Programming project

Ahmad Alae, Dreyfus Jacques

Décembre 2022

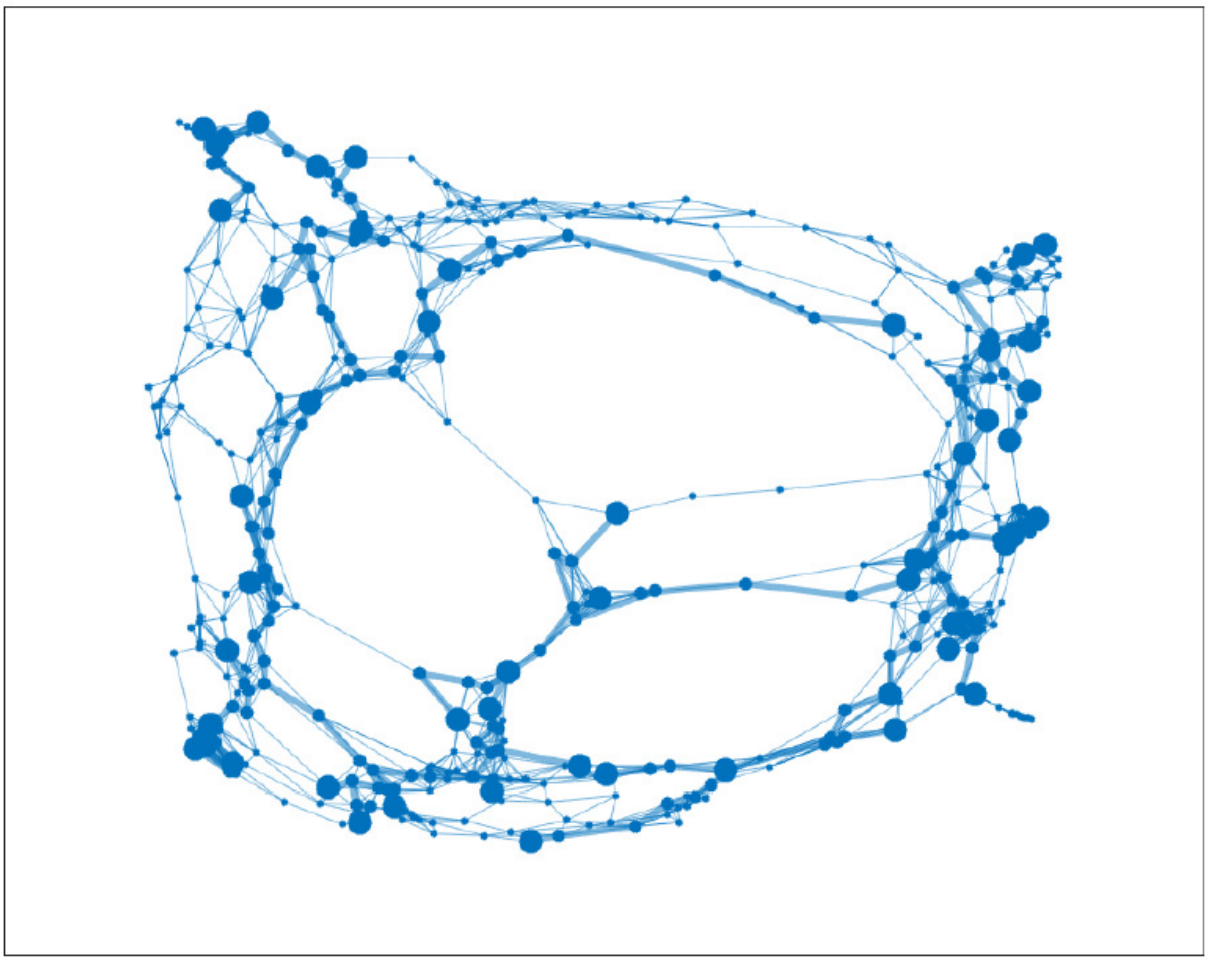


Table des matières

1 Introduction

Les arbres de Steiner, en théorie des graphes, sont utilisés pour trouver la plus courte distance entre des points d'un réseau. Ils sont utilisés dans le contexte de l'Internet des Objets (IdO) pour trouver des routes de communication optimales entre différents objets connectés. On maximise ainsi l'efficacité des lignes de communication entre les objets.

On travaillera dans ce sujet sur des graphes, représentés par des matrices d'adjacence. On supposera dans l'ensemble de ce sujet que la fonction de poids c est à valeurs dans $\mathbb{R}_+^*$ afin d'assurer la bonne définition de la marice d'adjacence et d'éviter les cas pathologiques. C'est d'ailleurs ce qui est fait dans le cours.

