Axel Benyamine
Dimitri de Saint Guilhem

Projet Informatique - INF421

1 Introduction

L'Internet of things (IoT) se caractérise par une grande quantité d'objets connectés les uns avec les autres. Cependant la communication sans fil étant limitée par la distance on utilise des relais. Ces objets connectés et les relais forment un graphe dont les arêtes sont les connexions. On s'intéresse donc à la façon de relier efficacement les objets connectés. Le problème d'arbre Steiner minimal consiste à trouver un arbre de poids minimal reliant les objets souhaités entre eux, quitte à utiliser des relais.

Dans le problème, on considère un graphe $G = (V, E, c)$ et K un sous-ensemble de V de cardinal k. Nous avons choisi d'implémenter les algorithmes en python.

Par manque de temps et d'efficacité, nous n'avons pas traité la partie sur l'algorithme de Dreyfus-Wagner.

2 Structure de données.

Classe NodeList Nous avons créer une classe NodeList qui représente des noeuds pointant vers trois autres noeuds: le noeud précédent dans la liste, le noeud suivant et un noeud p.

Classe Graphe Pour représenter un graphe G, nous avons créer une classe. Celle-ci contient un tableau V qui est le tabelau des différents sommets (représentés par des int), un tableau E de NodeList qui sont des listes d'adjacences et une matrice de poids C correspondant aux arêtes. Celle-ci est principalement utilisée en temps constant (consulter des valeurs).

E est particulier. Pour un sommet $v \in V$, $E[v]$ contient une liste chaînée de NodeList. La première valeur est le sommet v lui-même (plus pratique pour gérer les listes vides). Ce premier élément noté u pointe vers un autre sommet qui est voisin. Le noeud p pour u est le noeud v qui apparaît dans la liste des voisins de u. C'est une structure qui apparaît en PC5.

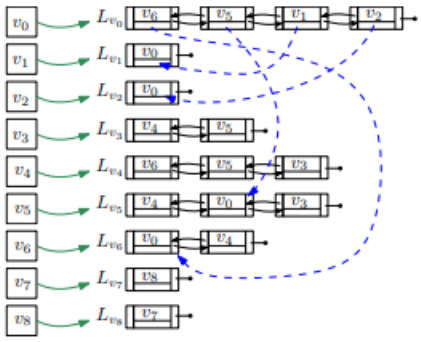

Figure 1: Structure de données pour les arêtes.

La classe Graphe posséde également un attribut D qui est la matrice des distances des point calculées grâce à une méthode Floyd qui utilise l'algorithme de Floyd vu en cours. Lorsqu'il est question du Grapphe correspondant dans D nous utilisons uniquement cette matrice, le tableau E étant inutile puisque le graphe dans D est complet. Nous avons modifié Floyd pour qu'il complète également un attribut SP qui une matrice

telle que $SP[i,j]$ contient le plus court chemin de i à j.

Nous avons codé des méthodes pour retirer des arêtes et des sommets, ce sont des transformation en place. Retirer une arête ou un sommet se fait en $O(\deg v)$, où v est le sommet en question ou l'un des sommets de l'arête. Pour faire cela on parcourt la NodeList contenue en E sans oublier de supprimer le sommet des autres listes grâce au dernier pointeur, il permet de le faire en temps constant. Si l'on retire un sommet, V réduit en taille, mais pas E car les valeurs des sommets stockées dans V sont utilisées pour indicer E.

3 Etapes de pré-traitement

Dans cette partie, on applique des etapes de pré-traitements pour réduire la taille d'un graphe donné dont on va chercher à résoudre le problème d'arbre de Steiner minimal. Dans cette partie, nous utilisons exclusivement la structure de données présentée ci-dessus.

Retirer les sommets non-terminaux de degré 1.

On considère une arête e de G dont une extrémité est s, un sommet non-terminal de degré 1 dans G. On suppose par l'absurde qu'il existe un arbre de Steiner minimal T pour K tel que $e \in T$.

