

Variations sur les algorithmes de plus courts chemins

Michel Habib
M1 Algo Avancé 2011

2 mars 2011

Plan

- 1 L'amélioration proposée par Torben HAGERUP
 - Principe
 - Notations
 - Principe
 - Validité
 - Complexité
 - Conclusion
- 2 L'algorithme A* de Hart, Nilsson et Raphael

Une version simplifiée

DIJKSTRA

```
1  $d(s) \leftarrow 0$ 
2 forall  $x \in X \setminus \{s\}$  do
3    $d(x) \leftarrow \infty$ 
4  $Ouverts \leftarrow X$ 
5 while  $Ouverts \neq \emptyset$  do
6   Choisir  $z \in Ouverts$  tel que  $d(z)$  soit minimal
7    $Ouverts \leftarrow Ouverts \setminus \{z\}$ 
8   forall  $y \in X$  avec  $zy \in U$  do
9      $d(y) \leftarrow \min\{d(y), d(z) + \omega(z, y)\}$ 
```

Pour aller à l'essentiel en matière de complexité, on néglige la distinction OUVERTS versus FERMES

- 1 L'amélioration proposée par Torben HAGERUP
 - Principe
 - Notations
 - Principe
 - Validité
 - Complexité
 - Conclusion
- 2 L'algorithme A* de Hart, Nilsson et Raphael

Idée

Le principe de l'algorithme de HAGERUP peut se voir comme une *altération* de l'algorithme de DIJKSTRA.

Au lieu de rechercher un minimum absolu pour le choix d'un nouveau sommet, l'algorithme recherche un minimum «*flou*» (Ligne 6 de l'algorithme)

Il existe des algorithmes dûs à Meyer (2001) ou Golberg (2001) proposant un calcul des SSSP en temps linéaire en moyenne. Mais l'algorithme ci-après dû à Hagerup (2004) est un bon compromis entre efficacité et simplicité. On suppose toujours les valuations des arcs positives.

Conditions sur les Arcs

- $\forall e \in U, \omega(e) \in [0, 1]$
- Distribution uniforme des poids sur les arcs

Conditions sur les Arcs

- $\forall e \in U, \omega(e) \in [0, 1]$
- Distribution uniforme des poids sur les arcs

Définitions

- $k = \lfloor \log_2 m \rfloor$
- $\Delta = \frac{1}{k}$
- $\hat{d}(x) = \begin{cases} d(x) & \text{si } x \in X_1 \\ \lfloor \frac{d(x)}{\Delta} \rfloor \Delta & \text{sinon} \end{cases}$

Conditions sur les Arcs

- $\forall e \in U, \omega(e) \in [0, 1]$
- Distribution uniforme des poids sur les arcs

Définitions

- $k = \lfloor \log_2 m \rfloor$
- $\Delta = \frac{1}{k}$
- $\hat{d}(x) = \begin{cases} d(x) & \text{si } x \in X_1 \\ \lfloor \frac{d(x)}{\Delta} \rfloor \Delta & \text{sinon} \end{cases}$

Partitionnement des sommets

- $X_1 = \{x \in X : \exists y \text{ avec } (y, x) \in U \text{ et } \omega(y, x) < \Delta\}$
- $X_2 = X \setminus X_1$

Algorithme générique de HAGERUP 2004

HAGERUP

```
1  $d(s) \leftarrow 0$ 
2 forall  $x \in X \setminus \{s\}$  do
3    $d(x) \leftarrow n - 1$ 
4  $OVERTS \leftarrow X$ 
5 while  $OVERTS \neq \emptyset$  do
6   Choisir  $z \in OVERTS$  tel que  $\hat{d}(z)$  soit minimal
7    $OVERTS \leftarrow OVERTS \setminus \{z\}$ 
8   forall  $y \in X$  avec  $(z, y) \in U$  do
9      $d(y) \leftarrow \min\{d(y), d(z) + \omega(z, y)\}$ 
```

Remarques sur l'algorithme

- 1 Quand un sommet est sélectionné, il n'est plus remis en cause.

