Routing in the Internet of Things

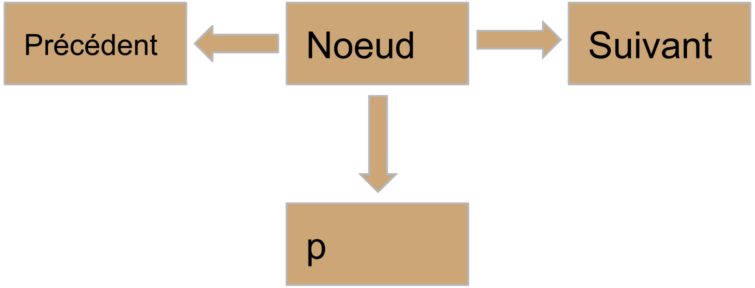
Axel Benyamine
Dimitri de Saint Guilhem

Implémentation initiale

Classe NodeList

Attributs :

- previous
- next
- p



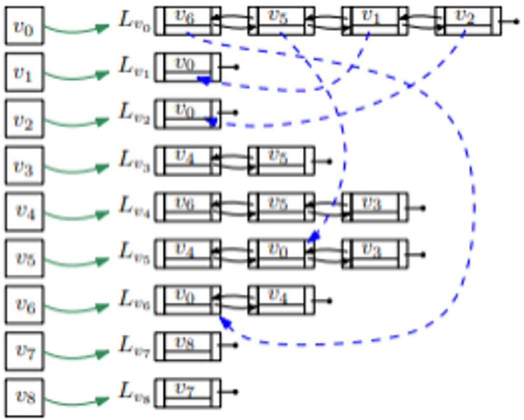
Méthodes :

- add, remove...
- supprBilateral (retire u et p)

Classe Graphe :

Attributs :

- V, E, C
- D et SP
- K



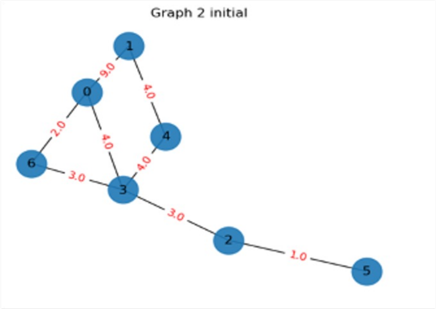
Méthodes:

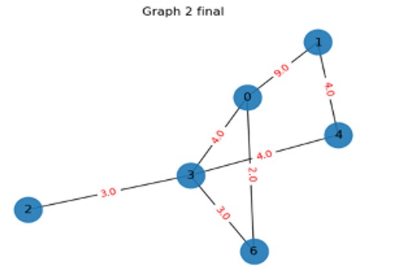
- degré, voisins
- retirerSommet, retirerArêtes
- Floyd
- remplissage

Pre-Processing Steps

Questions 1-4.

Retirer degré 1 :





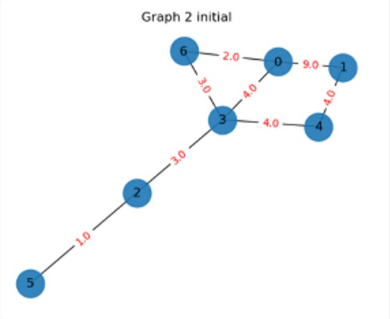
Algorithm 1 retirerDegré1

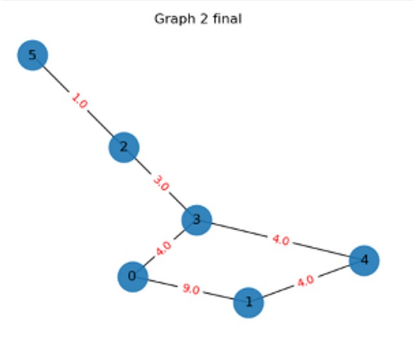
Entrée : un graphe G
for s dans $V \backslash K$ **do**
  if s est de degré 1 **then**
    Retirer s de G.

