shortest path problem

Le problème de la recherche du plus court chemin est très répandu et l'une des application les plus utilisées et la recherche d'un plus court chemin dans une carte routière avec distance ou temps de parcours.
Si on peut énumérer toutes les possibilités (tous les chemins entre le point de départ et le point d'arrivée), alors un algorithme naïf serait :
- énumérer tous les chemins
- calculer la distance parcouru pour chacun de ces chemins
- et ne retenir que les chemins optimaux.

Deux problèmes émergent :

- Existence de cycles : puisqu'on peut passer par un cycle un nombre quelconque de fois, il existe une infinité de chemins
- D'autre part, même si on ne veut pas considérer les chemins comportant des cycles, il y a des millions et des millions de chemins entre deux points. Ce qui est problématique d'un point de vue calculatoire.

Inputs to the problem

- un graphe pondéré $G = (S, A)$, c'est-à-dire un graphe muni d'une fonction de pondération : $\omega : A \longrightarrow \mathbb{R}$
- le poids d'un chemin est la somme des poids des arêtes (arcs) qui le constituent. On note alors $\omega(p)$ le poids du chemin p.

Definition

- le **poids du plus court chemin** entre u et v est défini par :

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

- **un plus court chemin** d'un sommet u à un sommet v est un chemin tq $\omega(p) = \delta(u,v)$

Several variants

Dans ce cours : problème de la recherche du plus court chemin à origine unique :

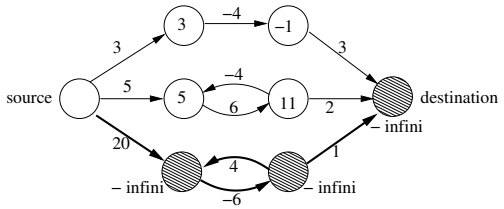
- on se donne un sommet source qu'on va noter s, et
- on cherche un plus court chemin entre s et n'importe quel autre sommet.

Des variantes :

- recherche d'un plus court chemin à destination unique : renverser les flèches
- recherche d'un plus court chemin pour un couple de sommets,
- recherche d'un plus court chemin pour tout couple de sommets.

Problème dû aux arcs de poids négatifs :

- Si cycle de poids négatif, le poids du plus court chemin peut ne pas exister...



Edge Relaxation technique

Intuition :

- diminuer progressivement une estimation par excès du poids du plus court chemin pour chaque sommet,
- boucler jusqu'à ce que cette estimation par excès soit égale au poids du plus court chemin.

Lemma

Soit G un graphe pondéré $G = (S, A)$ et $\omega : A \longrightarrow \mathbb{R}$. Soit $p =< v_1, ...v_k >$ un plus court chemin de v_1 à v_k. Alors un chemin $p_{ij} =< v_i...v_j >$, $1 \leq i < j \leq k$ est un plus court chemin de v_i à v_k.

Preuve : par l'absurde □

Initialisation

$d[v]$: une structure de données représentant l'estimation par excès du poids d'un plus court chemin de s à v (s : sommet source fixé).
$d[v]$: *estimation* de la pondération d'un plus court chemin.

Les méthodes de relâchement se décomposent toutes en

- une phase d'initialisation des estimations par excès $d[v]$

$$\textbf{Initialisation :} \begin{cases} d[v] = \infty; & \forall v \in S[G] \setminus \{s\} \\ d[s] = 0; & \\ \pi[v] = NIL & \forall v \in S[G] \end{cases}$$

- une itération de *relâchements*.

Relâcher un arc : c'est tester l'arc (u, v) pour savoir s'il est possible en passant par cet arc d'améliorer le plus court chemin jusqu'à v.
Si oui, mettre à jours $d[v]$ et $\pi[v]$

$$\texttt{relacher}(u, v, \omega) = \texttt{si } \ d[v] > d[u] + \omega(u, v) \ \texttt{ alors} \begin{cases} d[v] & = & d[v] + \omega(u, v) \\ \pi[v] & = & u \end{cases}$$

Dijkstra's Algorithm

Cet algorithme n'est valable que si tous les arcs ont des poids positifs ou nul.

- L'algorithme maintient à jour l'ensemble E des sommets dont le poids du plus court chemin a été calculé.