On note $e = (s,t)$. On pose $T' = T \backslash \{e\}$, montrons que T' est un arbre de Steiner avec $c(T') < c(T)$.

1. Comme s est non terminal, on a $K \subset V(T')$.

2. Pour deux points v_1 et v_p de K reliés par un chemin $v_1v_2v_3...v_{p-1}v_p$ dans T, si e est emprunté alors st ou ts figure dans le chemin mais comme $v_1 \neq s$ et $v_p \neq s$ et que s est de degré 1, on a nécessairement la séquence tst qui apparaît dans le chemin à une position que l'on note i, ie. $v_iv_{i+1}v_{i+2} = tst$ et on peut remplacer $v_1...v_iv_{i+1}v_{i+2}...v_p$ par $v_1...v_iv_{i+3}...v_p$. On remplace autant de fois que e apparaît dans le chemin. Le chemin obtenu appartient bien à T' et relie v_1 et v_p.

Les deux points précédents montrent que T' est bien un arbre de Steiner. Comme on a de plus $c(T') = c(T) - c(e)$ et que $c(e) > 0$ ($e \in E$), on a $c(T') < c(T)$. Ainsi T n'est pas minimal ce qui est absurde.

On considère en entrée le graphe $G = (V, E, c)$ le sous-ensemble de sommet K on modifie G en place pour retirer le sommet concerné.

Algorithm 1 retirerDegré1

```
1: Entrée : un graphe G
2: for s dans V\K do
3:     if s est de degré 1 then
4:         Retirer s de G.
```

L'algorithme se justifie avec le paragraphe ci-dessus, au lieu de retirer simplement l'arête concernée, on retire le sommet puisqu'il n'est plus relié au graphe. Cela retire également l'arête.

Sommets non-terminaux de degré 2.

On considère un sommet v_k non-terminal de degré 2. On note $e_1 = (v_i, v_k)$, $e_2 = (v_k, v_j)$ ses deux arêtes et $e = (v_i, v_j)$ de poids infini si $e \notin E$. On veut montrer que si $c(e_1) + c(e_2) \geq c(e)$, il existe un arbre de Steiner minimal sans v_k et on peut alors le supprimer. Sinon, si on a $c(e_1) + c(e_2) < c(e)$ alors e n'appartient à aucun arbre de Steiner minimal et on peut le retirer.

On suppose $c(e_1) + c(e_2) < c(e)$.

Si $c(e) = +\infty$ alors e n'appartient pas à G et donc à aucun arbre de Steiner.

Sinon, on suppose qu'il existe un arbre de Steiner minimal T contenant e. En posant T' l'arbre obtenu en ajoutant e_1 et e_2 et en retirant e, on obtient un arbre de Steiner. En effet, on remplace e par e_1 puis e_2(ou l'inverse) dans les chemins. Comme $c(e_1) + c(e_2) < c(e)$, $c(T') < c(T)$ et donc T n'était pas minimal, ainsi e n'appartient à aucun arbre de Steiner minimal.

On suppose maintenant que $c(e_1) + c(e_2) \geq c(e)$.

On considère un arbre de Steiner minimal T. S'il ne contient pas v_k, c'est bon. Sinon, il ne peut contenir ni e_1, ni e_2 car v_k est non-terminal et T est un arbre de Steiner.

S'il en contient seulement un des deux, v_k est alors non-terminal pour K dans le graphe T et de degré 1 dans T. D'après la question 1, on peut le retirer. Un arbre de Steiner pour K dans T reste un arbre de Steiner pour K dans G puisque T es également un arbre de Steiner.

Si T contient e_1 et e_2, on peut lui ajouter e et lui retirer v_k. Il reste un arbre de Steiner puisque $v_k \notin K$ et que l'on peut toujours relier v_i et v_j. Il reste également minimal car $c(e_1) + c(e_2) \geq c(e)$.

