CPA-2016 - Cours 5
Plus courts chemins entre tous les couples de points. Arbre de Steiner

M. Soria

1 Plus courts chemins entre tous les couples de points

$G = < S, A, w >$, graphe connexe non orienté, n sommets et m arêtes
G graphe valué : $w : A \rightarrow R+$ valuation positive sur les arêtes
Résolution du calcul des plus courts chemins entre tous les couples de points par *programmation dynamique*. Deux récurrences possibles :
- 1) longueur du chemin en nombre d'arêtes;
- 2) nombre de sommets par lesquels passe le chemin (Floyd-Warshall).
La première donne un algo en $O(n^4)$ (ou $O(n^3 \log n)$). La seconde donne un algo en $O(n^3)$.
Le graphe est représenté par la matrice des coûts $W = (w_{i,j})$

1.1 Allonger chemins

On part de la matrice $D^{(1)} = W$ et on calcule la suite de matrices $(D^{(1)}, \cdots, D^{(p)}, \cdots)$ telles que $D^{(p)}(i,j)$ contient la valeur d'un plus court chemin *de longueur* p entre i et j. Comme la longueur maximale des chemins est $n-1$ (chemins élémentaires), le résultat (PPDistances entre tous couples) est dans $D^{(n-1)}$.

Récurrence : $d^{(0)}(i,j) = 0$ si $i = j$ et $d^{(0)}(i,j) = +\infty$ si $i \neq j$.
Et pour $p \geq 1$ $d^{(p)}(i,j) = \min(d^{(p-1)}(i,j), \min_{k=1..n} d^{(p-1)}(i,k) + w_{k,j}) = \min_{k=1..n}(d^{(p-1)}(i,k) + w_{k,j})$.
L'algorithme 1 `AllongerChemin` calcule la matrice $d' = D^{(p)}$ en fonction de $d = D^{(p-1)}$ et W.
Sa complexité est en $O(n^3)$.

Algorithm 1 AllongerChemin(matrice d et w) si $d = D^{(m-1)}$ alors $d' = D^{(m)}$

for $i := 1$ to n **do**
  for $j := 1$ to n **do**
    $d'(i,j) \leftarrow maxInt$
    for $k := 1$ to n **do**
      $d'(i,j) \leftarrow \min(d'(i,j), d(i,k) + w_{k,j})$
    end for
  end for
end for
return d'

Les chemins étant élémentaires, aucun chemin de taille supérieure à $n-1$; donc finalement pour obtenir tous les plus petites distances, on a l'algorithme 2 `PPDtous`, qui calcule $D^{(n-1)}$ et dont la complexité est en $O(n^4)$.
Remarque On peut ramener la complexité à $O(n^3 \log n)$ en faisant le calcul de $D^{(p)}$ non pas séquentiellement mais par dichotomie.

Algorithm 2 PPDtous(matrice W)

$D^{(1)} \leftarrow W$
for $p := 2$ to $n-1$ **do**
  $D^{(p)} \leftarrow$ AllongerChemin$(D^{(p-1)}, w))$
end for
return $D^{(n-1)}$

1.2 Floyd-Warshall

On part de la matrice $D^{(0)} = W$ et on calcule la suite de matrices $(D^{(0)}, \cdots, D^{(p)}, \cdots)$ telles que $D^{(p)}(i,j)$ contient la valeur d'un plus court chemin entre i et j, passant uniquement par des sommets de $\{1, \cdots, p\}$ (les sommets du graphes sont étiquetés par $1, \ldots, n$).

Récurrence :$d^{(0)}(i,j) = w_{i,j}$
et pour $k \geq 1$: $d^{(k)}(i,j) = \min(d^{(k-1)}(i,j), d^{(k-1)}(i,k) + d^{(k-1)}(k,j))$.
La matrice des distances minimales entre tous les couples de points est calculée par l'algorithme 3 (on utilise une seule matrice D, initialisée à W et qui contient à chaque étape k les PPD passant par des points de $\{1, \cdots, k\}$).