Remarques sur l'algorithme

- 1 Quand un sommet est sélectionné, il n'est plus remis en cause.
- 2 Il faut démontrer que lorsqu'un sommet x est choisi, $d(x)$ est égal à la valeur d'un plus court chemin de s à x .

L'idée des deux ensembles de sommets

On remarque :

$$\forall x \in X_1, \widehat{d}(x) \in]d(x) - \Delta, d(x)]$$

L'idée des deux ensembles de sommets

On remarque :

$$\forall x \in X_1, \widehat{d}(x) \in]d(x) - \Delta, d(x)]$$

Invariant principal identique à celui de l'algorithme de DIJKSTRA

A la fin de l'exploration d'un sommet :

$\forall x \in \mathbf{Fermés} \cup \mathbf{Ouverts}$

$d(x)$ = la longueur du plus court chemin de s à x n'utilisant que des sommets fermés sauf éventuellement x

La preuve se fait par récurrence sur le nombre d'exploration

Preuve de l'invariant

- Si le sommet choisi est dans X_1 , alors sa valeur $d(x)$ est minimale. Preuve identique à celle pour l'algorithme de DIJKSTRA.

Preuve de l'invariant

- Si le sommet choisi est dans X_1 , alors sa valeur $d(x)$ est minimale. Preuve identique à celle pour l'algorithme de DIJKSTRA.
- Si le sommet choisi est dans X_2 , sa valeur $d(x)$ n'est pas nécessairement minimale. La seule différence avec la preuve de DIJKSTRA est pour le sommet x lui-même.
Est-il possible qu'il existe un chemin plus court arrivant en x ?
Non car un tel chemin passerait par un premier sommet non Fermés $t \neq x$. Mais nous avons que $\forall y \in X, d(y, x) > \Delta$.
Ceci est en contradiction avec le choix de x .

- Autre preuve ?

- Autre preuve ?
- Par échange on se ramène à une exécution de DIJKSTRA

Moralité

Il est possible de calculer exactement une arborescence des plus courts chemins, sans prendre le minimum à chaque étape.
En particulier dans le cas du graphe réduit à une étoile,
l'algorithme n'ordonne pas totalement les valuations des arcs.

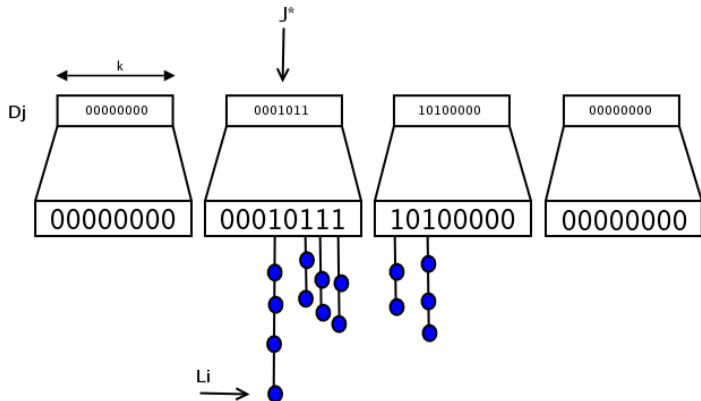
Etudions les implémentations de cette idée algorithmique

Implémentation : Structure de Données

- Les sommets de X_1 sont stockés dans un Tas de Fibonacci.
- Les sommets de X_2 sont stockés dans une liste doublement chaînée.

Implémentation : Structure de Données

- Les sommets de X_1 sont stockés dans un Tas de Fibonacci.
 - Les sommets de X_2 sont stockés dans une liste doublement chaînée.
-
- L_i l'ensemble des sommets de X_2 tel que $\hat{d}(v) = i\Delta$ pour $i \in [0, \dots, nk - 1]$.
 - $I_j = \{i \mid jk \leq i < (j + 1)k \text{ et } L_i \neq \emptyset\}$, $j \in [0, \dots, n - 1]$.
 - I_j peut être représenté comme un entier (de k bits).



