

# Modèles déterministes et plus court chemin

Fabian Bastin

DIRO, Université de Montréal

IFT-6521 – Hiver 2011

# PDS déterministe et plus court chemin

Pour  $x_0$  fixé, on veut résoudre

$$\min_{\mu_0, \dots, \mu_{N-1}} g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k))$$

$$\text{s.l.c. } \mu_k(x_k) \in U_k(x_k) \text{ et } x_{k+1} = f_k(x_k, \mu_k(x_k)), \quad k = 0, \dots, N-1$$

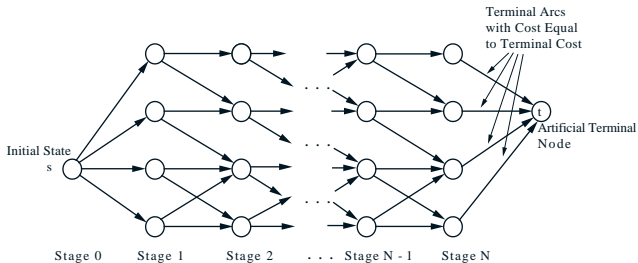
ce qui équivaut à

$$\min_{u_0, \dots, u_{N-1}} g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$$

$$\text{s.l.c. } u_k \in U_k(x_k) \text{ et } x_{k+1} = f_k(x_k, u_k), \quad k = 0, \dots, N-1.$$

Ici, on peut calculer les **décisions optimales**  $u_0, \dots, u_{N-1}$  dès le départ, car aucune nouvelle information n'est obtenue en cours de route.

Si les  $X_k$  et  $U_k$  sont finis, résoudre ce problème équivaut à trouver un plus court chemin dans un réseau, où les noeuds sont tous les états  $(k, x_k)$  possibles, pour  $0 \leq k \leq N$  et  $x_k \in X_k$ , auxquels on ajoute un noeud artificiel  $t$  qui correspond à l'état où tout est terminé (étape  $N + 1$ ). Pour chaque noeud  $(k, x_k)$ ,  $k < N$ , et chaque décision  $u_k \in U_k(x_k)$ , il y a un arc de longueur  $g(x_k, u_k)$  partant du noeud  $(k, x_k)$  et allant au noeud  $(k + 1, x_{k+1})$ , avec  $x_{k+1} = f_k(x_k, u_k)$ . Chaque noeud  $(N, x_N)$  est relié au noeud  $t$  par un arc de longueur  $g(x_N)$ . On cherche un plus court chemin de  $s = (0, x_0)$  à  $t$ .



Si on numérote les noeuds couche par couche, par ordre croissant de valeur de  $k$ , on obtient un **réseau sans cycle et ordonné topologiquement** (i.e., un arc  $(i,j)$  ne peut exister que si  $i < j$ ).

Dans le cas où il n'est pas nécessaire de mémoriser le numéro d'étape, on peut simplifier le réseau en agrégeant des noeuds. Le réseau résultant peut ne plus être ordonné topologiquement.

Inversement, tout problème de recherche d'un plus court chemin dans un réseau peut se formuler comme un problème de PDS déterministe, que l'on peut résoudre par la PD.

# Calcul du plus court chemin dans un réseau.

De nombreux problèmes pratiques se formulent donc comme des problèmes de plus court chemin dans un réseau. On peut les résoudre par l'algorithme du simplexe pour les problèmes de flot, mais les algorithmes que nous allons examiner sont souvent beaucoup plus efficaces.

**Problème:** On cherche le plus court chemin du noeud  $s = 0$  au noeud  $t$ , dans un réseau où les noeuds sont  $\{0, \dots, t\}$  et où chaque arc  $(i, j)$  a une longueur  $a_{ij} \geq 0$ .

**Méthode myope:** Toujours prendre l'arc le plus court, jusqu'à ce qu'on atteigne  $t$ . **Rarement optimal; souvent très mauvais.**

**Force brute:** Essayer tous les chemins possibles. **Trop inefficace.**

## Notation:

$J_i$  = distance minimale du noeud  $i$  au noeud  $t$ ;

$D_i$  = distance minimale du noeud 0 au noeud  $i$ ;

$u_i^*$  = le prochain noeud où il faut aller en partant du noeud  $i$ .

Si on trouve tous les  $u_i^*$ , on aura un chemin optimal.

On a

$$J_t = D_0 = 0; \quad J_0 = \min_{1 \leq i \leq t} (D_i + J_i)$$

pour tout  $i$ , mais cette dernière équation ne nous dit pas comment résoudre.

# A. Méthodes simples pour réseau ordonné

Ordre topologique: Arc  $(i, j)$  existe  $\Rightarrow i < j$ .