Dans tous les cas on peut obtenir un arbre de Steiner minimal ne contenant pas v_k.

Algorithm 2 retirerDegré2

```
1: Entrée : G
2: for s dans V\K do
3:     if s est de degré 2 then
4:         On note v et u les deux voisins de s dans E[u].
5:         if C[v,u] > 0 (si v et u sont voisins) then
6:             if C[u,s] + C[v,s] ≥ C[u,v] then
7:                 Retirer s de G.
8:             else
9:                 Retirer e = (u,v) de G.
```

On applique simplement les conclusions du paragraphe précédent. On exploite la structure de E pour effectuer cela en temps constant pour chacun des sommets s.

Pour en retirer plus...

En reprenant les notations de la question 5 du problème :

On considère, s'il existe, une arête $e = (v_i, v_j) \in E$ telle que $c(e) > d(v_i, v_j)$. On suppose qu'il existe un arbre de Steiner T, tel que $e \in T$.

On note $u_0 u_1 ... u_p$ le chemin de v_i à v_j qui vérifie la distance $d(v_i, v_j)$, avec $u_o = v_i$ et $u_p = v_j$ c'est à dire les sommets tels que $d(v_i, v_j) = \sum_{\substack{k=0 \\ (u_k, u_{k+1}) \in E}}^{k} c(u_k, u_{k+1})$.

On pose $T' = T \cup \{(u_k, u_{k+1}), k \in [0, p]\} - \{e\}$. C'est évident que T' est toujours un arbre de Steiner : la seule arête retirée est remplacée par un chemin avec même début et même fin. De plus $c(T') \leq c(T) + d(v_i, v_j) - c(e) < c(T)$. Ainsi T ne peut pas être un arbre de Steiner minimal et on peut donc retirer e de T.

Algorithm 3 retirerSupplémentaire

```
1: Entrée : un graphe G
2: On calcule la matrice D de G grâce à la l'algorithme de Floyd.
3: for v ∈ V do
4:     for u voisin de v do
5:         if c(u,v) > D[u,v] then
6:             Retirer (u,v) de G.
```

On applique le résultat précédent à G en place. L'aglorithme de Floyd est implémenté en suivant le pseudo-code du cours.

L'ensemble des étapes de pré-traitement est donc la répétition de ces étapes jusqu'à convergence du graphe.

4 Algorithme d'énumération

Basé sur les trois points donnés dans l'énoncé, voici le pseudo-code de l'algorithme d'énumération.

Algorithm 4 enumeration

Entrée : un graphe G, l'ensemble K.
branchingPoints ← branchingPoints(G,K)
$T_{min}G$
$w_{min} \leftarrow +\infty$
for $i \in [1, k-2]$ **do**
    S ← ensemble des combinaisons de branching points de cardinal i
    for sommets ∈ S **do**
        T ← prim($D(G)$, sommets $\cup K$) (l'algorithme de Prim sur le sous graphe de l'ensemble des points de sommets pour le graphe de distance)
        w ← $c(T)$ (le poids dans D et dans G est le même)
        if $w < w_{min}$ **then**
            $w_{min} \leftarrow w$, $T_{min} \leftarrow T$
return T_{min}, w_{min}

La preuve de cet algorithme sera faite dans la présentation.

L'algorithme de Prim se fait en $O(|E|\log(|V|)$, donc chaque exécution dans l'algorithme ci-dessus est en $O((|sommets| + k)^2\log(|sommets| + k)$ car l'abre est complet et l'ensemble des sommets est sommets $\cup K$. Cependant les ensembles sommets sont de cardinal i. La comparaison et le calcul du coût ont une complexité inférieure. Ainsi, comme le nombre de branching points est inférieur à $|V| - k$ et qu'il y a $\binom{|V|-k}{i}$ itérations et ceci pour i allant de 0 à $k-2$, la complexité globale est en

$$O\left(\sum_{i=0}^{k-2}\binom{|V|-k}{i}(i+k)^2\log(i+k)\right)$$

En majorant brutalement, on peut obtenir une complexité exponentielle en $O(2^{|V|-k}k^3\log(k))$.

