivent être mis à "utilisé". Ainsi on modifie l'initialisation des sommets dans l'algorithme de Dijkstra, et les sommets déjà notés "utilisé" doivent être labellisés "marked" pour ne pas être réutilisés.

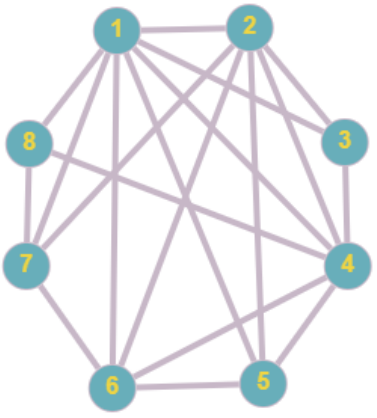
Ces changements en début et fin de Dijkstra ne modifient pas la complexité de l'algorithme Dijkstra qui est en O((n+m)log(n)). La complexité de l'algorithme est donc en O(n) + O(1) + 8.O(n) + O((n+m)log(n)) = **$O(n^2*log(n)+mn*log(n))$**

Notre algorithme nous donne donc un chemin de 42km plus ou moins la marge défini à l'initialisation. Si la distance est égale à 42+marge km alors il faudrait par la suite enlever une des arêtes du chemin pour obtenir un circuit de 42 km. Pour cela on retire une arête dont la taille est égale à la marge. Ainsi, obtient un circuit ouvert de 42 km pour lequel le point d'arrivée et de destination sont suffisamment proches.

Problèmes rencontrés :

Comme nous l'avons précisé plus tôt, notre algorithme fonctionne uniquement dans le cas d'un graphe dense. En effet, supposons que l'on trouve 8 points dans un graphe formant un octogone de 42 km. Le plus court chemin construit par Dijkstra pour toute paire de sommets voisins du cercle risque de s'éloigner d'autant plus de la distance à vol d'oiseau que le graphe est peu dense.





Dans l'exemple ci-dessus, nous pouvons voir facilement que notre algorithme a beaucoup plus de chances de fonctionner dans le graphe dense.
En effet, dans le graphe de gauche, le plus court chemin entre les sommets successifs, par exemple de 1 à 2 passe par un chemin dont la distance est plus grande que le trajet à vol d'oiseau. Une fois toutes les distances cumulées, le marathon ne fait pas forcément 42km comme voulu. Dans le graphe de droite, plus dense, on remarque que les sommets successifs sont reliés directement entre eux et donc dans ce cas idéal, les plus courts chemins correspondent exactement aux distances souhaitées.

Pour pallier ce problème, nous pouvons introduire un coefficient de densité représentant le rapport entre m (nombre d'arêtes) et n(n-1)/2 (nombre d'arêtes maximal). Plus ce coefficient est proche de 1, meilleur sera le résultat donné par notre algorithme.

Améliorations futures :

Afin d'adapter notre algorithme à un graphe peu dense, nous pouvons fixer une limite lorsque nous appliquons un parcours Dijkstra pour trouver un chemin entre deux sommets voisins de notre octogone. Si le coût de ce chemin est trop grand ou trop petit par rapport à la distance à vol d'oiseau, nous cherchons huit nouveaux points dans le graphe qui seraient plus adaptés à notre problème. Nous pouvons également changer le nombre de points en fonction du graphe.

En conclusion, nous pouvons lancer notre algorithme plusieurs fois en adaptant les paramètres en fonction du graphe. En contrepartie, la complexité de notre algorithme augmente rendant notre solution peu utilisable.