Chapitre 4 : Problème du plus court chemin

Y. DIENG, Département Informatique
UFR ST, Université Assane Seck de Ziguinchor

4. september 2021

1 Plus courts chemins à origine unique

Un automobiliste souhaite trouver le plus court chemin possible entre Dakar et Dioubel. Étant donnée une carte routière du Sénégal, avec les distances de chaque portion de route, comment peut-il déterminer la route la plus courte ?

Une possibilité consiste à énumérer toutes les routes de Dakar à Diourbel, additionner les distances de chacune puis choisir la plus courte. Toutefois, on s'apercoit rapidement que, même en n'autorisant pas les routes qui contiennent des cycles, il existe des millions de possibilités, dont la plupart ne valent même pas la peine d'être considérées. Par exemple, une route allant de Dakar à Diourbel en passant par Kaolack est manifestement une mauvaise option car, Kaolack fait faire un détour d'une centaine de kilomètres.

Dans ce chapitre, nous allons montrer comment résoudre efficacement ce type de problèmes. Dans un problème de plus courts chemins, on possède en entrée un graphe orienté pondéré $G = (V, E)$, avec une fonction de pondération $w : E \to \Re$ qui fait correspondre à chaque arc un poids à valeur réelle. La longueur du chemin $p = \{v_0, v_1, ..., v_k\}$ est la somme des poids (longueurs) des arcs qui le constituent :

$W(p) = \sum_{i=1}^{k} w(v_{i+1}, v_i)$

On définit la longueur du plus court chemin entre u et v par

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

Un plus court chemin d'un sommet u à un sommet v est alors défini comme un chemin p de longueur $w(p) = d(u, v)$.

Dans l'exemple Dakar-Diourbel, on peut modéliser la carte routière par un graphe : les sommets représentent les intersections de route, les arcs les portions de route, et les poids les distances. Le but est de trouver un plus court chemin entre un intersection donnéeàDakar (par exemple, le rond-point Yoff, route de l'aéroport) et une intersection donnéeàDiourbel (par exemple, le carrefour route de Dakar/route de Gossas).

Les poids des arcs peuvent mesurer autre chose que des distances. On s'en sert souvent pour représenter un temps, un coût, des pénalités, une déperdition, ou n'importe quelle autre quantité qui s'accumule linéairement le long d'un chemin, et qu'il s'agit de minimiser.

L'algorithme de parcours en largeur vu précédemment est un algorithme de recherche des plus courts chemins, qui fonctionne sur des graphes non pondérés, c'est- à-dire des graphes pour lesquels on considère que chaque arc possède un poids unitaire. Bon nombre des concepts du parcours en largeur se retrouvent dans l'étude des plus courts chemins dans les graphes pondérés. Le lecteur de cette partie du cours est donc encouragé avant de continuer.

Variantes

Dans ce chapitre, nous restreindrons notre étude au problème de recherche du plus court chemin à origine unique : étant donné un graphe $G = (V, E)$, on souhaite trou- ver un plus court chemin depuis un sommet origine donné $sinV$ vers n'importe quel sommet $vinV$. Beaucoup d'autres problèmes peuvent être résolus par l'algorithme à origine unique, notamment les variantes suivantes :

Plus court chemin à destination unique : Trouver un plus court chemin vers un sommet de destination t à partir de n'importe quel sommet v. En inversant le sens de chaque arc du graphe, on peut ramener ce problème à un problème à origine unique.

Plus court chemin pour un couple de sommets donné : Trouver un plus court chemin de u à v pour deux sommets donnés u et v. Si on résout le problème à origine unique pour le sommet origine u, on résout ce problème également. Par ailleurs, on ne connaît aucun algorithme qui soit meilleur asymptotiquement 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 à partir de chaque sommet ; mais on peut généralement le résoudre plus rapidement, et sa structure est intéressante en elle-même.

Sous-structure optimale

Les algorithmes de plus court chemin s'appuient généralement sur la propriété selon laquelle un plus court chemin entre deux sommets contient d'autres plus courts chemins.

Lemme 1. *Etant donné un graphe orienté pondéré $G = (V, E)$ ayant la fonction de pondération $w : V \to R$, soit $p = \{v_1, v_2, dot, v_k\}$ un plus court chemin du sommet v_1 au sommet v_k et, pour tout i et tout j tels que $1 \leq i \leq j \leq k$, soit $p_{ij} = setv_i, v_i + 1, dot, v_j$ le sous-chemin de p entre le sommet v_i et le sommet v_j. Alors, p_{ij} est un plus court chemin de v_i à v_j.*

Bevis. A faire ...