2 Pre-Processing steps

Question 1

Supposons qu'on ait un Arbre de Steiner Minimal (ASM) T qui contient une arête dont l'un des sommets est non-terminal et de degré 1. Ce sommet est donc une feuille de l'arbre T. En le retirant, on conserve la structure d'arbre ; le sommet étant non-terminal, notre nouvel arbre T' est également de Steiner, et $c(T') < c(T)$ car on a retiré une arête (qu'on suppose de poids stictement positif). Donc T n'était pas minimal, ce qui est absurde par hypothèse. D'où le résultat.

Question 2

L'algorithme retire les sommets non-terminaux de degré 1.

Entrée : M la matrice d'adjacence$(n \times n)$ (avec poids) du graphe G, K l'ensemble de sommets étudié

Pour $1 \leq i \leq n$:

Si $i \notin K$ et $L_i(M)$ contient un seul élément non nul en position j :

Retirer le sommet i du graphe G

Fin pour

On utilise la matrice d'adjacence du graphe G, dont on parcourt les n-k dernières lignes. Voici un exemple de résultat d'application de cet algorithme, ici sur le scénario 1 :

```
0 5 6 1 3 0 0 0 0
5 0 0 5 0 2 7 0 0
6 0 0 0 2 0 3 0 3
1 5 0 0 2 1 0 0 0
3 0 2 2 0 4 0 0 0
0 2 0 1 4 0 4 0 0
0 7 3 0 0 4 0 1 1
0 0 0 0 0 0 1 0 0
0 0 3 0 0 0 1 0 0
```

(a)

```
0 5 6 1 3 0 0 0
5 0 0 5 0 2 7 0
6 0 0 0 2 0 3 3
1 5 0 0 2 1 0 0
3 0 2 2 0 4 0 0
0 2 0 1 4 0 4 0
0 7 3 0 0 4 0 1
0 0 3 0 0 0 1 0
```

(b)

FIGURE 1 – (scénario 1) - (a) : avant exécution, (b) : après exécution

Question 3

Supposons que $c(e_1) + c(e_2) \geq c(e)$ $(*)$. On cherche un ASM qui ne contienne pas le sommet v_k. Soit T un ASM pour K. Si $v_k \notin T$, alors T convient. Sinon, on considère T' l'arbre obtenu en remplaçant les

arêtes e_1 et e_2 par l'arête e. Alors, T' est toujours connexe car $\deg(v_k) = 2$, et c'est en fait encore un arbre de Steiner (car le sommet v_k n'est pas terminal), et T' est minimal d'après l'inégalité $(*)$. Donc, T' convient.

Supposons désormais que $c(e_1) + c(e_2) > c(e)$, et montrons qu'on peut supprimer l'arête e de G. Supposons par l'absurde qu'il existe un ASM T contenant l'arête e. Considérons T' l'arbre obtenu en remplaçant l'arête e par les arêtes $e1$ et $e2$. T' est évidemment toujours un arbre de Steiner, et $c(T) - c(T') = c(e) - c(e_1) - c(e_2) > 0$. Absurde par minimalité de T. Aucun ASM ne peut contenir l'arête e, on peut donc la retirer du graphe G.

Question 4

Pour $1 \leq p \leq n$:

Si $p \notin K$ et $L_p(M)$ contient exactement deux éléments non nuls en position i et j :

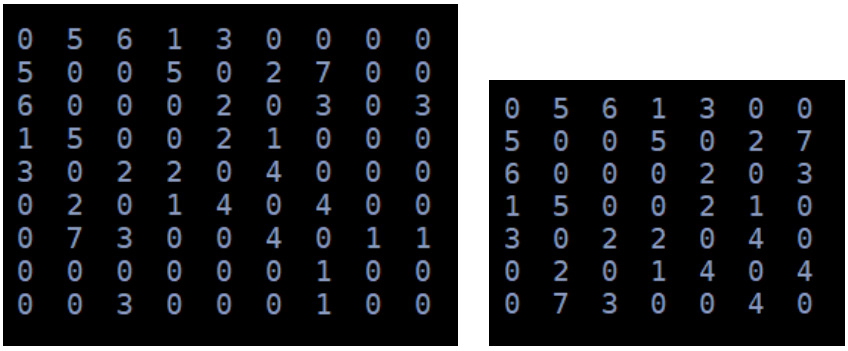
Si $M_{pi} + M_{pj} \geq M_{ij}$ et $M_{ij} > 0$:

Retirer le sommet p du graphe G

Sinon :

$M_{ij} \leftarrow 0$

Fin pour



(a) (b)

FIGURE 2 – (scénario 1) - (a) : avant exécution, (b) : après exécution des algorithmes Q.1 et Q.3

Question 5

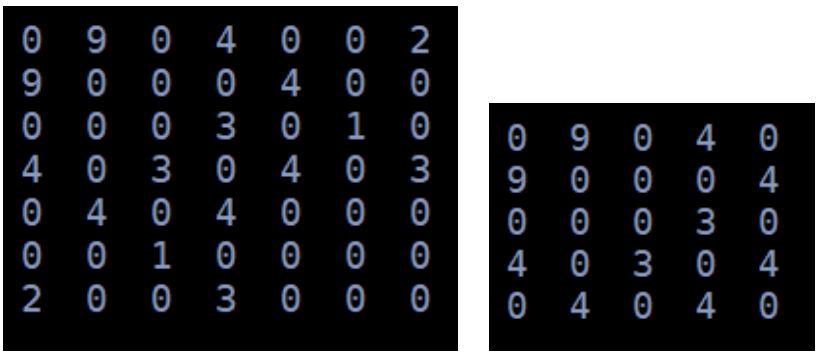
Soit une arête $e = (v_i, v_j)$ avec $c(e) > d(v_i, v_j)$. Notons C le chemin réalisant la distance entre v_i et v_j. Soit par l'absurde T un ASM contenant l'arête e. Soit T' l'arbre obtenu en retirant e à T et en ajoutant le nombre minimal d'arêtes à T pour que $C \subset T$. T' est également un arbre de Steiner, de poids total inférieur à celui de T. Absurde car T est supposé minimal. Donc, aucun ASM ne peut contenir l'arête e, on peut la supprimer du graphe.

Question 6

Nous ne donnons pas ici le pseudo-code l'algorithme de Floyd-Warshall, car il a été vu en cours. Pour plus de détails, voir son implémentation dans le code Python.

Calculer D la matrice des distances du graphe G via l'algorithme de Floyd-Warshall

Pour $1 \leq i \leq n$:
    Pour $i \leq j \leq n$: ($*$ on utilise le fait que M soit symétrique $*$)
        Si $M_{ij} > 0$ et $M_{ij} > D_{ij}$:
            $M_{ij} \longleftarrow 0$
            $M_{ji} \longleftarrow 0$
        Fin pour
    Fin pour



(a) (b)

FIGURE 3 – (scénario 2) - (a) : avant exécution, (b) : après exécution des 3 algorithmes précédents

3 Algorithmes exacts

3.1 L'algorithme d'énumération

Question 7

On énumère tous les sous-ensembles S possibles de cardinal exactement k-2 (inutile d'observer ceux de cardinal inférieur qui sont inclus dans un sous-ensemble de cardinal k-2) et on considère les arbres couvrants minimaux obtenus sur le sous-graphe induit par $K \cup S$. On choisit celui de poids minimal qu'on transforme en un arbre de Steiner avec la fonction Remonter(M, MST, P) qui utilise la matrice des chemins les plus courts de Floyd-Warshall P pour remonter d'un arbre minimal du distance network

à un arbre de Steiner du graphe (cf. question 12 pour son pseudo-code)

Données :
Matrice d'adjacence $M(n \times n)$
Ensemble K

Variables :
$D \leftarrow$ DistanceNetwork(M)
$P \leftarrow$ CheminsCourts(M) (* D et P sont obtenus avec un algorithme de Floyd-Warshall *)
ArbresMinimal $\leftarrow \emptyset$
PoidsMinimal $\leftarrow \infty$

Pour S sous-ensemble de $M \backslash K$ de cardinal $k-2$:
    Soit M' le sous-graphe de$D$induit par $K \cup S$
    nt = ArbreCouvrantMinimal (M')
    Si Poids(nt) < PoidsMinimal :
        ArbreMinimal $\leftarrow nt$
        PoidsMinimal $\leftarrow Poids(nt)$
Fin pour
T = Remonter(M, ArbreMinimal)
Return T

Question 8

L'algorithme énumère l'ensemble des $O(n^{k-2})$ liste de points de branchement possibles. Pour chacune d'elle, le calcul d'un arbre couvrant minimal se fait en $O(n^2)$.
L'algorithme de Floyd-Warshall et la fonction Remonter, respectivement en $O(n^3)$ et $O(n^2)$, ont une complexité négligeable par rapport à l'étape d'énumération.

