Programmation Effective – TD 02 : Flot maximum

matthieu.gallet@ens-lyon.fr

mardi 5 février 2008

1 Définitions, contexte

Réseau de transport : Un réseau de transport $G = (S, A)$ (S est l'ensemble des sommets – *vertices* – et A l'ensemble des arêtes – *edges*) est un graphe orienté dans lequel chaque arc $(u, v) \in A$ se voit attribuer une capacité $c(u, v) \geq 0$. Si $(u, v) \notin A$, on supposera $c(u, v) = 0$. Deux sommets sont différents des autres : la source s et le puit (*sink*) t, et on suppose que les autres sommets se trouvent toujours sur un chemin reliant la source au puit. Le graphe est donc connexe et on a $|A| \geq |S| - 1$.

Flot : Soit $G = (S, A)$ un réseau de transport avec une capacité c, de source s et de puit t. Un flot de G est une fonction $f : S \times S \to \mathbb{R}$, respectant les conditions suivantes :
capacité : Pour tout $u, v \in S$, on a $f(u, v) \leq c(u, v)$,
symétrie : Pour tout $u, v \in S$, on a $f(u, v) = -f(v, u)$,
conservation : Pour tout $u \in S - \{s, t\}$, on a $\sum_{v \in S} f(u, v) = 0$
La valeur d'un flot f est donnée par $|f| = \sum_{v \in S} f(s, v)$.

Réseaux à sources et puits multiples : On peut très bien imaginer avoir tout un ensemble de sources $\{s_1, \ldots, s_m\}$ et de puits $\{t_1, \ldots, t_n\}$. En fait, le problème n'est pas plus compliqué que le problème précédent : on ajoute au graphe une supersource s relié aux sources originales par des arcs de capacité infinie. De la même façon, on relie les puits originaux à un superpuit t par des arcs de capacité infinie.

2 Méthode de Ford-Fulkerson

En général, on cherche à déterminer quel est le flot maximum qu'on peut trouver dans un graphe donné. Ford et Fulkerson ont donné une méthode très générale (Algorithme 1) pour résoudre ce problème, méthode pouvant être implémentée de diverses façons.

Nous allons avoir besoin de quelques autres définitions :

Flots résiduels : Intuitivement, c'est la capacité inutilisée du réseau. La capacité résiduelle de (u, v) est donnée par $c_f(u, v) = c(u, v) - f(u, v)$. Le réseau résiduel induit sur G par f est donné par $G_f = (S, A_f)$, où $A_f = \{(u, v) \in S \times S : c_f(u, v) > 0\}$. On remarque que dans les réseaux résiduels, les liens utilisés au maximum de leurs capacités ne sont pas présents.

On montre que si f' est un flot de G_f, alors la somme $f + f'$ est un flot de G de valeur $|f + f'| = |f| + |f'|$.

Algorithm 1 Méthode de Ford-Fulkerson

initialiser f à 0
tantque il existe un chemin améliorant p **faire**
  augmenter f le long de p
fin tantque
retourner f

Chemins améliorants : Étant donnés un réseau de transport $G = (S, A)$ et un flot f. Un chemin améliorant p est un chemin élémentaire de s vers t dans le réseau résiduel G_f. La plus grande capacité de flux transportable à travers les arcs d'un chemin améliorant p s'appelle la capacité résiduelle de p et est définie par $c_f(p) = \min\{c_f(u, v) : (u, v) \in p\}$.

On montre que si $G = (S, A)$ est un réseau de transport, f est un flot de G et que p est un chemin améliorant de G_f, alors si on définit $f_p : S \times S \to \mathbb{R}$ par

$$f_p(u,v) = \begin{cases} c_f(p) & \text{si } (u,v) \text{ appartient } p, \\ -c_f(p) & \text{si } (v,u) \text{ appartient } p, \\ 0 & \text{sinon} \end{cases}$$

On peut alors ajouter f_p à f pour obtenir un meilleur flot de G.

