Mathématiques disctères 2 - graphes

Cours 3 - CCM

N. de Rugy-Altherre

Généralité

Défintions

- Un graphe $G = (V, E)$ orienté ou non est valué si on lui associe une fonction de coût $c : E \to \mathbb{R}$ (similairement, une étiquette sur les arêtes).
- Le coût d'un chemin $p =< v_0, v_1, \ldots, v_k >$ est

$$c(p) = \sum_{i=0}^{k} c((v_{i-1}, v_i))$$

- Notons $\delta(u, v)$ le coût minimal d'un chemin de u vers v (CCM).

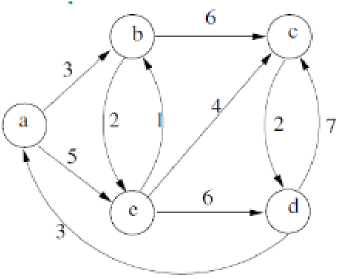
$$\delta(u, v) = \min\left(\{c(p) \mid p = \text{ chemin de u à v}\}\right)$$

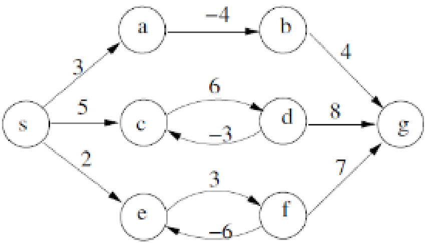
Exemples et conventions

Par convention :

- $\delta(u, v) = +\infty$ s'il n'existe pas de chemin de u vers v.
- $\delta(u, v) = -\infty$ s'il existe un circuit de coût négatif entre u et v.

Exemples





Principe d'optimalité des sous structures

Théorème

Tout sous-chemin d'un chemin chemin de coût minimal est un chemin de coût minimal.

Soit $G = (V, E, c)$ un graphe valué (orienté ou non) et $p =< v_0, v_1, \ldots, v_k >$ un chemin de coût minimal. Alors

$$\forall i, j \in [0, k],\ i \leq j \Rightarrow < v_i, \ldots, v_j > \ \text{CCM de } v_i \text{ à } v_j$$

Principe d'optimalité des sous structures

Théorème

Tout sous-chemin d'un chemin chemin de coût minimal est un chemin de coût minimal.
Soit $G = (V, E, c)$ un graphe valué (orienté ou non) et $p =< v_0, v_1, \ldots, v_k >$ un chemin de coût minimal. Alors

$$\forall i, j \in [0, k],\ i \leq j \Rightarrow < v_i, \ldots, v_j > \text{ CCM de } v_i \text{ à } v_j$$

Démonstration : Commençons par noter posons $p_{i,j}$ le sous chemin de p entre les sommets v_i et v_j.
Démontrons le théorème par l'absurde. Supposons qu'il existe un chemin $p =< v_0, v_1, \ldots, v_k >$ de coût minimal et $0 < \alpha < \beta < k$ tel que $p_{\alpha,\beta}$ est un sous chemin de p non minimal. p peut se décomposer en $p =< p_{0,\alpha}, p_{\alpha,\beta}, p_{\beta,k} >$ et $c(p) = c(p_{0,\alpha}) + c(p_{\alpha,\beta}) + c(p_{\beta,k})$
Par hypothèse, $p_{\alpha,\beta}$ n'est pas minimal, il existe donc un autre chemin $p'_{\alpha,\beta}$ tel que $c(p'_{\alpha,\beta}) < c(p'_{\alpha,\beta})$. Donc le chemin $p' =< p_{0,\alpha}, p'_{\alpha,\beta}, p_{\beta,k} >$ serait du coût plus petit que p, ce qui est impossible.

Algorithme de Dijkstra

Algorithme de Dijkstra

- Entrée : un graphe valué $G = (V, E, c)$ dont les valuations sont positives et deux sommets $s, t \in V$
- Postcondition : calcul d'une fonction (ou d'un tableau) $d : V \to \mathbb{R}$ telle que $d(v) = \delta(s, v)$.