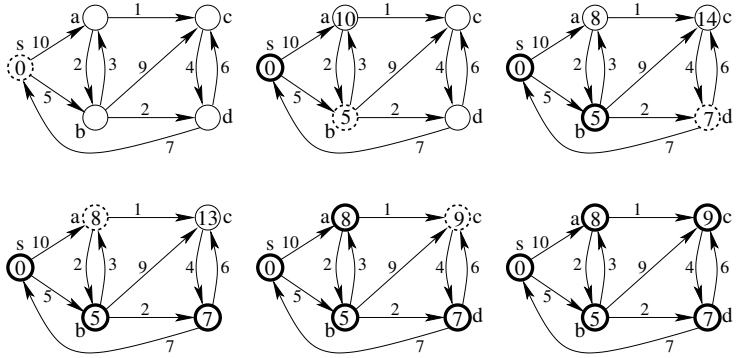
$$E = \{v \ \mid \ d[v] = \delta(s, v)\}$$

- L'algorithme choisit le sommet $u \in S \setminus E$ tel que $d[u]$ soit minimum
- u est alors inséré dans E,
- L'algorithme relâche alors tous les arcs partant de u.

```
1  Dijkstra(G,omega,s)
2     initialisation de d[]   /* initialisées à + infty sauf d[s] =0 */
3     initialisation de PI[]  /* initialisées à NIL                  */
4     E = {}
5     F = S(G)
6
7     Tant que F != {} faire
8         u = extraire argmin { d[u] | u in F}
9         E = Union(E,{u})
10        pour chaque sommet adjacent à u faire
11           relâcher(u,v,omega)
12        fin pour
13    fin tantque
14 Fin
```


Dijkstra's Algorithm : Example

1. initialisation : seul le sommet s a une valeur finie
2. le sommet qui a la valeur $d[.]$ minimale est le sommet s (pointillés), il est mis dans l'ensemble E (gras) et on relâche les deux arcs sortants, ce qui permet de mettre à jour $d[a]$ et $d[b]$.
3. ...



Dijkstra's Algorithm : Example

Il reste à prouver que lorsqu'on insère un sommet dans E, on a en effet calculer sa valeur de Plus Court Chemin : $d[v] = \delta(s, v)$.

- On va montrer que $d[u] = \delta(s, u)$ au moment où u est inséré dans E. Par l'absurde : supposons que u soit le premier sommet pour lequel $d[u] \neq \delta(s, u)$.
  1. On a $u \neq s$ (car s a été inséré avant).
  2. $E \neq \{\}$ car s en fait partie.
  3. Il existe un chemin de s à u car sinon $d[u] = \infty$,
     donc il existe un plus court chemin noté $p = \underbrace{s...x}_{\in E} \rightarrow \underbrace{y...u}_{\notin E}$ où y est le premier élément n'appartenant pas à E.
     Puisque u est choisi comme 1^{er} sommet tel que $d[u] \neq \delta(s, u)$, on avait $d[x] = \delta(s, x)$
     L'arc (x, y) a été relâché (quand x a été inséré dans E)
     p étant un plus court chemin, $d[y] = \delta(s, y)$ (car sinon p ne serait pas un plus court chemin).
  4. $d[y] = \delta(s, y)$
     $d[y] = \delta(s, y) \leq \delta(s, u)$ car un plus court chemin de s à u passe par y
     $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$ car pour tout t on a $\delta(s, t) \leq d[t]$
     Or u a été choisi pour relâchement de ses arcs sortants, $d[u] \leq d[y]$
     donc $d[y] = \delta(s, y) = \delta(s, u) = d[u]$
     et par conséquent, $d[u] = \delta(s, u)$.
     Contradiction.

Complexity

Considérons une représentation du graphe par liste d'adjacence.

- La recherche du minimum dans la liste F : $O(|F|)$
- Le temps total de recherche des différents mininum est en $O(|S|^2)$, car au départ $|F|$ correspond au nombre de sommets, et à chaque fois qu'on rentre dans le `while`, on enlève un élément de F.
- On relâche tous les arcs sortant d'un sommet qu'une et une seule fois, lorsque ce sommet est choisi. Donc, le temps nécessaire à l'ensemble des relâchements est en : $O(|A|)$ parce que chaque liste d'adjacence est parcourue une seule fois, et que la somme des longueurs des listes d'ajacence vaut $|A|$.
- La complexité temporelle est donc $O(|S|^2 + |A|) = O(|S|^2)$