Coupe dans un réseau de transport Une coupe (E, T) d'un réseau de transport $G = (S, A)$ est une partition de S dans E et $T = S - E$ telle que $s \in E$ et $t \in T$. Le flot net à travers la coupe (E, T) est défini par $f(E, t)$ et la capacité de la coupe est $c(E, T)$.

On montre que le flot net à travers (E, T) est $f(E, T) = |f|$. De même, on montre les deux résultats suivants, pour toute coupe (E, T) et pour tout flot f :

$$c(E,T) \geq |f|$$

$$\min_{(E,T) \text{ coupe}} c(E,T) = \max_{f \text{ flot}} |f|$$

3 Algorithme de Ford-Fulkerson

L'algorithme complet est donné par l'Algorithme 2.

Cet algorithme laisse le choix de la méthode pour trouver le chemin améliorant. Un mauvais choix pourrait donner un algorithme qui n'est pas assuré de finir ! La méthode la plus commune consiste à trouver le chemin améliorant à l'aide d'une recherche en largeur, et on peut alors assurer que l'algorithme s'exécute en temps polynomial. C'est l'algorithme d'Edmonds-Karp, de complexité en temps $O(|S||A|^2)$. Si le réseau de transport est particulièrement creux, cet algorithme est très rapide.

4 Algorithme Réétiqueter-vers-l'avant

Préflot : Soit $G = (S, A)$ un réseau de transport avec une capacité c, de source s et de puit t. Un flot de G est une fonction $f : S \times S \to \mathbb{R}$, respectant les conditions suivantes :

capacité : Pour tout $u, v \in S$, on a $f(u, v) \leq c(u, v)$,

Algorithm 2 Algorithme de Ford-Fulkerson

pour chaque arc $(u, v) \in A[G]$ **faire**
  $f[u, v] \leftarrow 0$
  $f[v, u] \leftarrow 0$
fin pour
tantque il existe un chemin améliorant p de s à t dans le réseau résiduel G_f **faire**
  $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
  pour chaque arc $(u, v) \in p$ **faire**
    $f[u, v] \leftarrow f[u, v] + c_f(p)$
    $f[v, u] \leftarrow -f[u, v]$
  fin pour
fin tantque
retourner f

symétrie : Pour tout $u, v \in S$, on a $f(u, v) = -f(v, u)$,

« quasi-conservation » : Pour tout $u \in S - \{s\}$, on a $f(S, u) \geq 0$. On appelle *excédent* de flot sur u $e(u) = f(S, u)$.

$u \in S - \{s, t\}$ déborde ssi $e(u) > 0$.

On définira deux opérations à effectuer sur les sommets, pousser et réétiqueter. Durant tout l'algorithme, nous allons maintenir les variables suivantes :

- $f(u, v)$, le flot de u à v. La capacité résiduelle vaut $c_f(u, v) = c(u, v) - f(u, v)$.
- $h(u)$, la hauteur de u. On poussera seulement de u à v si $h(u) > h(v)$.
- $e(u)$, la somme du flot de et vers u.

À chaque étape, le flot f sera en fait un préflot. On remarque que le plus long chemin partant de s vers t comporte au plus $|S|$ sommets. On peut donc numéroter les sommets de telle façon que pour tout flot, $h(s) = |S|$ et $h(t) = 0$, et s'il existe un flot positif de u à v, alors $h(u) > h(v)$. Le flot va donc aller des plus grandes hauteurs vers les plus petites, tout comme si c'était de l'eau. Contrairement à l'algorithme de Ford-Fulkerson, pendant l'exécution de l'algorithme, f n'est pas un vrai flot. En quelques mots, les hauteurs des sommets (sauf s et t) sont ajustées, et le flot est envoyé entre les sommets, jusqu'à ce que tous les flots possibles ont atteint t. On continue alors à augmenter la hauteur des sommets internes jusqu'à ce que tous les flots entrés dans le réseau sans avoir atteint t soient retournés à s. Un sommet peut atteindre la hauteur $2|S| - 1$ avant cette étape, car le plus long chemin vers s (excluant t) est longue de $|S| - 1$ sommets et la hauteur de s est $h(s) = |S|$. t est toujours laissé à la hauteur 0.

