Chapitre 4 : chemins de longueur extrémale

1 Présentation du problème

Dans un graphe orienté valué, un problème souvent rencontré est celui de rechercher les chemins dont la "*longueur*" est minimale ou maximale. Nous nous limiterons à la recherche de chemins de longueur minimale bien que tous les algorithmes puissent s'adapter à la recherche de chemins de longueur maximale.

1.1 Définition

Soit G=(X,U) un graphe orienté, sans boucle, comportant n sommets.
A chaque arc (i,j)∈U est associé la longueur *l(i,j)* de cet arc; cette longueur est un nombre réel et sera aussi appelée poids ou valuation de l'arc (i,j).

Définition. *Soit* $C=(x_0,\ldots,x_p)$ *un chemin quelconque. On définit la longueur de ce chemin par :*

$$l(C)=\sum_{i=0,\ldots,p-1} l(x_i,x_{i+1})$$

On dit qu'un chemin C reliant un sommet i à un sommet j est dit de longueur minimale ou maximale s'il minimise ou maximise cette longueur l(C).

On notera δ(i,j) la longueur du *plus court chemin*, ou *du plus long chemin* de i à j. Il est à remarquer, qu'entre deux sommets i et j, il peut exister plusieurs chemin de longueur optimale δ(i,j). Par abus de langage, nous appellerons chemins optimaux ou chemins minimaux les chemins de longueur minimale. Le résultat suivant est un premier résultat concernant les chemins de longueur minimales.

Proposition. Soit $C=(x_0,\ldots,x_i,\ldots.x_p)$ un chemin optimal entre x_0 et x_p. Soit x_i un sommet de ce chemin. Les sous-chemins (chemins inclus dans le chemin C) $(x_0,\ldots,x_i)$ et $(x_i\ldots,x_p)$ sont optimaux. Autrement dit, tout sous-chemin d'un chemin optimal est optimal.

Exercice. *Prouver ce résultat.*

1.2 Problèmes d'existence

Le problème d'existence de la recherche de chemin de longueur minimale n'est défini que si le graphe ne comporte *aucun circuit de longueur négative*. De tels circuits sont parfois appelés *circuits absorbants*. Supposons qu'il existe un circuit de longueur négative passant par deux sommets i et j. Dans ce cas, il n'existe pas de chemin de longueur minimale entre i et j. En effet, plus on "tourne" sur le circuit absorbant, plus la longueur va diminuer.

Proposition. Soit G un graphe. Il existe un plus court chemin entre deux sommets quelconques u et v du graphe si et seulement si le graphe est sans circuits absorbants.

Afin de pallier ceci, on pourrait limiter la recherche aux chemins élémentaires (on interdit de passer plusieurs fois par le même sommet). Cette recherche a un sens car le nombre de chemins élémentaires est finie. Malheureusement le résultat suivant montre que ce problème n'est pas traitable informatiquement (car sa complexité est trop élevée).

Proposition. La recherche de chemins *élémentaires* optimaux est NP-difficile.

1.3 3 types de problèmes

A. Etant donné deux sommets s et t, trouver un plus court chemin de s à t. Nous ne présenterons pas d'algorithme spécifique à ce type de problèmes. Nous verrons une adaptation de l'algorithme de Dijkstra (algorithme de type B).

B. Etant donné un sommet de départ s, trouver un plus court chemin de s vers tout autre sommet. Nous verrons deux algorithmes pour résoudre ce problème : l'algorithme de Bellman-Ford et celui de Dijkstra. Le premier, le plus générale est de complexité O(m.n) alors que le second se limitant aux valuations positives et en $O(n^2)$.

C. Trouver un plus court chemin entre tout couple de sommets. Nous présenterons l'algorithme de Floyd-Warshall de complexité $O(n^3)$. Précisons que la structure de données utilisée par cet algorithme est une matrice d'adjacence.