Algorithm 2 retirerDegré2

Entrée : G
for s dans $V \backslash K$ **do**
  if s est de degré 2 **then**
    On note v et u les deux voisins de s dans $E[u]$.
    if $C[v, u] > 0$ (si v et u sont voisins) **then**
      if $C[u, s] + C[v, s] \geq C[u, v]$ **then**
        Retirer s de G.
      else
        Retirer $e = (u, v)$ de G.

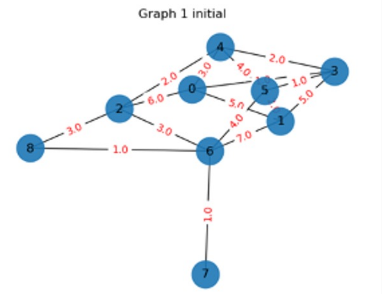
Retirer degré 2 :

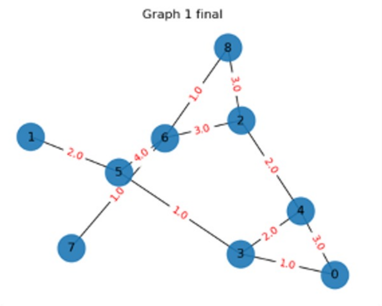




Questions 5-6 et fin du pre-Processing

Retirer Supplémentaire :





Algorithm 3 retirerSupplémentaire

Entrée : un graphe G
On calcule la matrice D de G grâce à la l'algorithme de Floyd.
for $v \in V$ **do**
    for u voisin de v **do**
        if $c(u,v) > D[u,v]$ **then**
            Retirer (u,v) de G.

Fin du pre-Processing :

Répétition des étapes Retirer degré 1, degré 2 et Supplémentaire jusqu'à test convergence.

Algorithme d’énumération

Algorithme d'énumération

La terminaison de l'algorithme est évidente.

La correction provient des trois points de l'énoncé :

- pour un ensemble *Sommets*, un MST du sous graphe *Sommets* U K est un arbre de Steiner pour K dans D.
- il existe un arbre de Steiner minimal vérifiant les conditions que vérifie l'ensemble des ensembles *Sommets*, comme on les parcourt tous, on trouve le minimum.

Algorithm 4 enumeration

Entrée : un graphe G, l'ensemble K.
branchingPoints $\leftarrow$ branchingPoints(G,K)
$T_{min}G$
$w_{min} \leftarrow +\infty$
for $i \in [1, k-2]$ **do**
  S $\leftarrow$ ensemble des combinaisons de branching points de cardinal i
  for sommets $\in$ S **do**
    T $\leftarrow$ 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 $\leftarrow c(T)$ (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}

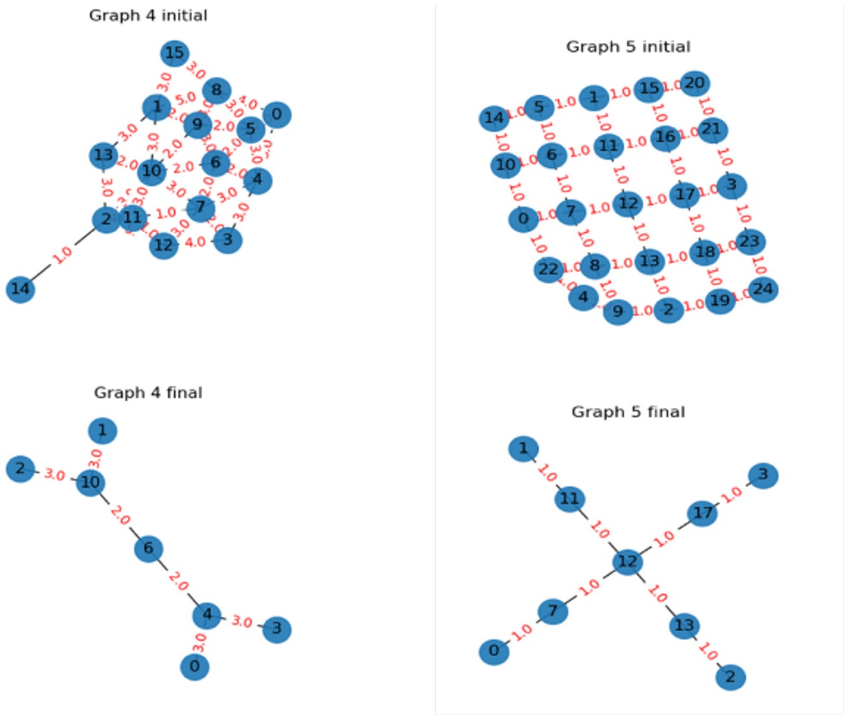
Complexité en :