Algorithm 3 Floyd-Warshall(matrice W)

```
D ← W
for k := 1 to n do
  for i := 1 to n do
    for j := 1 to n do
      D(i,j) = min(D(i,j), D(i,k) + D(k,j))
    end for
  end for
end for
return D
```

Calcul des chemins Si l'on veut calculer les plus courts chemins eux-mêmes et pas seulement leurs valeurs, il faut ajouter une matrice de liaison P, où $P(i,j)$ vaut 0 si $i = j$ ou s'il n'existe aucun chemin de i vers j ; sinon $P(i,j)$ contient le prédécesseur de j sur un plus court chemin issu de i - Voir algorithme 4.

Algorithm 4 Floyd-Warshall-Liaison(matrice W)

```
D ← W ; P ← 0
for k := 1 to n do
  for i := 1 to n do
    for j := 1 to n do
      if D(i,k) + D(k,j) < D(i,j) then
        D(i,j) ← D(i,k) + D(k,j); P(i,j) ← P(k,j)
      end if
    end for
  end for
end for
return D, P
```

Et pour lister les plus courts chemins on parcourt la matrice P – voir algorithme 5.

Algorithm 5 ListerPPC($i, j, L = \emptyset$)

```
if i=j then
  L ← add(i, L)
else
  ListerPPC(i, P(i,j), L)
  L ← add(j, L)
end if
return L
```

2 Arbre de Steiner

2.1 Principe

$G =< S, A >$, graphe non orienté, connexe, valué sur les arcs $w : A \to R+$, et soit $V \subset S$ tel que $|V| = k$. Touver un arbre de coût minimum couvrant V (et qui passe éventuellement par d'autres points de S).

Propriété : *Le problème de l'arbre de Steiner est NP-difficile, même si tous les poids sont égaux à 1.*

Remarque : On peut aussi présenter le pb de l'*arbre de Steiner dans le plan* : étant donné un ensemble V de k points du plan, trouver un arbre de coût minimum couvrant V (et qui passe éventuellement par d'autres points du plan). Le coût d'une arête est alors la distance dans le plan entre ses extrémités. Cela est aussi un problème NP-difficile (voir cours suivant).

2.2 Heuristique

On étudie tout d'abord une heuristique, basée sur des calculs de PPC et des recherches d'ACM, qui fournit une solution de poids au plus 2 fois plus grand que la solution optimale pour l'arbre de Steiner.

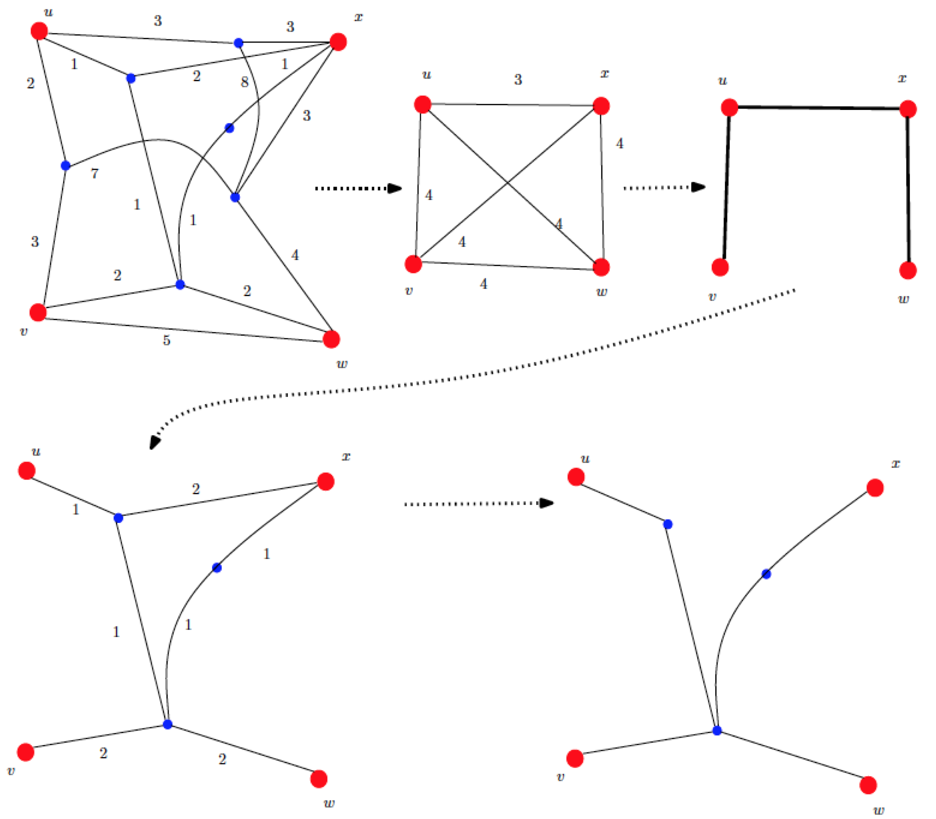
Algorithm 6 Fonction SteinerHeuristique ($G =< S, A, w >, V$)