2 Algorithme de Bellman-Ford (type B)

2.1 Algorithme

L'algorithme de Bellman résout le problème de la recherche de plus courts chemins dans le cas le plus général, puisque l'on ne s'interdit que la présence des circuits de longueur négatives (dans ce cas la recherche de plus courts chemins n'a pas de sens).
Tout sommet u est marqué par d[u] qui est la borne supérieure pour la distance minimale de s à u. On diminue progressivement d[u], étape par étape, en relachant les arcs. Le processus de relâchement d'un arc (u,v) consiste en un test permettant de savoir s'il est possible, en passant par u, d'améliorer le plus court chemin jusqu'à v et, si oui, à mettre à jour d[v] et π[v]. Lorsqu'une variation sur d[u] n'est plus possible, d[u] est la distance minimale de s à u.
Le graphe de liaison G_π , appelé graphe des chemins de longueur minimale, est un graphe pour lequel il existe un et un seul chemin de s à u (s'il existe au moins un chemin de s à u dans G) et sa longueur est minimale.

```
Bellman(G,l,s)
    d[s]=0
    pour chaque sommet u≠s
        d[u]:=∞
        π[u]:=nil
    fin pour
    changement:=vrai
    k:=0

    Tant que changement=vrai et k≤n
        k:=k+1
        changement:=faux
        pour chaque arc (u,v)∈U
            si d[v]> d[u]+l(u,v) alors
                d[v]:= d[u]+l(u,v)          // relachement de l'arc (u,v)
                π[v]:=u
                changement:=vrai
            fin si
        fin pour
    fin tant que

    si k=n  alors circuit de longueur négatif
        sinon d[u] longueur minimale de s à u.
```

La première remarque à faire est que l'algorithme est valide quelque soit la procédure de choix de l'arc (u,v) $\in$U dans la boucle sur les arcs. L'algorithme précédent détecte la présence de circuits absorbants s'il y en a et sinon retourne pour chaque sommet la valeur du plus court chemin.

2.2 Validité

Nous ne démontrerons pas cet algorithme, mais nous démontrerons sa version "allégée", adaptée aux graphes sans circuits.

2.3 Complexité

Cet algorithme est composé de 2 boucles imbriquées : une de 1 à p≤n et l'autre sur les arcs. Sa complexité est donc en

O(m.n)

3 Algorithme de Bellman adapté au cas des graphes sans circuit (type B)

3.1 Algorithme

L'algorithme précédent se simplifie lorsque le graphe est sans circuits. Dans ce cas, on adopte une numérotation topologique (voir chapitre 3).

Bellman_sans_circuits(G,l,s) *// on suppose que s est sans prédécesseur*

```
Renuméroter les sommets dans un ordre topologique tel que s=1
d[1]=0
pour chaque sommet i>1
        d[i]:=∞
        π[i]:=nil
fin pour
pour j:=2 à n
        pour chaque prédécesseur i du sommet j
                si d[j]> d[i]+l(i,j) alors
                        d[j]:= d[i]+l(i,j)        // d[j]=min(d[i]+l(i,j) : i prédécesseur de j)
                        π[j]:=i
                fin si
        fin pour
fin pour
```

3.2 Validité de l'algorithme

Proposition. Pour tout j≤n, la valeur d[j] calculée par l'algorithme de Bellman vérifie

$$d[j]=\delta(s,j)$$

Preuve (par récurrence). *Montrons par récurrence la proposition.*
$d[s]=\delta(s,s)=0$ est vérifiée trivialement. Il reste à montrer que

$$\forall k=1\ldots j-1,\ d[k]=\delta(s,k) \Rightarrow d[j]=\delta(s,j) \text{ pour tout } j=1\ldots n$$

A l'étape j, 2 cas se présentent :

i. *Il existe un chemin de s à j.*
   Soit $C_k=(s,\ldots,k,j)$ un chemin de longueur minimale de s à j passant par un prédécesseur k de j. Comme la numérotation des sommets est topologique, k<j. Donc par hypothèse de récurrence la longueur du chemin optimal entre s et k est déjà calculée à l'étape j et vaut d[k]. Ainsi la longueur du chemin C_k vaut d[k]+l(k,j). Ceci permet de conclure que

$$\delta(s,j)=\min(d[k]+l(k,j) : k \text{ prédécesseur de } j)=d[j]$$

ii. *Il n'existe pas de chemin entre s et j.*
   Soit j ne possède pas de prédécesseurs auquel cas d[j] n'est pas remis à jour par l'algorithme et reste infini. Soit j possède des prédécesseurs i "non atteignables" depuis s. Aussi par hypothèse de récurrence, $d[i]=\infty$ pour tout prédécesseur i de j, et donc

$$d[j]=\min(d[k]+l(k,j) : k \text{ prédécesseur de } j)=\infty=\delta(s,j)$$

□

3.3 Complexité

La numérotation topologique des sommets se fait en O(m+n) (voir chapitre 3 du cours). Puis on exécute une boucle sur les sommets dans laquelle on exécute une boucle sur les successeurs du sommet traité. La complexité de ces deux boucles est donc en O(m+n). Aussi la complexité de l'algorithme est en

O(m+n)

Cette complexité est à comparer à celle de l'algorithme générale de Bellman en 0(m.n). L'amélioration est très significative.

