CPA-2016 - Cours 4
Arbres couvrants minimaux et plus courts chemins

M. Soria

1 Présentation du bloc de 3 semaines

3 semaines consacrées à la recher de structures optimales dans un graphe valué :
1. Arbres couvrants minimaux (Prim, Kruskal)
2. Plus courts chemins (Dijkstra, Floyd-Warshall)
3. Arbres de Steiner (programmation dynamique + heuristiques)

Notion de valuation : $G =< S, A, w >$, avec $w : A \to R+$. Valuation additive (le poids d'un chemin est la somme des poids des arêtes)
Graphe non valué : le poids de chaque arête vaut 1
Graphe euclidien : ensemble de points dans le plan ; on suppose que le graphe est complet et la valuation entre 2 points est la distance euclidienne.
Graphe géométrique : ensemble de points dans le plan ; 2 points sont reliés si leur distance euclidienne est inférieure à un certain seuil, et la valuation est cette distance.

Complexités Pour les arbres couvrants minimaux on a des algorithmes très efficaces ($m \log m$), donc on peut traiter des graphes jusqu'à 10^8 arêtes. Algorithmess gloutons. Pareil pour Dijkstra.
Pour les plus courts chemins entre tous les couples de points; programmation dynamique, en $O(n^3)$. Graphes traitables pour n jusqu'à 10^3 sommets.
Pour les arbres de Steiner le probleme est NP-difficile !!!!! meilleur algorithme exact est exponentiel 2^n ou 3^n, ok pour n jusqu'a 30. C'est aussi l'occasion de travailler sur des algorithmeq approchés ou sur des heuristiques.

2 Arbres couvrants minimaux

2.1 Principe

$G =< S, A, w >$, graphe connexe non orienté, n sommets et m arêtes
G graphe valué : $w : A \to R+$ valuation positive sur les arêtes

Définition : Un *arbre couvrant minimum, ACM*, est un sous-graphe connexe sans cycle $T =< S, B >$, qui minimise $w(T) = \sum_{(x,y)\in B} w(x, y)$
Existence ACM assurée (enlever les aretes qui forment des cycles) Pas unicité de ACM (sauf si toutes valuations différentes)

Calcul d'un ACM par stratégie gloutonne : la suite des minimums locaux (ajout d'une arête minimale à chaque étape) produit un minimum global.
On fabrique à chaque étape un sous-ensemble de l'ACM (arête qui ne *compromet pas* B = qui fait que B est toujours un sous-ensemble de l'ACM)

Définition : Une *coupe* de $G =< S, A >$ est une partition $(R, S - R)$ de l'ensemble des sommets de G. Une arête (x, y) *traverse* la coupe si l'une de ses extrémités est dans R et l'autre dans $S - R$. Une coupe *respecte* un ensemble d'arêtes B si aucune arête de B ne traverse la coupe.

Algorithm 1 Fonction ACM(G)

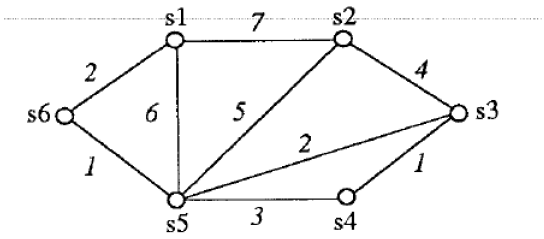
$B \leftarrow \emptyset$
while B non arbre couvrant (n-1 étapes) **do**
  Soit x,y une arête minimale qui ne compromet pas B
  $B \leftarrow B \cup (x, y)$
end while
return B

Propriété : *Soit $G =< S, A, w >$, et soit B un sous-ensemble de A inclus dans un ACM de G ; soit $(R, S - R)$ une coupe de G qui respecte B et soit (x, y) une arête minimale traversant $(R, S - R)$. Alors $B \cup (x, y)$ est inclus dans un ACM de G.*

Preuve. *Soit T un arbre couvrant minimum qui inclut B ; si terminé; sinon on construit un autre arbre couvrant minimum T' qui inclut $B \cup (x, y)$: L'arête (x, y) forme un cycle dans T avec les arêtes du chemin p de x à y. Puisque x et y se trouvent de chaque côté de la coupe $(R, S-R)$, il existe au moins une arête de T sur le chemin p qui traverse aussi la coupe. Soit (z, t) une telle arête. Elle n'est pas dans B, car la coupe respecte B. On forme le nouvel arbre couvrant $T' = T - (x, y) + (z, t)$, qui est un ACM : Puisque (x, y) est une arête minimale traversant $(R, S - R)$ et que (z, t) traverse aussi cette coupe, on a $w(x, y) \leq w(z, t)$*

On présente deux algo-ACM qui reposent sur cette propriété :

- Algo de Prim : B est un sous-arbre à la racine de $ACM(G)$; la coupe s'appuie sur les sommets de B.
- Algo de Kruskal : B est un ensemble de sous-arbre de $ACM(G)$; la coupe s'appuie sur les sommets de l'un de ces sous-arbres.



et faire w(2,5) = w(4,5) = 2

Plusieurs arbres couvrants minimaux (5 aretes et choisir parmi valuations 1,1, 2,2,2,2) Il devrait y avoir 4 choix (3 parmi les quatre 2) mais on ne peut pas avoir (4,5) et (3,5) car sinon ca fait un cycle avec l'arete (3,4) valuee 1. Donc il ne reste que 2 choix (on est obligé de garder (1,6) et (2,5))

2.2 Kruskal

voir algorithme 2.