L'algorithme T_A

Affinage de la ligne 6 de l'algorithme Générique

- 1 $d \leftarrow \min(\{\widehat{d}(v) \mid v \in OUVERTS \cap X_1\})$
- 2 **while** $l_{j^*} = \emptyset$ and $j^* \leq d$ **do** $j^* \leftarrow j^* + 1$
- 3 $i^* \leftarrow \min(l_{j^*} \cup \{\infty\})$
- 4 **if** $i^* \Delta \leq d$ **then**
- 5 └ Choisir $z \in L_{j^*}$ Arbitrairement
- 6 **else**
- 7 └ Choisir $z \in OUVERTS \cap X_1$ avec $\widehat{d}(z) = d$

Ce petit algorithme calcule bien un sommet ayant un minimum \hat{d}

Implémentation : Algorithme T_B

Motivations

- Abandon des tas de Fibonacci. Car certains trouvent qu'en pratique cette structure de données est complexe.
- Remplacement par un tas binaire.

Algorithme T_B

Affinage de la ligne 6 de l'algorithme Générique

```
1 if  $H \cup L = \emptyset$  then  
2   while  $l_{j^*}$  do  $j^* \leftarrow j^* + 1$   
3    $i^* \leftarrow \min l_{j^*}$   
4   Représenter  $L_{i^*} \cap X_1$  dans  $H$  et  $L_{i^*} \cap X_2$  dans  $L$   
5 if  $L \neq \emptyset$  then  
6   Choisir  $z \in L$  arbitrairement  
7 else Choisir  $z \in H$  tel que  $\hat{d}(z)$  soit minimal
```

L'idée est de représenter tous les sommets dans une même structure et à la demande on en extrait un tas H (éléments de X_1) et une liste L pour les éléments de X_2 .

Rappels sur les opérations pour les Tas de FIBONACCI

Opérations usuelles : complexité amortie

- *insert* en $O(1)$
- *decreasekey* en $O(1)$
- *deletemin* en $O(\log n)$

Complexité de l'algorithme T_A

Complexité en $\Theta(n + m)$

Théorème

La complexité du SSSP selon la méthode de HAGERUP est linéaire quand la distribution des poids sur les arêtes est uniforme.

Complexité de l'algorithme T_A

Complexité en $\Theta(n + m)$

Théorème

La complexité du SSSP selon la méthode de HAGERUP est linéaire quand la distribution des poids sur les arêtes est uniforme.

Démonstration

Il est clair que l'algorithme générique sans la ligne 6 (choix d'un minimum) s'effectue en $O(n + m)$. En ce qui concerne la ligne 6, elle est exécutée $n-1$ fois. l'itération de la ligne 6.2 est effectuée $n-1$ fois de manière globale (amortie), donc $O(n)$. L'algorithme exécute exactement $|X_1|$ appels à la fonction *insert* et *decreasekey* du tas de Fibonacci

Complexité (suite)

Démonstration.

et exécute au plus m fois l'appel à *decreasekey*. Ce qui donne un algorithme en $O(n + m + |X_1| \log n)$ Or $|X_1|$ est borné par le nombre d'arcs de U de poids inférieur à Δ (noté $U(S)$). Comme la distribution est uniforme, $U(S) = m\Delta$.

Ce qui donne $O(n + m + m\Delta \log_2 n)$.

or $\Delta = \lfloor \frac{1}{k} \rfloor = \lfloor \frac{1}{\log_2 m} \rfloor$.

Ce qui donne $O(n + m + m \frac{\log_2 n}{\log_2 m}) = O(n + m)$

Nous avons donc un algorithme en $O(n + m)$ pour calculer le SSSP. à noter que dans le pire des cas l'algorithme s'exécute en $O(m + n \log_2 n)$.



Amélioration possible de l'algorithme T_A ?

Amélioration envisagée

Limiter les éléments de X_1 afin que $|X_1| \leq k$ en ne sélectionnant que les k premiers éléments. Or trouver le k -ième élément se fait en temps linéaire.

Fournissant ainsi un algorithme linéaire pour le problème du plus court chemin à origine unique.

Amélioration possible de l'algorithme T_A ?