4 Algorithme Dijkstra (type B)

4.1 Algorithme

L'algorithme de Dijkstra résout le problème de la recherche du plus court chemin à origine unique dans le cas où tous les arcs ont des poids positifs (oui nuls). Cet algorithme maintient à jour un ensemble E de sommets x pour lesquels le plus court chemin de s à x a déjà été calculée. A chaque itération, on ajoute un sommet à E jusqu'à ce que E couvre l'ensemble des sommets.

Dijkstra(G,l,s)
    d[s]=0
    pour *chaque sommet* $u \neq s$
        $d[u] := \infty$
        $\pi[u] := nil$
    fin pour
    $E := \varnothing$
    tant que $E \neq X$
        choisir *un sommet* $u \notin E$ *tel que d[u] soit minimal*
        $E := E \cup \{u\}$
        pour *chaque successeur* $v \notin E$ *de u*
            si $d[v] > d[u]+l(u,v)$ ***alors***
                $d[v] := d[u]+l(u,v)$
                $\pi[v] := u$
            fin si
        fin pour
    fin tant que

Une fois qu'un sommet u rentre dans E alors d[u] n'est plus modifié et est égal à δ(s,u). Aussi cet algorithme peut être adapté plus facilement pour résoudre un problème de type A. En effet, pour trouver le plus court chemin entre deux sommets u, v donnés, il suffit de lancer Dijkstra (G,u) et de s'arrêter lorsque v intègre E.

Exercice. Montrer que l'algorithme n'est pas valide lorsque les valuations ne sont pas positives

4.2 Validité de cet algorithme.

On montre la proposition suivante.

Proposition. La valeur *d[v]* calculée par l'algorithme est égale à δ(s,v)

Preuve. *Raisonnons par récurrence en supposant que* $d[u] = \delta(s,u)$ *pour tous les sommets qui ont intégré E. Supposons que pour un sommet* $u \in X \setminus E$, $d[u] > \delta(s,u)$ *au moment où u intègre E. Soit C un*

chemin de longueur $\delta(s,u)$. On appelle v le premier sommet du chemin C tel que $v \notin E$ et w le prédécesseur de v dans le chemin C. Comme v n'est pas choisi au moment où u intègre E,

$d[w]+l(w,v) \geq d[v] \geq d[u]$ *// au moment où w a integer E, d[v] devient inférieur à* $d[w]+l(w,v)$

Or ,

$l(C)=\delta(s,u) \geq d[w]+l(w,v)$ *// car les valuations sont positives*

Donc,

$\delta(s,u) \geq d[u]$

Ceci contredit l'hypothèse et prouve le résultat.

Complexité

Choisir un sommet $u \in X \backslash E$ tel que d[u] soit minimal requiert un temps de O(n) et cette ligne est exécutée n fois. Le temps total passé à exécuter cette ligne est donc en $O(n^2)$. Quand aux lignes qui se trouvent dans la boucle sur les successeurs sont exécutée m+n fois. Le temps total passée à exécuter ces lignes est en O(m+n). Comme $m \leq n$, la complexité de l'algorithme est en

$O(n^2)$

5 Algorithme de Floyd-Warshall

5.1 Algorithme

Intéressons nous au problèmes de type C, i.e. la recherche de plus court chemins entre tout couple de sommets. Une première approche serait de lancer un de l'algorithmes de Bellman sur tous les sommets du graphes. La complexité serait donc de n fois la complexité de l'algorithme de Bellman à savoir $O(m.n^2)$. Floyd et Warshall proposent un algorithme très simple de complexité $O(n^3)$, ce qui est avantageux lorsque le nombre d'arcs est élevé. La seule hypothèse requise pour cet algorithme est que le graphe ne possède pas de *circuits absorbants*. La structure de données utilisée pour décrire ce graphe est, une fois n'est pas coutume, une matrice d'adjacence $M=[a_{ij}]$ adaptée aux graphes valués définie par:

$$a_{ij}=l(i,j)$$

Floyd-Warshall(G,l)

$d_{ij}^{0}:=l(ij)$
pour *tout k:=1 à n*
  pour *tout i:=1 à n*
    pour *tout j:=1 à n*
      $d_{ij}^{k}:=min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)})$
    fin pour
  fin pour
fin pour

Il est à remarquer que cet algorithme fournit la longueur du plus court chemin entre i et j mais ne donne pas le chemin lui-même ; une modification, non immédiate, de l'algorithme précédent est nécessaire.