D'où une complexité en $O(n^k)$ pour l'algorithme d'énumération. Cette complexité exponentielle le rend inutilisable pour des graphes de taille moyenne.

3.2 L'algorithme de Dreyfus-Wagner

Question 9

D'après le cas 1, on obtient l'équation suivante pour $s_v(X \cup v$:

$$s_v(X \cup v) = \min_{\emptyset \neq X' \nsubseteq X} (s(X' \cup v) + s(X \backslash X' \cup v))$$

on déduit d'autre part, d'après les équations 2 et 3, l'équation suivante pour $S(X \cup v)$:

$$s(X \cup v) = \min_{w \in V} (s(X).\mathbb{1}_{w \in X} + s_w(X \cup v).\mathbb{1}_{w \notin X} + d(v, w))$$

Pour simplifier l'écriture de l'algorithme, nous allons nous ramener à une seule équation. En réinjectant la première expression dans la seconde, on obtient :

$$s(X \cup v) = \min_{w \in V} \underbrace{\left(d(v,w) + s(X).\mathbb{1}_{w \in X} + \min_{\emptyset \neq X' \not\subseteq X} (s(X' \cup w) + s(X \backslash X' \cup w).\mathbb{1}_{w \notin X}) \right)}_{f(w)}$$

Montrons maintenant que cette expression peut se simplifier en la suivante :

$$s(X \cup v) = \min_{w \in V} \underbrace{d(v,w) + \min_{\emptyset \neq X' \not\subseteq X} s(X' \cup w) + s(X \backslash X' \cup w)}_{g(w)}$$

Commençons par montrer que $\min g \leq \min f$. Soit w_0 tel que $f(w_0)$ soit minimal. Si $w_0 \notin X$, l'inégalité est claire. Si $w \in X$, alors on peut prendre $X' = w$ ce qui nous donne l'inégalité voulue. D'où, $\min g \leq \min f$.

Montrons maintenant que $\min g \geq \min f$. Soit w_0 minimisant g. Si $w_0 \notin X$, alors $f(w_0) = g(w_0)$, ce qui nous donne immédiatement l'inégalité. Supposons donc que $w_0 \in X$. On a :

$$f(w_1) = d(v,w_1) + s(X) \tag{1}$$
$$g(w_1) = d(v,w_1) + \min_{\emptyset \neq X' \not\subseteq X} s(X' \cup w) + s(X \backslash X' \cup w) \tag{2}$$

Soit X'_1 réalisant le min dans (2). Au lieu de $T(X'_1 \cup w)$ et $T(X \backslash X'_1 \cup w)$, on peut choisir comme sous-ensembles $T(X \backslash \{w\})$ et $\{w\}$ (car $w \in X$!) qui auraient au pire le même poids.

Ceci conclut et nous donne l'égalité voulue :

$$s(X \cup v) = \min_{w \in V} d(v,w) + \min_{\emptyset \neq X' \not\subseteq X} s(X' \cup w) + s(X \backslash X' \cup w)$$

Question 10

On cherche à calculer :

$$\forall\, v \in V, \forall\, X \subseteq K, ST(X,v) := [T(X \cup v,\, s(X \cup v)]$$

Et le résultat sera alors donné par $ST(K, k_0)$ pour $k_0 \in K$ quelconque.

Pour tout $v \in V$:

  Pour tout $w \in V$:

    $ST(w,v) \leftarrow [c(w,v), d(w,v)]$ (* $c(w,v)$ chemin réalisant $d(w,v)$ *)

    Fin Pour

  Fin Pour

Pour tout $X \in K$, par cardinal de X croissant :

  Pour tout $v \in V$:

    $(w_0, X_0) = \text{argmin}(d(v,w) + \min_{\emptyset \neq X' \not\subseteq X} s(X' \cup w) + s(X \backslash X' \cup w))$

    $ST(X,v) = [d(v,w_0) + s(X_0 \cup w_0) + s(X \backslash X_0 \cup w_0),\ c(v,w_0) \cup T(X_0 \cup w_0) \cup T(X \backslash X_0 \cup w_0)]$

    Fin Pour

  Fin Pour

Renvoyer $ST(K, k_0)$

Temps d'exécution en fonction du scénario :

$s1 : 0\ s$
$s2 : 0\ s$
$s3 : 0\ s$
$s4 : 0.0155\ s$
$s5 : 0.0523\ s$
$s6$: trop gros pour être testé vu la complexité en 3^k de l'algorithme de DW.