Bellman–Ford Algorithm

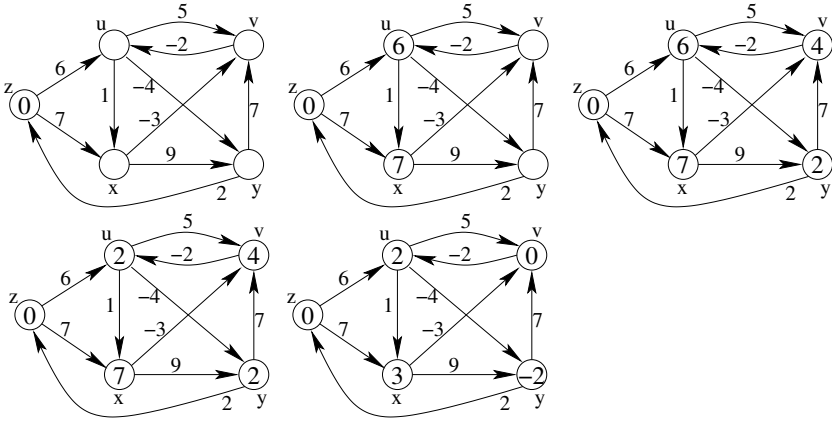
- les arcs de poids négatifs sont acceptés.
- l'algorithme de Bellman–Ford est donc plus général.
- Si un cycle de poids négatif existe (est accessible à partir de s), l'algorithme le detecte.
  Sinon, l' algorithme donne les plus courts chemins.

```
1  Bellman(G,omega,s)
2     initialisation de d[]   /* initialisées à + infty sauf d[s] =0 */
3     initialisation de PI[]  /* initialisées à NIL                  */
4
5     Pour i=1 à |S|-1 faire
6        Pour chaque arc (u,v) de A
7           relacher(u,v,omega)
8        Fin Pour
9     Fin Pour
10
11    Pour chaque arc (u,v) de A faire
12      si d[v] > d[u] + omega(u,v) alors
13        Renvoyer("présenced'un cycle de poids négatif")
14
15    Retourner Vrai.
16 Fin
```


Example

on suppose que les arcs sont énumérés par ordre lexicographique : (u, v), (u, x), (u, y), (v, u), (x, v), (x, y), (y, v), (y, z), (z, u) et enfin (z, x).

1. initialisation : seul la source z a une distance strictement inférieur à $+\infty$.
2. après le 1^{er} tour : seuls les sommets u et x ont des distances mises à jour.



3. Lorsqu'on fait un tour de relâchements en plus, on devrait mettre à jour : $d[u]$ car $d[v] + \omega(v, u) = 0 - 2 = -2 < d[u] = 2$. Il y a donc un cycle de poids négatif... Dans cet exemple, il s'agit du cycle (u, x, v).

Complexity of the Bellman–Ford Algorithm


```
1  Bellman(G,omega,s)
2     initialisation de d[]   /* initialisées à + infty sauf d[s] =0 */
3     initialisation de PI[]  /* initialisées à NIL                  */
4
5     Pour i=1 à |S|-1 faire
6        Pour chaque arc (u,v) de A
7           relacher(u,v,omega)
8        Fin Pour
9     Fin Pour
10
11    Pour chaque arc (u,v) de A faire
12      si d[v] > d[u] + omega(u,v) alors
13        Renvoyer("présenced'un cycle de poids négatif")
14
15    Retourner Vrai.
16 Fin
```

- Phase d'initialisation : $\Theta(|S|)$
- Chacun des $|S| - 1$ passages prennent $\Theta(|A|)$
- La dernière boucle se fait en $O(|A|)$
- Ainsi la complexité globale de l'algorithme est en $O(|S| \times |A|)$

A few words on the validity of the algorithm

Lemma

Si $G = (S, A)$ ne contient aucun circuit de poids négatif accessible à partir de s, alors, après l'algorithme, on a $d[v] = \delta(s, v)$ $\forall v$ accessible depuis s.

Preuve :

Soit $p =< s, v_1, v_2, ...v_{k-1}, v >$ un plus court chemin de s vers v. On peut considérer que ce chemin est élémentaire : en effet, s'il passe deux fois par le même sommet, la contribution du circuit est positive ou nulle (pas de circuit de poids négatif). Éventuellement, le poids du circuit peut être nul, mais dans ce cas là, il existe un chemin élémentaire de même poids (construit à partir du chemin initial en supprimant le circuit). On en déduit que $k \leq |S| - 1$.
Montrons par récurrence qu'on a $d[v_i] = \delta(s, v_i)$ après le i-ème passage sur les arcs de G.

- pour v_0 : $d[v_0] = \delta(s, v_0) = 0$ après initialisation.
- on suppose que $d[v_{i-1}] = \delta(s, v_{i-1})$ après le $(i-1)$-ème passage. L'arc (v_{i-1}, v_i) est relaché au cours du i-ème passage. Donc, $d[v_i] = \delta(s, v_i)$.

□

CÔTE D'AZUR

A few words on the validity of the algorithm

Algorithmics for Biology
Jean-Paul Comet
Complexity
Pat. Matching
Graphs
Intro
Definitions
Adjacency List
Adjacency Matrix
BFS
DFS
Topological sorting
strongly. connect.
Shortest path
Dijkstra
Bellman–Ford
Dyn.Prog.
Sequences

Theorem

Soit $G = (S, A)$ un graphe.

- *Si G ne contient aucun circuit de poids négatif accessible à partir de s, alors l'algorithme retourne les valeurs de d et on a : $d[v] = \delta(s, v)$ $\forall v$ accessible depuis s.*
- *Si G contient un circuit de poids négatif accessible à partir de s, alors l'algorithme le signale.*

Preuve :

laissée au lecteur