## A.1. Détermination itérative, chaînage arrière.

On a les équations de récurrence:  $J_t = 0$  et

$$J_i = \min_{\{j|j>i\}} \{a_{ij} + J_j\}, \quad i < t;$$
$$u_i^* = \arg \min_{\{j|j>i\}} \{a_{ij} + J_j\}.$$

### PROCÉDURE ChaînageArrière;

$J_t \leftarrow 0;$

POUR  $i \leftarrow t - 1$  DESCENDANT À 0 FAIRE

$J_i \leftarrow \infty;$

POUR  $j \leftarrow i + 1$  À  $t$  FAIRE

SI  $a_{ij} + J_j < J_i$  ALORS  $J_i \leftarrow a_{ij} + J_j$  ET  $u_i^* \leftarrow j$ .

On calcule en fait le chemin optimal de chaque noeud  $i$  au noeud  $t$ .  
Cette formulation s'appelle le **problème de la valeur initiale**.

**A.2. Détermination itérative, chaînage avant.** On pose:

$D_j$  = distance minimale du noeud 0 au noeud  $j$ ;

$v_j^*$  = le noeud précédant  $j$  sur le chemin optimal de 0 à  $j$ .

Une fois les  $v_j^*$  obtenus, on retrouve le chemin optimal à reculons.

Sa longueur est  $D_t$ . **Récurrance:**  $D_0 = 0$  et

$$D_j = \min_{\{i|i < j\}} \{D_i + a_{ij}\};$$

$$v_j^* = \arg \min_{\{i|i < j\}} \{D_i + a_{ij}\}.$$

**PROCÉDURE ChaînageAvant;** // Calcule les  $D_j$  et  $v_j^*$ .

$D_0 \leftarrow 0$ ;

POUR  $j \leftarrow 1$  À  $t$  FAIRE

$D_j \leftarrow \infty$ ;

POUR  $i \leftarrow 0$  À  $j - 1$  FAIRE

SI  $D_i + a_{ij} < D_j$  ALORS  $D_j \leftarrow D_i + a_{ij}$  ET  $v_j^* \leftarrow i$ .

On calcule ici le chemin optimal du noeud 0 à chaque noeud  $i$ .

Équivaut au chaînage arrière pour le plus court chemin de  $t$  à 0.

Cette formulation s'appelle le **problème de la valeur finale**.

### A.3. Méthode d'accession (“reaching”).

Mêmes récurrences que pour le chaînage avant, mais on permute les deux boucles “POUR” dans l’algorithme.

**Idée:** on fixe à tour de rôle  $D_1, D_2, \dots$ , et dès que  $D_i$  est fixé, on s’arrange pour que pour tous les sommets  $j$  successeurs,  $D_j$  soit la longueur du plus court chemin de 0 à  $j$  parmi les chemins qui ne peuvent passer que par les sommets de  $\{0, 1, \dots, i\}$ .

Lorsque  $D_i$  est fixé, on dit que  $i$  a une **étiquette permanente**  $D_i$ .

Les  $D_j$  qui ne sont pas encore fixés définitivement sont des **étiquettes temporaires**.

## PROCÉDURE Accession;

$D_0 \leftarrow 0$ ;    POUR  $j \leftarrow 1$  À  $t$  FAIRE  $D_j \leftarrow \infty$ ;

POUR  $i \leftarrow 0$  À  $t - 1$  FAIRE

// Invariant: ici,  $D_i$  est fixé de façon permanente.

// Pour  $j > i$ , les  $D_j$  sont encore temporaires.

POUR  $j \leftarrow i + 1$  À  $t$  FAIRE

SI  $D_i + a_{ij} < D_j$  ALORS  $D_j \leftarrow D_i + a_{ij}$  ET  $v_j^* \leftarrow i$ .

Tous ces algorithmes prennent un temps dans  $O(t^2)$ .

La méthode d'accession devient avantageuse si on peut éliminer des noeuds  $i$  en cours de route. Par exemple, si  $D_i$  est fixé et dépasse la valeur courante (temporaire) de  $D_t$ . On va exploiter cela davantage plus loin.

## B. Algorithmes d'étiquetage.

Il s'agit de généralisations de la méthode d'accession.

Ces méthodes sont en général **plus efficaces**, surtout pour les grands réseaux. Elles n'exigent **pas** que le réseau soit **ordonné topologiquement**. Il peut aussi y avoir des **cycles** et des arcs de longueur négative, mais pas des cycles de longueur négative.

Note: Lorsque le réseau n'est pas ordonné topologiquement, les méthodes de détermination itérative décrites précédemment ne s'appliquent pas.

On suppose ici que  $0 \leq a_{ij} \leq \infty$ . Soient:

- $T$  = ensemble des sommets dont l'étiquette est encore **temporaire**.
- $d_j$  = longueur du plus court chemin de 0 à  $j$  sans passer par  $T$   
=  $\min_{i \notin T} (d_i + a_{ij})$ .
- $v_j$  = le noeud précédant  $j$  sur le meilleur chemin à date de 0 à  $j$ .

