Programmation Effective – TD 03 : Détections de cycles de poids négatifs

matthieu.gallet@ens-lyon.fr

mardi 12 février 2008

1 Définitions, contexte

Le problème de la détection de cycle de poids négatif est très lié à celui du calcul du plus court chemin à origine unique. Dans ce TD, nous considérons un graphe orienté pondéré $G = (S, A)$, avec une fonction de pondération $w : A \mapsto \mathbb{R}$. Depuis un sommet origine $s \in S$ donné, on cherche à trouver un plus court chemin de s vers n'importe quel autre sommet $v \in S$. La **longueur** du chemin $p = (v_0, v_1, \ldots, v_k)$ est la somme des poids (longueurs) des arcs qui le constituent : $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$. La longueur du plus court chemin entre u et v par

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \text{s'il existe un chemin de } u \text{ vers } v, \\ +\infty & \text{sinon.} \end{cases}$$

Un plus court chemin de u à v est donc un chemin p de longueur $w(p) = \delta(u, v)$.

Un lemme intéressant : Les sous-chemins de plus courts chemins sont des plus courts chemins.

Arcs de poids négatifs : La fonction de pondération est à valeurs dans $\mathbb{R}$, rien n'empêche donc les arcs d'avoir des poids strictement négatifs. De fait, s'il y a un circuit de poids total strictement négatif, pour certains couples de sommets, il n'existera pas de plus court chemin, car le minimum est atteint en parcourant une infinité de fois le cycle de poids strictement négatif. Pour pallier ce problème, soit les algorithmes supposent que tous les poids sont positifs (comme l'algorithme de Dijkstra), soit les algorithmes peuvent tout de même fonctionner, renvoyant une erreur uniquement en cas de circuit de poids strictement négatif (comme celui de Bellman-Ford que nous allons voir).

Circuits : Un plus court chemin ne contient aucun circuit de poids strictement positif, car sinon on peut facilement diminuer sa longueur facilement. En revanche, un plus court chemin peut contenir des circuits de poids nul, mais on peut facilement les éliminer. Ainsi, on peut supposer que l'on ne traitera que des plus courts chemins sans circuit, cela permet d'assurer que *les plus courts chemins contiennent au plus* $|S| - 1$ *arcs*.

Représentation des plus courts chemins : La méthode la plus simple pour représenter un plus court chemin, et qui fait appel au lemme précédent, est d'utiliser un tableau π telle que $\pi[v]$ contienne le prédécesseur de v dans un plus court chemin de s à v. Si le prédécesseur n'est pas (encore) défini, $\pi[v]$ vaut NIL.

Sous-graphe prédécesseur : il s'agit du graphe $G_\pi = (S_\pi, A_\pi)$ induit par les valeurs de π au cours de l'exécution d'un algorithme. On le définit ainsi :

$$S_\pi = \{v \in S : \pi[v] \neq \text{NIL}\} \cup \{s\},$$

$$A_\pi = \{(\pi[v], v) \in A : v \in S_\pi - \{s\}\}.$$

On peut montrer qu'une fois l'algorithme terminé, G_π est un arbre de racine couvrant de G, de racine s, qui permet de rejoindre n'importe quel sommet de S en partant de s en suivant un plus court chemin.

Relâchement : Il s'agit d'une technique utilisée par Bellman-Ford pour calculer ces fameux plus courts chemins. Pour chaque sommet $v \in S$, on gère un attribut $d[v]$ qui est un majorant (et une estimation) de la longueur d'un plus court chemin entre s et v. Avant le début de l'algorithme, cette estimation vaut $+\infty$ pour tous les sommets, sauf s pour qui elle vaut 0. Le pseudo-code de cette initialisation est donné par l'algorithme 1.

Algorithm 1 INITIALISATION(S, s)

pour chaque sommet $v \in S$ **faire**
  $d[v] \leftarrow +\infty$
  $\pi[v] \leftarrow$ NIL
fin pour
$d[s] \leftarrow 0$

Le relâchement d'un arc (u, v) consiste à tester si l'on peut améliorer le plus court chemin vers v en passant par u. Si c'est le cas, on met à jour $\pi[v]$ et $d[v]$. Le pseudo-code est donné par l'algorithme 2.

Algorithm 2 RELÂCHER(u, v, w)

si $d[v] > d[u] + w(u, v)$ **alors**
  $d[v] \leftarrow d[u] + w(u, v)$
  $\pi[v] \leftarrow u$
finsi

2 Algorithme de Bellman-Ford

Il s'agit d'un des algorithmes les plus classiques de calcul de plus court chemin, avec Dijkstra. Il a l'avantage de fonctionner même en présence d'arcs de poids négatifs, ce qui nous intéresse pour les problèmes d'aujourd'hui. L'algorithme va diminuer progressivement une estimation $d[v]$ du poids d'un plus court chemin depuis l'origine s vers chaque sommet $v \in S$ jusqu'à atteindre la valeur réelle du poids de plus court chemin $\delta(u, v)$. L'algorithme renvoie VRAI si, et seulement si, le graphe ne contient aucun graphe de longueur strictement négative accessible depuis l'origine. Le pseudo-code complet de cet algorithme est donné par l'algorithme 3. Cet algorithme s'exécute en temps $O(|S||A|)$.