pousser : Une poussée de u à v est faisable quand u déborde, on va déverser le trop-plein de u dans v. Il y a 3 conditions :

1. $e(u) > 0$
2. $c(u, v) - f(u, v) > 0$
3. $h(u) > h(v)$

Le trop-plein déversé sera égal à $\min(e(u), c(u, v) - f(u, v))$.

réétiqueter : On augmente la hauteur de u jusqu'à ce que ce qu'il soit plus haut qu'au moins un des sommets adjacent. Il y a 2 conditions :

1. $e(u) > 0$

2. $\forall v \in S, (u,v) \in A_f, h(u) \leq h(v)$

La nouvelle hauteur vaut donc $h(u) \leftarrow 1 + \min\{h(v) : (u,v) \in A_f\}$.

Enfin, il faut savoir décharger un sommet : tant que $e(u) > 0$, si tous les voisins de u n'ont pas été essayé depuis le dernier réétiquetage, on tente de pousser du flot vers un voisin. Sinon, on réétiquette u. Il faut donc mémoriser quels sont les sommets testés depuis le dernier réétiquetage.

Algorithme Réétiqueter-vers-l'avant, avec une heuristique FIFO

1. envoyer autant que possible du flot à partir de s
2. construire une liste des sommets, sauf s et t
3. tant que toute la liste n'a pas été traversée :
   (a) décharger le sommet actuel
   (b) mettre le sommet actuel en tête de liste
   (c) recommencer la traversée en partant du début de la liste

Le temps total de l'algorithme est alors en $O(|S|^3)$.

4.1 Un pseudo-code complet

discharge
ENTRÉES: u
tantque $e(u) > 0$ **faire**
  si $\text{seen(u)} < n$ **alors**
    si $c(u,v) - f(u,v) > 0$ et $h(u) > h(v)$ **alors**
      $\text{push}(u,v)$
    sinon
      $\text{seen}(u) \leftarrow \text{seen}(u) + 1$
    finsi
  sinon
    $\text{relabel}(u)$
    $\text{seen}(u) \leftarrow 0$
  finsi
fin tantque

push
ENTRÉES: u, v
$\text{send} \leftarrow \min(e(u), c(u,v) - f(u,v))$
$f(u,v) \leftarrow f(u,v) + \text{send}$
$f(v,u) \leftarrow f(v,u) - \text{send}$
$e(u) \leftarrow e(u) - \text{send}$
$e(v) \leftarrow e(v) + \text{send}$

relabel
ENTRÉES: u {trouve la plus petite hauteur rendant une poussée possible}

```
  min_height ← h(u)
  pour v ∈ {0, ..., n} faire
    si c(u, v) − f(u, v) > 0 alors
      min_height ← min(min_height, h(v))
      h(u) ← min_height + 1
    finsi
  fin pour
```

Fonction globale
ENTRÉES: C la matrice des capacités, s la source, t le puit

```
  n ← len(c) {on obtient le nombre de sommets}
  ∀(u, v) ∈ S × S, f(u, v) ← 0 {initialisation du flot}
  ∀u ∈ S, h(u) ← 0 {initialisation des hauteurs}
  ∀u ∈ S, e(u) ← 0 {initialisation des excédents}
  ∀u ∈ S, seen(u) ← 0 {initialisation des derniers sommets visités}
  ∀i ∉ {s, t}, list[i] ← i
  h(s) ← n
  e(s) ← ∞
  pour v ∈ {0, ..., n} faire
    push(s, v)
  fin pour
  p ← 0
  tantque p < len(list) faire
    u ← list(p)
    old_height ← h(u)
    discharge(u)
    si h(u) > old_height alors
      supprimer list(p) de list
      l'insérer en tête de list
      p ← 0
    finsi
    p ← p + 1
  fin tantque
  return ∑_u f(s, u)
```

5 Application

résolvez les problèmes suivants : `http://acm.uva.es/p/v109/10983.html`, `http://acm.uva.es/p/v104/10480.html`.