Arcs de poids négatif

Pour certaines instances du problème du plus court chemin à origine unique, on peut rencontrer des arcs dont les poids sont négatifs. Si le graphe $G = (V, E)$ ne contient aucun circuit de longueur strictement négative accessible à partir de l'origine s, alors, pour tout $v \in V$, la longueur du plus court chemin $d(s, v)$ reste bien défini, même si sa valeur est négative. Toutefois, s'il existe un circuit de longueur strictement négative accessible depuis s, la longueur d'un plus court

chemin n'est plus bien défini. Aucun chemin entre s et un sommet du circuit ne peut être un plus court chemin : on peut toujours trouver un chemin de moindre longueur qui suit le « plus court » chemin puis traverse le circuit de longueur strictement négative. S'il existe un circuit de longueur strictement négative sur un chemin de s à v, on définit $d(s, v) = -\infty$.

Certains algorithmes de recherche du plus court chemin, tel l'algorithme de Dijkstra, supposent que tous les poids d'arc du graphe d'entrée sont positifs ou nuls, comme dans l'exemple de la carte routière.

D'autres, comme celui de Bellman-Ford autorisent des arcs de poids négatifs dans le graphe d'entrée, et fournissent une réponse correcte, du moment qu'aucun circuit de longueur strictement négative n'est accessible à partir de l'origine. En général, si un tel circuit existe, l'algorithme peut détecter et signaler son existence. Un plus court chemin ne peut pas contenir de circuit.

Représentation des plus courts chemins

On souhaite souvent calculer non seulement les poids des plus courts chemins, mais aussi les sommets présents sur ces plus courts chemins. La représentation utilisée pour les plus courts chemins ressemble à celle utilisée pour les arborescences de parcours en largeur du chapitre 3.

Etant donné un graphe $G = (V, E)$, on gère pour chaque sommet $v \in V$ un prédécesseur $parents[v]$ qui est soit un autre sommet, soit NIL. Les algorithmes de recherche du plus court chemin exposés dans ce chapitre gèrent les attributs p de manière que la chaîne des prédécesseurs partant d'un sommet v suive, en remontant, un plus court chemin entre s et v. Ainsi, étant donné un sommet v pour lequel $parents[v] \neq NIL$, la procédure IMPRIMER-CHEMIN(G, s, v) du chapitre 3 peut servir à imprimer un plus court chemin de s vers v.

Cependant, lors de l'exécution d'un algorithme de recherche du plus court chemin, les valeurs dans $parents$ n'ont pas besoin d'indiquer les plus courts chemins. Comme dans le parcours en largeur, nous nous intéresserons au sous-graphe prédécesseur $Gp = (V_p, E_p)$ induit par les valeurs $parents$. Ici aussi, on définira l'ensemble Sp comme étant l'ensemble des sommets de G ayant des prédécesseurs différents de NIL, complété par l'origine s :

$V_p = \{v \in V : parents[v] \neq NIL\} \cup \{s\}$

L'ensemble E_p est l'ensemble des arcs induits par les valeurs de p pour les sommets appartenant à V_p :

$E_p = \{(parents[v], v) \in E : v \in V_p - \{s\}\}$

Nous allons prouver que les valeurs de $parents$ obtenues par les algorithmes de ce chapitre ont pour propriété que, lorsqu'ils se terminent, G_p est une arborescence de plus courts chemins. Une arborescence de plus courts chemins ressemble à une arborescence de parcours en largeur vue au chapitre 3. la seule diffrérence est que les plus cours chamins à partir de son origine ne sont plus exprimés en fonction du nombre d'arcs mais du poids des arcs. Plus précisément, si $G = (V, E)$ un graphe podéré, sans cisrcuit de longueur strictement négative, accécible depuis le sommet origine s. Alors, une arborescence de plus courts chemins de racine s est un sous graphe orienté $G' = (V', E')$ avec $V' \subseteq V$ et $E' \subseteq E$ tel que :

- V' est l'ensemble des sommets accessibles depuis s dans G.
- G' forme une arborecence de racine s

- pour un sommet $v \in V$, le chemin unique de s à v dans G' est un plus court chemin de s à v dans G.

Les plus courts chemins ne sont pas forcément uniques, pas plus que les arborescences de plus courts chemins.

Relachement

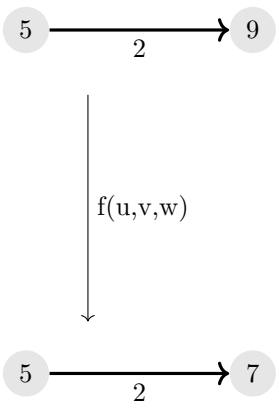
Les algorithmes de ce chapitre emploient la technique du relâchement. Pour chaque sommet $v \in V$, on gère un attribut $d[v]$ qui est un majorant de la longueur d'un plus court chemin entre l'origine s et v. On appelle $d[v]$ une estimation de plus court chemin. On initialise les estimations et les prédécesseurs via la procédure en temps $\Theta(V)$ que voici. On suppose que les sommet de G sont numérotés de 1 à $n =\mid V \mid$. On suppose aussi que $INFINI = +\infty$.

```
SOURCE-UNIQUE-INITIALISATION(G, s)
1       for (v = 1; v <= n; v++)
2                   d[v] = INFINI
3                   parents[v]  = NIL
4       d[s] =  0
```

Après l'initialisation, on a $d[v] = +\infty$ pour tout sommet v de $V \setminus s$ et $d[s] = 0$. de même, pour tout sommet v de V, on a $parents[v] = NIL$.