Amélioration envisagée

Limiter les éléments de X_1 afin que $|X_1| \leq k$ en ne sélectionnant que les k premiers éléments. Or trouver le k -ième élément se fait en temps linéaire.

Fournissant ainsi un algorithme linéaire pour le problème du plus court chemin à origine unique.

Sous réserve...

que l'invariant soit toujours vérifié !

Question ouverte

Existe-t-il une caractérisation des ordres obtenus par un parcours de type DIJKSTRA ?

A la manière des condition sur 4 points ?

- 1 L'amélioration proposée par Torben HAGERUP
 - Principe
 - Notations
 - Principe
 - Validité
 - Complexité
 - Conclusion

- 2 L'algorithme A* de Hart, Nilsson et Raphael

L'algorithme A*

Si la fonction *choix* fournit un sommet *z* vérifiant :

$d(z) + h(z) = \min_{y \in \text{OUVERTS}} \{d(y) + h(y)\}$ où *h(y)* est une information "heuristique" sur la distance qui reste à parcourir.

et si l'instruction : "Ne rien faire" de l'algorithme de Dijkstra est remplacée par :

- 1 **si** $d(z) + \omega(z, y) < d(y)$ **alors**
- 2 *Parent*(*y*) $\leftarrow z$;
- 3 *d*(*y*) $\leftarrow d(z) + \omega(z, y)$;
- 4 *Ajout*(*y*, *OUVERTS*) ; *Retrait*(*y*, *FERMES*) ;

Algorithme A* proposé par Hart, Nilsson et Raphael 1968

Données: un graphe orienté $G = (X, U)$, une fonction de coût
 $\omega : U \rightarrow \mathcal{R}^+$

Résultat: une arborescence de chemins issus de x_0

- 1 $OUVERTS \leftarrow \{x_0\}$; $FERMES \leftarrow \emptyset$; $Parent(x_0) \leftarrow NIL$;
- 2 $\forall y \neq x_0$, $Parent(y) \leftarrow y$ $d(x_0) \leftarrow 0$;
- 3 $\forall y \neq x_0$, $d(y) \leftarrow +\infty$;
- 4 **tant que** $OUVERTS \neq \emptyset$ **faire**
- 5 Choisir un sommet $z \in OUVERTS$ tel que
 $d(z) + h(z) = \min_{y \in OUVERTS} \{d(y) + h(y)\}$;
- 6 Ajout(z , $FERMES$) ;
- 7 Explorer(z);
- 8 Retrait(z , $OUVERTS$) ;

```
1 Explorer(z);
2 pour Tous les voisins y de z faire
3     si  $y \in \text{FERMES}$  alors
4         si  $d(z) + \omega(z, y) < d(y)$  alors
5             Parent(y)  $\leftarrow z$ ;  $d(y) \leftarrow d(z) + \omega(z, y)$ ;
6             Ajout(y, OUVERTS); Retrait(y, FERMES);
7     sinon
8         si  $y \in \text{OUVERTS}$  alors
9             si  $d(z) + \omega(z, y) < d(y)$  alors
10                Parent(y)  $\leftarrow z$ ;  $d(y) \leftarrow d(z) + \omega(z, y)$ 
11        sinon
12            Ajout(y, OUVERTS) ;
13            Parent(y)  $\leftarrow z$ ;  $d(y) \leftarrow d(z) + \omega(z, y)$ 
```

On suppose que cette fonction $h(y)$ est une information disponible en chaque sommet du graphe. L'usage de cet algorithme est adapté au cas où l'on cherche un plus court chemin d'un sommet x_0 à un sommet t . La valeur de $h(x)$ pour un sommet x donné, étant une estimation de la distance qu'il reste à parcourir pour aller de x à t . Le goulot d'étranglement de complexité provient de la gestion de l'ensembles des OUVERTS. Il faut utiliser une structure de données qui permette le calcul du minimum en $O(\log n)$ ou mieux.

Pour l'algorithme A* le principal problème n'est pas tant celui de la complexité, mais celui de la pertinence du résultat.
Le domaine d'application visé est celui de très grands graphes sur lesquels l'algorithme de DIJKSTRA demanderait trop de temps de calcul.