5 Algorithmes approximatifs

Distance Network Heuristic Dans notre code de la classe Graphe, la méthode Floyd rempli l'attribut D par la matrice des distances et l'attribut SP par la matrice des plus courts chemins. Cela permet de facilement coder une fonction de conversion :

Algorithm 5 conversion(G,T)

Entrée : une liste d'arête T (liste de tuple) et un graphe G dont T est une liste d'arêtes.
$V \leftarrow, E \leftarrow$
$w_{min} \leftarrow +\infty$
for $e \in T$ **do**
    On note $e = (i, j)$, et $S = G.SP[i, j]$ (les sommets du chemin de i à j).
    V ← $V \cup S$
    for $i \in [0, |S| - 1$ **do**
        Ajouter $(S[i], S[i + 1])$ à E (les sommets sont stockés dans l'ordre).
return V, T

On l'utilise dans Distance Network Heuristic.

Shortest-Path Heuristic

Complexités

Pour DNH, toutes les étapes précédant la boucle while sont en temps polynomial (Prim, conversion et copie). Dans la boucle while, on a la propriété suivante : si retirerDegré1(T) retire un nombre nul de sommet à T, la boucle s'interrompt. En effet, si cela se produit, il n'y aura pas de différence entre M et T, donc la boucle s'interrompera. Comme le nombre de sommet est positif et comme retirerDegré1 retire un nombre positif de sommet, le nombre de sommet est positif et strictement décroissant jusqu'à interruption de la boucle. Ainsi, la boucle s'interrompt après au plus le nombre de sommet de T a l'entrée de la boucle itérations, et donc au plus $|V|$ itérations. Il est donc polynomial.

Algorithm 6 DNH

Entrée : Un graphe G et un ensemble de sommets K de G.
On calcule D la matrice de distance des points de G et SP la matrice des plus courts chemins pour G, grâce à l'algorithme de Floyd.
$T_D \leftarrow \text{Prim}(D(K))$
$T_D \leftarrow \text{conversion}(G, T_D)$
T $\leftarrow \text{Prim}(T_D)$
M $\leftarrow$ copie(T)
while M $\neq$ T **do**
    M $\leftarrow$ copie(T)
    retirerDegré1(T)
return T

Algorithm 7 SPH

Entrée : Un graphe G et un ensemble de sommets K de G.
On calcule D la matrice de distance des points de G et SP la matrice des plus courts chemins pour G, grâce à l'algorithme de Floyd.
On fixe $x \in K, V_t \leftarrow \{x\}$
T un Graphe consitué du sommet x.
Q une file de priorité vide
for $i \in K \backslash \{x\}$ **do**
    On note j et d l'élement de V_t le plus proche de i, la distance associée.
    On ajoute à Q, le triplet (i,j,d) la clef étant d.
while Q n'est pas vide **do**
    $x, y, d \leftarrow$ extractmin(Q)
    On ajoute à V_t les sommets sur le chemin de x à y (grâce à SP).
    On met à jour Q : les distances à V_t et on retire les sommets de K qui auraient pu être ajoutés.
    On associe à T le MST associé au sous-graphe induit par lui même sur G.
    retirerDegré1(T)
return T

Pour SPH, tous les fonctions concernant la file de priorité sont au maximum en O(l) où l est la longueur de la file. Retrouver l'élément de V_t le plus proche est en O($|V|$) une fois la matrice de distance D calculée par Floyd. Celui-ci est polynomial. Les points précédents montrent que la boucle for est de complexité polynomiale. extractmin est linéaire, l'ajout des sommets également et le calcul du MST par Prim est polynomial. Comme la liste Q est de longueur k-1 à l'entrée de la liste et que sa taille réduit de 1 à chaque passage, la boucle s'interrompera en k-1 passage. Cela conserve la complexité polynomiale.