3 Plus courts chemins à origine unique dans un graphe orienté sans circuit

Dans cette section, nous supposons que les graphes sont sans circuit. Ainsi, même avec des poids négatifs, les plus courts chemins sont toujours correctement définis. Cette nouvelle contrainte permet d'obtenir un algorithme en temps $O(|S| + |A|)$. Le pseudo-code est donné par l'algorithme 4.

4 Dijkstra : avec uniquement des poids positifs

Je rappelle quand même le pseudo-code avec l'algorithme 5, qui tourne en temps $O\left((|S| + |A|)log|S|\right)$ en utilisant des files de priorité.

Algorithm 3 BELLMAN-FORD(G, w, s)

INITIALISATION(G, s)
pour $i \leftarrow 1$ à $|S| - 1$ **faire**
  pour chaque arc $(u, v) \in A$ **faire**
    RELÂCHER(u, v, w)
  fin pour
fin pour
pour chaque arc $(u, v) \in A$ **faire**
  si $d[v] > d[u] + w(u, v)$ **alors**
    renvoyer FAUX
  finsi
fin pour
renvoyer VRAI

Algorithm 4 PLUS-COURTS-CHEMINS-GSS(G, w, s)

trier topologiquement les sommets de G
INITIALISATION(G, s)
pour chaque sommet u dans l'ordre topologique **faire**
  pour chaque sommet $v \in Adj[u]$ **faire**
    RELÂCHER(u, v, w)
  fin pour
fin pour

Algorithm 5 DIJKSTRA(G, w, s)

INITIALISATION(G, s)
$E \leftarrow \emptyset$
$F \leftarrow S$
tantque $F \neq \emptyset$ **faire**
  $u \leftarrow \text{EXTRAIRE} - \text{MIN}(F)$
  $E \leftarrow E \cup \{u\}$
  pour chaque sommet $v \in Adj[u]$ **faire**
    RELÂCHER(u, v, w)
  fin pour
fin tantque

5 Autres variantes du problème

Plus court chemin à destination unique : Trouver un plus court chemin vers une destination t à partir de n'importe quel sommet v. Il suffit évidemment d'inverser le problème précédent :o

Plus court chemin pour un couple de sommets donné : ce problème est un cas particulier du cas étudié dans ce TD. On ne connaît aucun algorithme qui soit asymptotiquement meilleur que les meilleurs algorithmes à origine unique dans le pire des cas.

Plus court chemin pour tout couple de sommets : Trouver un plus court chemin de u à v pour tout couple de sommets u et v. Ce problème peut être résolu en exécutant un algorithme à origine unique pour chaque sommet. Cependant, on peut généralement le résoudre plus rapidement, par exemple avec l'algorithme de Floyd-Warshall qui suppose qu'il n'existe pas de cycle de longueur strictement négative.

Pour cet algorithme, nous avons besoin de quelques nouvelles définitions :

matrice de poids des arcs : cette matrice $W = (w_{ij})$ est définie par

$$w_{ij} = \begin{cases} 0 & \text{si } i = j, \\ w(i,j) & \text{si } i \neq j \text{ et } (i,j) \in A, \\ +\infty & \text{si } i \neq j \text{ et } (i,j) \notin A, \end{cases}$$

matrices de plus courts chemins : $D^{(k)} = (d_{ij}^{(k)}$ est la matrice des poids des plus courts chemins de i à j dont tous les sommets intermédiaires sont dans l'ensemble $\{1, \ldots, k\}$. Pour $k = 0$, il s'agit des chemins sans sommet intermédiaires, donc des arcs. La matrice finale $D^{(n)}$ donne le résultat final $d_{ij}^{(n)} = \delta(i,j)$ pour tout $i, j \in S$.

Le pseudo-code complet de cet algorithme est donné par l'algorithme 6.

Algorithm 6 FLOYD-WARSHALL(W)

```
n ← lignes[W]
D^(0) ← W
pour k ← 1 à n faire
  pour i ← 1 à n faire
    pour j ← 1 à n faire
      d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
    fin pour
  fin pour
fin pour
renvoyer D^(n)
```

6 Application

résolvez les problèmes suivants : `http://acm.uva.es/p/v5/558.html`, `http://acm.uva.es/p/v5/515.html`.