5.2 Validité

Pour prouver la validité de l'algorithme, on adoptera les notations suivantes :

- $E_{i,j,k}$ l'ensemble de tous les chemins entre i et j n'empruntant que des sommets de numéros inférieurs ou égaux à k et $\delta^{(\leq k)}(i,j)$ la longueur du chemin optimal de cet ensemble.
- $G_{i,j,k}$ l'ensemble de tous les chemins entre i et j n'empruntant que des sommets de numéros inférieurs ou égaux à k et passant par k et $\delta^{(=k)}(i,j)$ la longueur du chemin optimal de cet ensemble.

Proposition. Lorsque l'algorithme se termine, pour tout k≤n

$$d_{ij}^{\,k}=\delta^{(\leq k)}(i,j)$$

preuve

La proposition précédente se montre par récurrence. $\forall(i,j)\in\{1,\dots,n\}^2\ d_{ij}^{(0)}=s_{ij}^{(0)}$ *est vérifiée (lors de l'initialisation de l'algorithme. Soit* $k<n$. *Supposons que* $\forall(i,j)\in\{1,\dots,n\}^2\ d_{ij}^{(k-1)}=\delta^{(\leq k-1)}(i,j)$ *et montrons que* $\forall(i,j)\in\{1,\dots,n\}^2\ d_{ij}^{(k)}=\delta^{(\leq k)}(i,j)$. *2 cas se présentent :*

- $\delta^{(=k)}(i,j)<\infty$
  Soit $C=(i,i_1,\dots,i_p,k,j_1,\dots j_q,j)$ *un chemin optimal de* $G_{i,j,k}$. *Comme il n'existe pas de circuits absorbants, le sommet k n'apparaît qu'une seule fois dans le chemin C , i.e.* $i_1,\dots i_p, j_1\dots j_q$ *inférieurs ou égaux à k-1*
  En utilisant le fait que tout sous chemin d'un chemin optimal est optimal, les sous-chemins $(i,i_1,\dots,i_p,k)$ *entre i et k et* $(k,j_1,\dots j_q,j)$ *entre k et j sont optimaux.*
  La longueur du chemin C est donc égale à
  $$\delta^{(=k)}(i,j)=\delta^{(\leq k-1)}(i,k)+\delta^{(\leq k-1)}(k,j)$$
- $\delta^{(=k)}(i,j)=\infty$
  Il n'existe pas de chemin n'empruntant que des sommets de numéros inférieurs à k et passant par k. Aussi $E_{i,k,k-1}=\varnothing$ *ou* $E_{k,j,k-1}=\varnothing$ *(sinon il suffirait de concaténer un chemin de* $E_{i,k,k-1}$ *et de* $E_{k,j,k-1}$ *pour constituer un chemin de* $G_{i,j,k}$*). De ceci découle que*
  $$\delta^{(\leq k-1)}(i,k)+\ \delta^{(\leq k-1)}(k,j)=\infty=\ \delta^{(=k)}(i,j)$$

Aussi dans les 2 cas,

$$\delta^{(=k)}(i,j)=\delta^{(\leq k-1)}(i,k)+\delta^{(\leq k-1)}(k,j)$$

Comme $E_{i,j,k}=E_{i,j,k-1}\cup G_{i,j,k}$, $\delta^{(\leq k)}(i,j)=min\ (\delta^{(=k)}(i,j)\ ,\ \delta^{(\leq k-1)}(i,j)\ )$. *Par hypothèse de récurrence,*

$$d_{ij}^{(k-1)}=\delta^{(\leq k-1)}(i,j)$$

On conclut que :

$$\delta^{(\leq k)}(i,j)=min(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)}+d_{kj}^{(k-1)})=\ d_{ij}^{(k)}$$

□

5.3 Complexité

Cet algorithme est basé sur 3 boucles imbriquées sur les sommets. La complexité est donc en

$$\mathbf{O(n^3)}$$

6 Récapitulatif

Récapitulons les diffétentes complexités ainsi que les différentes conditions d'applications des algorithmes précédemment cités.

Nom	**Type**	**Validité**	**Complexité**
Bellman-ford	B	toujours	O(m.n)
Bellman_sans_circuits	B	graphe sans circuit	O(m+n)
Dijsktra	B	valuations positives	$O(n^2)$
Floyd-Warshall	C	toujours (sans circuits absorbant)	$O(n^3)$