Algorithme de Dijkstra

Algorithme de Dijkstra

- Entrée : un graphe valué $G = (V, E, c)$ dont les valuations sont positives et deux sommets $s, t \in V$
- Postcondition : calcul d'une fonction (ou d'un tableau) $d : V \to \mathbb{R}$ telle que $d(v) = \delta(s, v)$.

Généralisation du BFS à des graphes valués en calculant dynamiquement d. À tout moment de l'algorithme,

1. Si v est blanc (i.e. n'a pas encore été découvert), alors $d(v) = +\infty$
2. Si v est gris (i.e. a été découvert, mais d peut encore changer) : $\delta(s, v) \leq d(v) < +\infty$
3. Si v est noir, alors $d(v) = \delta(s, v)$

Algorithme de Dijkstra

Algorithme de Dijkstra

- Entrée : un graphe valué $G = (V, E, c)$ dont les valuations sont positives et deux sommets $s, t \in V$
- Postcondition : calcul d'une fonction (ou d'un tableau) $d : V \to \mathbb{R}$ telle que $d(v) = \delta(s, v)$.

Principe :

- Stratégie gloutonne : à chaque itération on choisi le sommet gris minimisant d et on le colorie en noir
- d est mis à jour pour ses successeurs. On dit qu'on *relache* ces arêtes.

Dijkstra ne fonctionne que si les coûts sont posifits ou nuls :

$$\forall e \in E,\ c(e) > 0$$

Relâcher

Définition

Soit $G = (V, E, c)$ un graphe pondéré et $(u, v) \in E$. On s'intéresse au calcul de $d : V \to \mathbb{N}$.

On dit qu'on *relâche* l'arête (u, v) en faisant

```
Si d(v) > d(u) + c(u,v) Alors
        d(v) = d(u) + c(u,v)
```

Dijkstra

```
Fonction Dijkstra(g, c, s0)
    pour chaque sommet si ∈ S faire
        d[si] ← +∞ ; π[si] ← null ; Colorier si en blanc
    d[s0] ← 0 ; Colorier s0 en gris
    tant que il existe un sommet gris faire
        Soit si le sommet gris tel que d[si] soit minimal
        pour tout sommet sj ∈ succ(si) faire
            si sj est blanc ou gris alors
                relacher((si, sj), π, d)
                si sj est blanc alors
                    Colorier sj en gris
        Colorier si en noir
    retourne π et d
```

Validité de Dijkstra

Théorème

Si on exécute l'algorithme de Dijkstra sur un graphe pondéré $G = (V, E, c)$, c étant positive, et une origine s, alors après exécution on a

$$\forall u \in V,\ d(u) = \delta(s, u)$$

Démonstration : on va utiliser l'invariant de boucle suivant :

$$\textit{Tout sommet noir } v \textit{ vérifie } d(v) = \delta(s, v)$$

- Initialisation : au début de la boucle, aucun sommet n'est noir. Donc l'invariant est vérifié.

Demonstration de la validité de Dijkstra

- Récurrence. Raisonnons par l'absurde. Soit u le premier sommet à être colorié en noir tel que $d(u) \neq \delta(s, u)$. Plaçons nous à l'itération I.5 juste avant que u devienne noir. Alors :

1. $u \neq s$
2. il existe un chemin de s à u (car sinon $d(u) = \delta(s, u) = +\infty$)
3. Soit $p =< s, v_0, \ldots, v_k, u >$ un chemin de coût minimal de s à u. p relie un sommet noir (s) et un sommet non noir (u).
4. Soit y le premier sommet de p non noir et x son prédécesseur (noir). p peut se décomposer en deux chemins :

   $$s \ldots^{p_1} \ldots x \rightarrow y \ldots^{p_2} \ldots u$$

   p_1 peut être vide et est composé que de sommets noirs, p_2 aussi et peut aussi contenir des sommets noirs.
5. $d(y) = \delta(s, y)$ car comme son prédécesseur est noir et que u est le premier noir ne vérifiant pas cette affirmation, y la vérifie

Demonstration de la validité de Dijkstra

1. Comme les poids sont positifs $\delta(s, y) \leq \delta(s, u)$ et donc

$$d(y) = \delta(s, y) \leq \delta(s, u) \leq d(u)$$

   D'après l'inégalité triangulaire.