Le processus de relâchement d'un arc (u, v) consiste à tester si l'on peut améliorer le plus court chemin vers v trouvé jusqu'ici en passant par u et, si tel est le cas, en actualisant $d[v]$ et $parent[v]$. Une étape de relâchement peut diminuer la valeur de l'estimation de plus court chemin $d[v]$ et mettre à jour le champ prédécesseur $parent[v]$ de v. Le code suivant effectue une étape de relâchement sur l'arc (u, v).

```
RELACHER(u, v, w){
1       if (d[v] > d[u] + w(u,v)){
2                 d[v] = d[u] + w(u, v)
3                 parent[v] = u
4     }
}
```



Vous avez ci-dessus un exemple de relachement f de l'arc (u, v) de poids $w(u, v) = 2$. L'estimation de plus court chemin de chaque sommet est affiché dans le sommet. Comme $d[v] = 9 > d[u]+w(u, v) = 5+2 = 7$ avant relachement, alors le relachement fait décroitre la valeur de $d[v]$.

Chaque algorithme de ce chapitre relâche les arcs du graphe donnée en entrée, de manière réitérée grace à un appelle de la fonction SOURCE-UNIQUE-INITIALISATION. Le relâchement est la seule facon de modifier les estimations de plus court chemin et les prédécesseurs. La diférence entre les algorithmes de ce chapitre porte sur le nombre de fois qu'ils relâchent chaque arc et par l'ordre dans lequel ils relâchent les arcs. Dans l'algorithme de Dijkstra et dans l'algorithme de plus court chemin pour graphe sans circuit orienté, chaque arc est relâché une fois et une seule. Dans l'algorithme de Bellman-Ford, chaque arc est relâché plusieurs fois.

Propriété de plus courts chemins et relachement

Pour s'assurer de la conformité des algorithmes de ce chapitre, nous allons eploiter plusieurs propriétés des plus courts chemins et au relachement.

`Inégalité triangulaire :` Pour tout arc $(u, v) \in E$, on a $\delta(s, v) <= \delta(s, u) + w(u, v)$. Propriété du majorant : On a toujours $d[v] >= \delta(s, v)$ pour tous les sommets $v \in V$, et une fois que $d[v]$ a atteint la valeur $\delta(s, v)$, elle ne change plus.

`Propriété aucun-chemin :` S'il n'y a pas de chemin de sà v, alors on a toujours $d[v] = \delta(s, v) = \infty$.

`Propriété de convergence :` Si $s \rightsquigarrow u \rightarrow v$ est un plus court chemin dans G pour un certain $u, v \in V$ et si $d[u] = \delta(s, u)$à un certain instant antérieur au relâchement de l'arc (u, v), alors $d[v] = \delta(s, v)$ en permanence après le relâchement.

`Propriété de relâchement de chemin :` Si $p = \{v_0, v_1, \ldots, v_k\}$ est un plus court chemin de $s = v_0$ à v_k et si les arcs de p sont relâchés dans l'ordre $\{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\}$, alors $d[v_k] = \delta(s, v_k)$. Cette propriété est vraie indépendamment de toutes autres étapes de relâchement susceptibles de se produire, même si elles s'entremêlent avec des relâchements d'arcs de p.

`Propriété de sous-graphe prédécesseur :` Une fois que $d[v] = \delta(s, v)$ pour tout $v \in V$, le sous-graphe prédécesseur est une arborescence de plus courts chemins de racine s.

1.1 Algorithme de Bellman-Ford

L'algorithme de `Bellman-Ford` résout le problème des plus courts chemins à origine unique dans le cas général où les poids d'arc peuvent avoir des valeurs négatives. Étant donné un graphe orienté pondéré $G = (V, E)$, d'une fonction de pondération $w : EverR$, et d'une origine s, l'algorithme de Bellman-Ford retourne une valeur booléenne indiquant s'il existe ou non un `circuit de longueur strictement négative` accessible à partir de s. Si un tel circuit n'existe pas, l'algorithme donne les `plus courts chemins` ainsi que leurs poids.