Au début, on met tous les sommets dans  $T$ , sauf 0.

À chaque itération, on enlève un sommet  $i$  de  $T$  pour lui donner une **étiquette permanente**:  $D_i = d_i$ . Lorsque  $T$  est vide, on a fini.

**PROCÉDURE Accession**; // Algorithme de Dijkstra.

$d_0 \leftarrow 0$ ; POUR  $j \leftarrow 1$  À  $t$  FAIRE  $d_j \leftarrow a_{0j}$ ;  $v_j \leftarrow 0$ ;

$T \leftarrow \{1, 2, \dots, t\}$ ;

TANTQUE  $T \neq \emptyset$  FAIRE

$i \leftarrow \arg \min_{j \in T} d_j$ ;  $T \leftarrow T - \{i\}$ ;

// Invariant: on a ici  $d_i = D_i$ .

POUR CHAQUE  $j \in T$  FAIRE

SI  $d_i + a_{ij} < d_j$  ALORS  $d_j \leftarrow d_i + a_{ij}$  ET  $v_j \leftarrow i$ .

## Proposition.

S'il existe un chemin de 0 à  $t$ , à la fin de l'algorithme on a  $T = \phi$ ,  $d_j = D_j$ , et  $v_j = v_j^*$  pour tout  $j$ .

On a donc un plus court chemin de 0 à chacun des autres sommets.

**Preuve.** Appelons  $H(n)$  l'hypothèse qui dit que la  $n$ -ième fois que l'on teste si  $T \neq \phi$  dans l'algorithme, on a

(a)  $\forall j \notin T, d_j = D_j$ ;

(b)  $\forall j \in T, d_j =$  longueur du plus court chemin de 0 à  $j$   
sans passer par  $T$ .

(c)  $\forall j, v_j =$  le noeud précédant  $j$  sur le meilleur chemin à date de 0 à  $j$ .

On montre  $H(n)$  pour  $n = 0, \dots, t$ , par **induction sur  $n$** .

(1) L'initialisation rend  $H(0)$  vraie.

(2) Supposons maintenant que  $H(n)$  est vérifiée et montrons que cela implique  $H(n + 1)$ .

On montre d'abord que lorsqu'on enlève  $i$  de  $T$  au  $n$ -ième tour de boucle, on a  $d_i = D_i$ . Si  $d_i \neq D_i$ , le plus court chemin de 0 à  $i$  doit passer par  $T$ , à cause de  $H(n)$ . Soit  $k$  le premier sommet de  $T$  rencontré sur ce chemin. La partie de ce chemin qui va de 0 à  $k$  doit être un plus court chemin de 0 à  $k$ , et sa longueur est  $D_k$ . Donc, la longueur de ce chemin qui passe par  $k$  pour aller à  $i$  est  $\geq D_k \geq d_i$ . Ainsi, il n'y a pas de chemin plus court que  $d_i$  pour aller à  $i$ . On peut donc enlever  $i$  de  $T$  et (a) demeure vrai.

Maintenant que  $i \notin T$ , on peut passer par  $i$  pour aller aux autres sommets de  $T$ . La boucle POUR fait les mises-à-jour pour en tenir compte et restaurer (b) et (c). Si  $i$  devient un sommet intermédiaire sur un plus court chemin,  $i$  sera le dernier sommet intermédiaire, par construction, aussi le chemin nouvellement construit sera de longueur minimale parmi les chemins ne passant pas par  $T$ . Ainsi, après la boucle, on a  $H(n+1)$ .

Par induction, on a donc  $H(1), H(2), \dots, H(t)$ .  $\square$

## Considérations pratiques.

On peut mettre les sommets dans  $T$  seulement lorsque leurs étiquettes deviennent finies.

Lorsqu'on met à jour les étiquettes  $d_j$ , on ne considère que les sommets directement accessibles de  $i$ , i.e., tels que  $a_{ij} < \infty$ .

Pour les grands réseaux, au lieu de stocker tous les  $a_{ij}$  dans une matrice, on maintient une **liste des successeurs** pour chaque noeud  $i$ . On conserve une **liste** des arcs  $(i, j)$  tels que  $a_{ij} < \infty$ .

Il faut aussi une structure de données qui contient tous les noeuds de  $T$  et qui permet de toujours extraire rapidement celui ayant le **plus petit**  $d_j$ . On utilise habituellement des structures arborescentes permettant le retrait du plus petit (à la racine) en  $O(1)$  opérations, et l'insertion ou la mise à jour de l'arborescence en  $O(\log |T|)$  opérations. Exemples: monceau, arbre rouge-noir, "splay tree", etc.