1. Construire le graphe complet $K =< V, A' >$ tq. sur chaque arête (x, y) de $V \times V$, on met le poids du PPC de x à y dans G : $w'(x, y) = PPC_G(x, y)$
2. Construire un ACM T_0 couvrant $K =< V, A', w' >$
3. Dans T_0, remplacer les arêtes (x, y) par les chemins $PPC_G(x, y)$; on obtient alors un graphe partiel $H =< V \cup St, A'', w >$ de G (St est l'ensemble des points de Steiner ajoutés).
4. Construire un ACM T' couvrant H
return T'

Remarque : on peut optimiser les 2 dernières étapes : lorqu'on remplace les arêtes (x, y) par les chemins $PPC_G(x, y)$, si on rajoute un chemin qui contient 2 points u et v qui sont deja dans l'arbre en construction, alors on ne rajoute pas tous les points du chemin de x à y, mais seulement les points de x à u et les points de y à v (pour éviter d'introduire un cycle). Mais cela ne change pas la complexité totale de l'algo, qui vient du calcul du graphe complet (étape 1).

Complexité : $O(n^3)$ pour la construction du graphe complet K des PPC (qui a $k(k-1)/2 = O(k^2)$ arêtes, où $k = |V|$). Puis $O(k^2 \log k)$ pour la construction de l'ACM T_0 de ce graphe K. Puis "developper" T_0 (qui a $(k-1)$ arêtes) en H prend un temps $O(kn)$. le calcul de l'ACM de H (qui a $O(kn)$ aretes ... mais en général beaucoup moins !) est en $O(kn \log(kn))$. D'ou complexité totale en $O(n^3)$.

Propriété : *L'algorithme* `SteinerHeuristique` *fournit une solution de poids au plus 2 fois plus grand que la solution optimale pour l'arbre de Steiner.*
preuve : Soit T l'arbre de Steiner de plus petit poids. En dupliquant toutes les arêtes de T on obtient un cycle Eulerien C (en partant d'un sommet quelconque et en empruntant exactement une fois chaque arête pour revenir au sommet de départ). Cela induit un cycle hamiltonien C' (passant une unique fois par tous les sommets) dans le graphe complet K de V : on parcourt l'ensemble des sommets de K dans l'ordre de leur première apparition dans C. Donc finalement $w(T') \leq w'(C') \leq w(C) = 2w(T)$.


u
x
v
w
3
3
1
2
1
2
8
3
7
1
1
3
2
2
4
5
u
x
v
w
3
4
4
4
4
4
u
x
v
w
u
x
v
w
2
1
1
1
1
2
2
u
x
v
w