2. Comme les sommets u et y sont gris, on a $d(u) \leq d(y)$ (car on a choisi le plus petit sommet gris). Donc les inégralités ci-dessus sont des égalités :

$$d(y) = \delta(s, y) = \delta(s, u) = d(u)$$

   Donc $d(u) = \delta(s, u)$ ce qui est contraire à notre hypthèse. cqfd.

Demonstration de la validité de Dijkstra

Inégalité triangulaire

$$\forall (u, v) \in E,\ \delta(s, v) \leq \delta(s, u) + c(u, v)$$

Complexité

Complexité

- $\mathcal{O}(n^2 + p)$ si la recherche du sommet gris minimal est linéaire.
- $\mathcal{O}((n+p)\log(n))$ si les sommets gris sont stockés dans un tas binaire.

Principe

- Programmation dynamique : décomposer en sous-problèmes
- On supposera les sommets du graphe numéroté de 1 à n. L'ordre est arbitraire.
- Le sous problème est la recherche d'un chemin de coût minimal dans le sous graphe $G_{i,j}$, composé des sommets numérotés de i à j.

Formules de Floyd-Warshall

Notations

Soit $G = ([1, n], E)$ un graphe dont les sommets sont numérotés de 1 à n. Pour $i, j, k \in [1, n]$, notons $W_{i,j}^k$ le poids minimal d'un chemin entre les sommets i et j n'utilisant que des sommets intérmédiaires numérotés de 1 à k.

Remarque : $(W_{i,j}^0)$ est la matrice d'adjacence

Théorème

$$\forall i, j, k \in [1, n],\ W_{i,j}^k = \min(W_{i,j}^{k-1}, W_{i,k}^{k-1} + W_{k,j}^{k-1})$$

Formules de Floyd-Warshall

Théorème

$$\forall i,j,k \in [1,n],\ W_{i,j}^{k} = \min(W_{i,j}^{k-1}, W_{i,k}^{k-1} + W_{k,j}^{k-1})$$

Démonstration : Soit $(i,j,k) \in [1,n]$ et p le chemin de coût minimal entre i et j dont les sommets intermédiaires sont dans $[1,k]$. C'est à dire $W_{i,j}^{k} = c(p)$ Alors :

- Soit k n'est pas dans p et donc $W_{i,j}^{k} = W_{i,j}^{k-1}$
- Soit k est dans p, exactement une fois et alors p est la concaténation du CCM de i à k et de celui de k à j.

Condition pour que ça fonctionne : k soit dans p au plus une fois : les circuits doivent avoir un poids positif ou nul.

Formules de Floyd-Warshall

Théorème

$$\forall i, j, k \in [1, n],\ W_{i,j}^{k} = \min(W_{i,j}^{k-1}, W_{i,k}^{k-1} + W_{k,j}^{k-1})$$

S'il n'existe pas de circuit de poids négatifs

Démonstration : Soit $(i, j, k) \in [1, n]$ et p le chemin de coût minimal entre i et j dont les sommets intermédiaires sont dans $[1, k]$. C'est à dire $W_{i,j}^{k} = c(p)$ Alors :

- Soit k n'est pas dans p et donc $W_{i,j}^{k} = W_{i,j}^{k-1}$
- Soit k est dans p, exactement une fois et alors p est la concaténation du CCM de i à k et de celui de k à j.

Condition pour que ça fonctionne : k soit dans p au plus une fois : les circuits doivent avoir un poids positif ou nul.