Question 11

La complexité d'une étape de calcul de $ST(X, v)$ est de $n.|\mathcal{P}(X)|$, chaque terme (n et $|\mathcal{P}(X)|$ étant la complexité de calcul d'un des deux min. La complexité totale est donc bornée par :

$$\begin{aligned} C_{tot} &= n^3 + \sum_{v \in V} \sum_{X \in K} n.2^{|X|} \quad \text{(* le terme en } n^3 \text{ vient de l'initialisation qui utilise Floyd-Warshall *)} \\ &= n^3 + n^2 \sum_{i=1}^{k} \binom{k}{i} 2^i \\ &= \mathcal{O}(n^3 + n^2 3^k) \end{aligned}$$

4 Algorithmes d'approximation

4.1 Heuristique du graphe des distances

Question 12

On utilise l'algorithme de Floyd-Warshall pour calculer le distance network D(K) (ainsi que la matrice P des plus courts chemins), l'algorithme de Kruskal pour calculer un arbre couvrant minimal sur D(K), et la fonction remonter(M, MST, P) pour remonter de l'arbre couvrant à un arbre de Steiner. Voici le pseudo-code de cette fonction :

Données :
Matrice d'adjacence $M(n \times n)$
Arbre minimal du distance network MST
Matrice des chemins P
Variable : $ST \leftarrow$ MatriceZeros$(n * n)$
Pour $1 \leq i \leq n$:
    Pour $i \leq j \leq n$: (* on utilise la symétrie *)
        Si $\text{MST}_{ij} \neq 0$:
            Pour (u, v) arête de P_{ij} :
                $ST_{uv} \longleftarrow M_{uv}$
                $ST_{vu} \longleftarrow M_{vu}$
                Fin pour
        Fin pour
    Fin pour
Return ST

D'où le pseudo-code de l'algorithme de Distance Network Heuristic :

Données : Matrice d'adjacence $M(n \times n)$, ensemble K

D $\leftarrow$ DistanceNetwork(M)
$P \leftarrow$ CheminsCourts(M)(D et P sont obtenus avec un algorithme de Floyd-Warshall)
$TD \leftarrow kruskal(D(K))$
TD $\leftarrow$ Remonter(M, MST, P)
T $\leftarrow$ kruskal(TD)
Enlever les non terminaux de degré 1
Return T

4.2 Heuristique des plus courts chemins

Question 13

On utilise l'algorithme de Dijkstra pour renvoyer le terminal le plus proche et le chemin correspondant.

Données : Matrice d'adjacence $M(n \times n)$, ensemble K

Variables : $T \leftarrow x \in K$

Tant que $K \nsubseteq T$:
    Ajouter à T le terminal t le plus proche de T :
    $T \leftarrow$ {Chemin de T à t} :
    $MST \leftarrow kruskal(T)$
    Enlever les non terminaux de degré 1 (dans MST) de T
Fin Tant Que
Return T

Question 14

L'algorithme DistanceNetworkHeuristic utilise (une fois) l'algorithme de Floyd-Warshall en $O(n^3)$ et (deux fois) l'algorithme de Kruskal en $O(n^2)$. La fonction remonter est clairement en $O(n^2)$ (parcours d'une matrice), de même que l'opération qui consiste à enlever les non-terminaux de degré 1. D'où une complexité polynomiale (en $O(n^3)$) pour l'algorithme Distance Network.

L'algorithme ShortestPathHeuristic ajoute au moins un terminal par itération à l'ensemble T, d'où au plus k passages dans la boucle while. A chacun de ces passages, la recherche du plus court chemin avec l'algorithme de Djikstra (en $O(n^2)$) et le calcul d'un arbre couvrant minimal avec l'algorithme de Kruskal (en $O(n^2)$), ainsi que l'opération qui consiste à enlever les non-terminaux de degré 1 (en $O(n^2)$) conduisent à une complexité en en $O(k * n^2)$ au total.

4.3 Analyse statistique

Question 15

Pour cette question, nous avons généré des graphes aléatoires et simulé sur ces graphes les deux algorithmes d'approximation. Nous n'avons pas pris $n = 500$ car le temps d'exécution était trop élevé, nous nous sommes restreints à tester deux valeurs, $n = 100$ et $n = 200$. Voici les résultats obtenus :

	t moyen ($n = 100$)	poids moyen ($n = 100$)	t moyen ($n = 200$)	poids moyen ($n = 200$)
SPH	1.73 s	20.5	45.45 s	61.25
DNH	1.54 s	20.5	11.82 s	61.25

Ainsi, si les deux algorithmes semblent avoir sensiblement les mêmes performances en termes de justesse, l'algorithme de DNH est nettement plus rapide, la différence augmentant avec la taille des données. Il est donc préférable de l'utiliser en pratique.