Complexité : Tri des arêtes : $O(m \log m)$. Ensuite pour chaque arête (m fois) verifier si son ajout introduit un cycle : n sommets colores selon composante connexe $\rightarrow$ coût $O(n)$ pour verifier si cycle donc coût total $O(nm)$, mais si on utilise une structure d'union-find optimisée on a cout $O(\log n)$ donc coût total $O(m \log n)$.

Algorithm 2 Fonction ACM-Kruskal(G)

$X \leftarrow \emptyset$; $i \leftarrow 1$
Trier les aretes en ordre croissant des poids : $w(u_1) \leq \cdots \leq w(u_m)$
while $i \leq m$ **do**
  if $X \cup u_i$ est sans cycle **then**
    $X \leftarrow X \cup \{u_i\}$; $i \leftarrow i + 1$
  end if
end while
return X

2.3 Prim

Définition : Soit S' un ensemble de sommets, le *cocycle* de S' est $\omega(S') = \{(x, y) \in A; x \in S' \text{ et } y \notin S'\}$

Algorithm 3 Fonction ACM-Prim(G)

$X \leftarrow \emptyset$
Soit $x \in S$; $S' \leftarrow \{x\}$
while $\omega(S') \neq \emptyset$ $\quad (\omega(S') = \{(x, y) \in A; x \in S' \text{ et } y \notin S'\})$ **do**
  choisir $u = (y, z) \in \omega(S')$ de poids minimum
  $S' \leftarrow S' \cup \{y, z\}$; $X \leftarrow X \cup \{u\}$
end while
return X

Complexité : On gère l'ensemble des sommets $S - S'$ sous forme d'un tas. La priorité d'un sommet x est la valeur minimale d'une arête (x, y), où $y \in S'$. Chaque arête (il en a m) permet d'évaluer au plus un sommet du tas (donc de décrémenter S') et les operation sur le tas se font en $O(\log n)$ donc complexité $O(m \log n)$.

2.4 Implantation

2.4.1 Représentation des graphes

Orientés ou non orientés

1. Représentation par la liste des arêtes (et eventuellement aussi la liste des sommets isolés)
   - place $O(n + m)$
   - on peut trier la liste par sommet initial si graphe orienté (sinon lister 2 fois chaque arete pour que chaque sommet soit suivi de tous ses voisins)
   - traitement des sommets adjacents à s en $O(degre(s))$, si la liste est triée par sommet sinon $O(m)$
   - traitement global des arêtes en $O(m)$.
2. Représentation par matrice $n \times n$
   - $M(i, j)$ contient un booléen ou une valuation,

- place $O(n^2)$,
- traitement des sommets adjacents à s en $O(n)$,
- traitement global des arêtes en $O(n^2)$.

3. Représentation par n listes d'adjacence
   - chaque sommet lié à la liste de ses sommets adjacents (avec val),
   - place $O(n+m)$,
   - traitement des sommets adjacents à s : $O(degre(s))$,
   - traitement global des arêtes en $O(m)$.
4. si le graphe est un arbre, representation hierarchisée habituelle

   Passer d'une représentation en liste d'arêtes à une représentation en arbre à partir de la racine r:
   Appeler `ListeToArbre(L,r)` avec
   Function ListeToArbre(L,x) renvoie un arbre
   For each x_i voisin de x, $A_i \leftarrow ListeToArbre(L, x_i)$
   Return $< x, A_1, \cdots, A_k >$
   End Function

2.4.2 Structures de données

- Union-find pour Kruskal
- File de priorité-Tas pour Prim

3 Plus courts chemins

3.1 Principe

$G =< S, A >$, graphe orienté (ou non), connexe, valué sur les arcs $w : A \to R+$

- PCC d'un point source à tous les autres (ou 1 autre) : algorithmes gloutons Dijkstra ($w : A \to R+$) (aussi Bellman si $w : A \to R$).
- PCC entre tous couples de points : prog. dynamique Floyd-Warshall (complexité $O(n^3)$) ou Dijkstra à partir de tous points (d'où complexité $O(n \times m \log n)$).

Circuits

- circuit de poids négatif (à partir de la source) alors PAS de chemin minimal
- un chemin minimal ne contient pas de circuit de poids positif ou nul

Remarque : L'ensemble des PPC d'une source à tous les autres points est une arborescence qui n'est pas forcément un ACM
ex :$a--b--c--d$ avec $w(a,b) = w(b,c) = w(c,d) = 1$ et $w(a,d) = 2$
Propriété : *Si $g = x_0, x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n$ est un chemin minimal de x_0 à x_n, alors tout sous-chemin de g entre x_i et x_j est un chemin minimal de x_i à x_j.*

3.2 Dijkstra

Algorithm 4 Fonction PCC-Dijkstra(x, G)

$Dist[1..n] \leftarrow \infty$; $Parent[1..n] \leftarrow 0$
$S' \leftarrow \{x\}$; $Dist[x] \leftarrow 0$
while $\omega(S') \neq \emptyset$ $\quad (\omega(S') = \{(x,y) \in A; x \in S' \text{ et } y \notin S'\})$ **do**
  choisir $u = (y,z) \in \omega(S')$ de poids minimum (avec $y \in S'$ et $z \notin S'\}$)
  for all voisin t de z tq $t \notin S'$ **do**
    if $Dist[t] > Dist[z] + w(z,t)$ **then**
      $Dist[t] \leftarrow Dist[z] + w(z,t)$; $Parent[t] \leftarrow z$
    end if
  end for
  $S' \leftarrow S' \cup \{y,z\}$
end while
return $Dist[1..n]$, $Parent[1..n]$

Complexité : la même que Prim