L'algorithme utilise la technique du relâchement, diminuant progressivement une estimation $d[v]$ du poids d'un plus court chemin depuis l'origine s vers chaque sommet $v \in V$ jusqu'à atteindre la valeur réelle du poids de plus

court chemin, $\delta(s, v)$. L'algorithme retourne `VRAI` si et seulement si le graphe ne contient aucun circuit de longueur strictement négative accessible depuis l'origine.

```
BELLMAN-FORD(G, w, s)
1 SOURCE-UNIQUE-INITIALISATION(G, s)
2 pour i = 1 a |V| -1  faire
3              Pour chaque arc (u, v) de  E faire
4                       RELACHER (u,  v,  w)
5 Pour chaque arc (u, v) de E faire
6              si d[v] > d[u] + w(u, v)) alors
7                                retourner FAUX
8 retourner VRAI
```

L'algorithme de Bellman-Ford s'exécute en temps $O(VE)$: l'initialisation de la ligne 1 prend $\Theta(V)$; chacun des $|V| - 1$ passages des lignes 2–4 prend $\Theta(E)$; la boucle pour des lignes 5–7 prend $O(E)$.

Pour démontrer la validité de l'algorithme de Bellman-Ford, on commence par montrer que, s'il n'existe aucun circuit de longueur strictement négative, l'algorithme calcule les longueurs de plus court chemin corrects pour tous les sommets accessibles depuis l'origine.

Lemme 2. *Soit $G = (V, E)$ un graphe orienté pondéré, de fonction de pondération $w : E \to R$ et d'origine s ; on suppose que G ne contient aucun circuit de longueur strictement négative accessible depuis s. Alors, après les $|S| - 1$ itérations de la boucle pour des lignes 2-4 de BELLMAN-FORD, on a $d[v] = \delta(s, v)$ pour tout sommet v accessible depuis s.*

Bevis. Soit v un sommet de G accessible depuis s. Soit $p = \{v_1, v_2, \ldots, v_k\}$ un plus court chemin entre $s = v_1$ a $v = v_k$. Le chemin p a au plus $|V| - 1$ arcs, et donc $k \leq |V| - 1$. Chacune des $|V| - 1$ itération de la boucle `Pour` des ligne $2 - 4$ relache tous les $|E|$ arcs. Pami les arcs relachés à la i^{eme} itération, pour $i = 1, 2, \ldots, k$, il y'a l'arc (v_{i-1}, v_i). D'après la propriété de relachement de chemin, on a $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

Théorème 1. *[Validité de l'algorithme de Bellman-Ford] Exécutons la procédure BELLMAN-FORD sur un graphe orienté pondéré $G = (V, E)$, valué par une fonction de pondération w et d'origine s.*

- *Si G ne contient aucun circuit de longueur strictement négative accessible depuis s, alors l'algorithme retourne `VRAI`, on a $d[v] = \delta(s, v)$ pour tous les sommets $v \in V$, et le sous-graphe prédécesseur G_p est une arborescence de plus courts chemins de racine s.*
- *Si G contient un circuit de longueur strictement négative accessible à partir de s, alors l'algorithme retourne retourne `FAUX`.*

Bevis. A faire, ...

1.2 Algorithme de Dijkstra

L'algorithme de `Dijkstra` résout le problème de la recherche d'un plus court chemin à origine unique pour un graphe orienté pondéré $G = (V, E)$ dans le cas où tous les arcs ont des poids positifs ou nuls. Dans cette section, on supposera donc que $w(u, v) \geq 0$ pour chaque arc $(u, v) \in E$. L'algorithme de Dijkstra a un temps d'exécution inférieur à celui de Bellman-Ford.

L'algorithme de Dijkstra gère un ensemble E de sommets dont les longueurs finales de plus court chemin à partir de l'origine s ont déjà été calculées. A chaque itération, l'algorithme choisit le sommet u de $V - E$ dont l'estimation de plus court chemin est minimale, l'ajoute à E, puis relâche tous les arcs partant de u. Dans l'implémentation ci-après, on gère une file de priorités min F de sommets, en prenant pour clé la valeur de leur attribut d.

```
DIJKSTRA(G, w, s)
 1 SOURCE-UNIQUE-INITIALISATION(G, s)
 2 E ← ø
 3 F ← V
 4 tant que F <> ø
 5      faire u ← EXTRAIRE-MIN(F)
 6             E ← E + {u}
 7              pour chaque sommet v de Adj[u]
 8                      faire RELACHER(u, v, w)
```

Théorème 2. *[Validité de l'algorithme de Dijkstra] Si l'on exécute l'algorithme de Dijkstra sur un graphe orienté pondéré $G = (V, E)$, avec une fonction de pondération positive w et une origine s, après exécution l'on a $d[u] = \delta(s, u)$ pour tous les sommets $u \in V$.*

Bevis. A fair,...

Théorème 3. *[**Korma**] Le temps d'éxécution est en $O(V^2)$ si $E = o(V^2/\log V)$. On peut l'implémenter en temps $O(VlogV + E)$*