Algorithme

```
Fonction FloydWarshall(g,c)
  Pour k allant de 0 a n Faire
    Pour i allant de 0 a n Faire
      Pour j allant de 0 a n Faire
        W[i][j][k] = min(W[i][j][k-1], W[i][k][k-1] + W[k][j][k-1])
      Fin Pour
    Fin Pour
  Fin Pour
        renvoyer W
Fin Fonction
```

Algorithme, version optimisée

```
Fonction FloydWarshall(g,c)
  Pour k allant de 0 a n Faire
    Pour i allant de 0 a n Faire
      Pour j allant de 0 a n Faire
        W[i][j] = min(W[i][j], W[i][k] + W[k][j])
      Fin Pour
    Fin Pour
  Fin Pour
        renvoyer W
Fin Fonction
```

Complexité

La complexité de cet algorithme est $\mathcal{O}(n^3)$.

Cet algorithme donne la taille des chemins de coût minimaux entre tous les couples de sommets.

Principe

- Décomposer le problème en sous-problèmes (programmation dynamique)
- Définition de la solution optimale par des équations récusives
- Calculer en partant du cas de base
- Ne pas recalculer plusieurs fois la même chose "mémoïsation".

Proposé par Bellamn (1952) pour résoudre des problèmes de planification.

Équations récursives pour Bellam-Ford

Définition

Soit $G = (V, E)$ un graphe, orienté ou non. Soit s, v deux sommets. On s'intéresse aux calculs des chemins de coûts minimaux de s à v, $\delta(s, v)$.
Notons $\delta^k(s, v)$ la poids du chemin de coût minimal de s à v passant par au plus k arcs.

Formules récusives :

- Cas de base : $\delta^0(s, s) = 0$ et $\delta^0(s, v) = +\infty$ si $v \neq s$.
- Récurence : si $k > 1$, $\delta^k(s, v) = \min(\{\delta^{k-1}(v)\} \cup \{\delta^{k-1}(s, u) + c(u, v) \mid u \in \text{pred}(v)\})$

Quel est le cas d'arret ? C'est à dire pour quel k a-t-on $\delta^k(s, v) = \delta(s, v)$?

Arrête de Bellman-Ford

Validité de Bellman-Ford

- Soit les valeurs des tableaux se stabilisent avant $|V| - 1$ itération. Auquel cas, $d(v) = \delta(s, v)$.
- Soit ces valeurs ne se stabilisent pas ou s'il existe un arc tel que $v.d > u.d + c(u, v)$. Dans ce cas il existe un circuit de poids strictement négatif.

Algorithmes CCM

- Algorithme glouton : (ex. Dijkstra). Principe : soit $u, v, w \in V$ tel que le chemin de coût minimal de u à w soit $< u, v_0, v_1, \ldots, v_k, v, w >$. Alors

$$\delta(u, w) = \delta(u, v) + c(v, w)$$

- Programmation dynamique : Bellman-Ford. Principe :

$$\delta(u, w) = \min_{v \in pred(w)} \delta(u, v) + c(v, w)$$

- Programmation dynamique : Floyd-Warshall. Principe :

$$\delta(u, w) = \min_{v \in E} \delta(u, v) + c(v, w)$$

Algorithmes CCM

Relâcher un arc consiste à recalculer la distance du CCM :

- Dijkstra : relâche les arcs partant du sommet en minimisant d :
  1. Chaque arc est relâché exactement une fois
  2. Ne marche que si les coûts sont positifs
- Bellman-Ford relâche tous les arcs à chaque itération jusqu'à convergence.
  1. Chaque arc est relâché plusieurs fois
  2. Marche dans tous les cas
- Floyd-Warshall. : relâche les arcs si tous ses prédécesseurs ont été relâchés
  1. Chaque arc est relâché exactement une fois
  2. Ne marche que si le graphe est acyclique

$$\delta(u, w) = \min_{v \in E} \delta(u, v) + c(v, w)$$