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

Simulations d’énumérations

Temps de calcul (sans compter le pre-Processing) et poids des MST:

```
Poids du MST(1)
8.0
CPU time 1 : 0.000133514404296875


Poids du MST(2)
15.0
CPU time 2 : 9.5367431640625e-05


Poids du MST(3)
39.0
CPU time 3 : 0.00026702880859375


Poids du MST(4)
16.0
CPU time 4 : 0.0016241073608398438


Poids du MST(5)
8.0
CPU time 5 : 0.004148244857788086
```



Algorithmes approximatifs

Algorithme Distance Network Heuristic

- Complexité polynomiale par utilisation d'algos polynomiaux dans des boucles

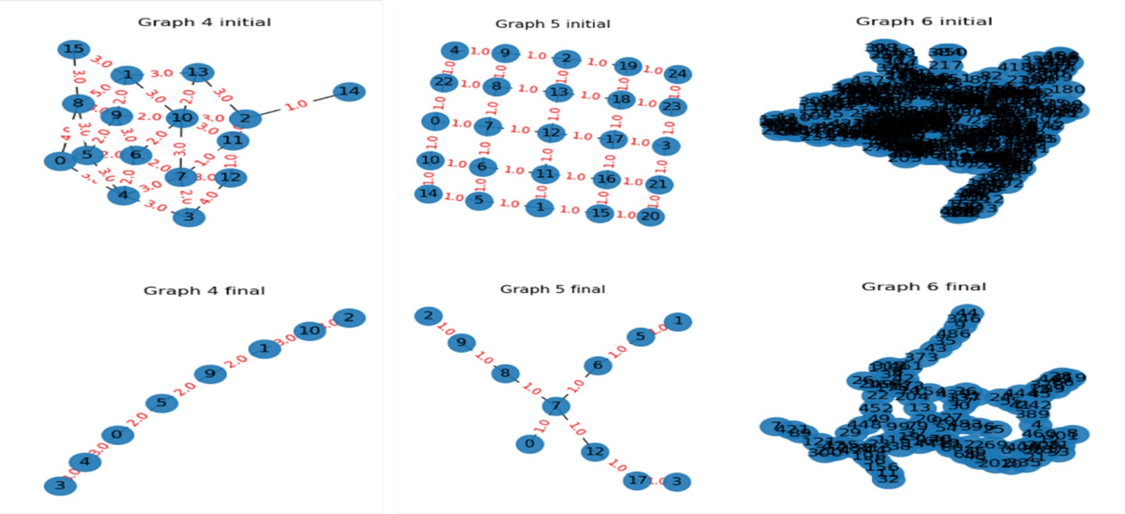
- Terminaison claire

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.
$T_D \leftarrow \text{Prim}(D(K))$
$T_D \leftarrow \text{conversion}(G, T_D)$
$\text{T} \leftarrow \text{Prim}(T_D)$
$\text{M} \leftarrow \text{copie}(T)$
while $\text{M} \neq \text{T}$ **do**
    $\text{M} \leftarrow \text{copie}(T)$
    retirerDegré1(T)
return T

Temps de calcul (sans compter le pre-Processing):

```
CPU time 1 : 0.00030612945556640625

CPU time 2 : 0.000141143798828125

CPU time 3 : 0.000293731689453125

CPU time 4 : 0.0003032684326171875

CPU time 5 : 0.0003304481506347656

CPU time 6 : 0.006083250045776367
```



Shortest-Path Heuristic

La terminaison est claire.

La complexité est bien polynomiale.

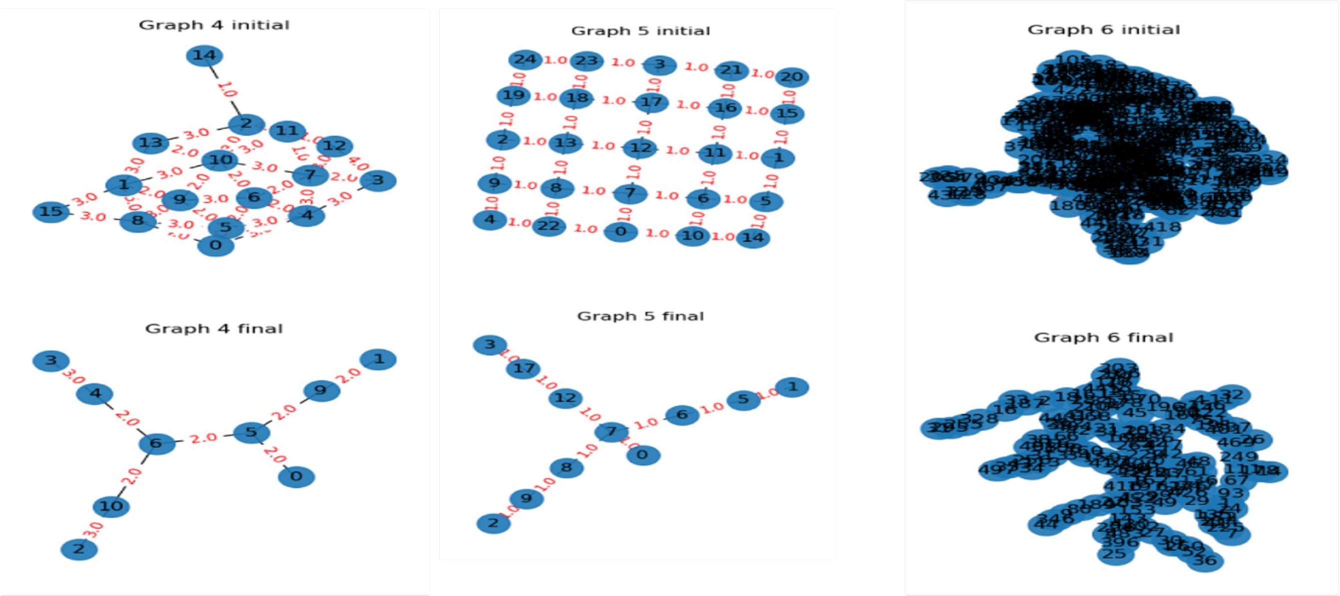
L'arbre retourné est bien un arbre de Steiner.

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 associe à T le MST associé au sous-graphe induit par lui même sur G.
  retirerDegré1(T)
return T

Temps de calcul (sans compter le pre-Processing):

```
CPU time 1 : 0.0001494884490966797

CPU time 2 : 0.00011467933654785156

CPU time 3 : 0.00018477439880371094

CPU time 4 : 0.00024437904357910156

CPU time 5 : 0.0003018379211425781

CPU time 6 : 0.23067927360534668
```



Comparaison des performances DNH et SPH

- Création d’un graphe de 22**2=484 points.
- Création d’un K aléatoire de 50 points parmis [0,483]
- Création d’une fonction distance (x=n//22 ; y =n%22)

Temps de calcul (sans compter la méthode Floyd(), commune aux deux algos):

```
CPU time moyen pour DNH : 0.005335688591003418

CPU time moyen pour SPH : 0.02108609676361084
```