Pour un réseau de  $t$  noeuds avec en moyenne  $n$  arcs émanant de chaque noeud, la quantité totale de **travail** est dans:

$O(t^2)$  si on travaille avec une matrice des  $a_{ij}$  (réseau complet);

$O(nt \log_2 t)$  avec des listes et une bonne arborescence.

**Exemple:** si  $t = 10000$  et  $n = 10$ , on a

$$t^2 = 10^8 \quad \text{et} \quad nt \log_2 t \approx 1.3 \times 10^6 \approx t^2/75.$$

“L’overhead” pour maintenir l’arborescence à jour est quand même importante. Plusieurs raffinements, simplifications, compromis, heuristiques, ..., permettent d’améliorer la performance en pratique.

**Idée:** heuristique qui choisit un noeud dans  $T$  avec un petit  $d_j$  en moyenne, mais pas toujours le plus petit.

Si le  $d_j$  choisi est petit, les chances sont plus grandes qu’il ne soit plus remodifié ( $d_j = D_j$ ), mais ce n’est pas assuré. Il faut ajuster l’algorithme en conséquence.

# Algorithme de correction d'étiquette

Soient:

- $d_j$  = longueur du plus court chemin de 0 à  $j$  à date.
- $v_j$  = le noeud précédant  $j$  sur le meilleur chemin à date de 0 à  $j$ .
- $U$  = UPPER
  - = borne supérieure sur la longueur du plus court chemin.
- $O$  = OPEN
  - = sommets  $i$  dont l'étiquette a été modifiée mais on n'a pas encore ajusté les étiquettes de leurs successeurs  $j$  en vérifiant si  $d_i + a_{ij} < d_j$ .

Au début,  $O$  ne contient que l'origine.

À chaque itération, on enlève un sommet  $i$  de  $O$  et on essaie de réduire les  $d_j$  des successeurs  $j$  de  $i$ , en regardant si  $d_i + a_{ij} < d_j$ .  
Lorsqu'on réduit  $d_j$ , on met  $j$  dans  $O$ . Lorsque  $O$  est vide, on a fini.

## PROCÉDURE CorrectionDétiquette;

$U \leftarrow \infty$ ;  $O \leftarrow \{0\}$ ;  $d_0 \leftarrow 0$ ;

POUR  $j \leftarrow 1$  À  $t$  FAIRE  $d_j \leftarrow \infty$ ;

TANTQUE  $O \neq \emptyset$  FAIRE

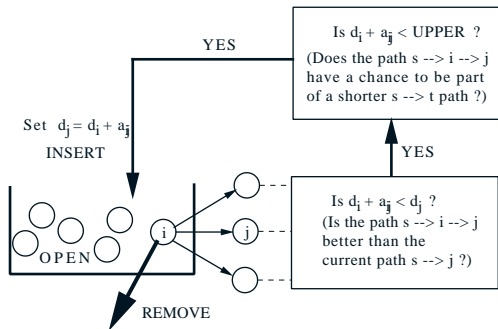
Choisir un  $i$  dans  $O$ ;  $O \leftarrow O - \{i\}$ ;

POUR CHAQUE  $j$  tel que  $a_{ij} < \infty$  FAIRE

SI  $d_i + a_{ij} < \min(d_j, U)$  ALORS

$d_j \leftarrow d_i + a_{ij}$ ;  $v_j \leftarrow i$ ;

SI  $j = t$  ALORS  $U \leftarrow \min(U, d_t)$  SINON  $O \leftarrow O + \{j\}$ .



**Stratégies:** On veut réduire  $U$  (trouver des bons chemins) rapidement, garder la liste  $O$  petite (essayer d'enlever d'abord les plus petits  $d_j$ ), et faire cela avec un minimum d'overhead. Question de compromis.

**Proposition 3.1.** S'il existe un chemin de  $0$  à  $t$ , l'algorithme va se terminer avec  $U = d_t = D_t$  et on aura  $v_j = v_j^*$  sur le chemin optimal, sinon il va se terminer avec  $U = \infty$ .

**Preuve:** On montre d'abord que l'algorithme se termine en temps fini. En effet, chaque fois qu'un noeud  $j$  entre dans  $O$ , son  $d_j$  diminue strictement et correspond à un nouveau plus court chemin de  $0$  à  $j$ . Comme il n'y a qu'un nombre fini de chemins de  $0$  à  $j$  plus courts que la première valeur finie affectée à  $d_j$ , cela ne peut se produire qu'un nombre fini de fois. L'ensemble  $O$  finira donc par se vider.

S'il n'y a pas de chemin de  $0$  à  $t$ , on ne pourra jamais changer  $U$ , donc l'algorithme va se terminer avec  $U = \infty$ .

S'il existe un chemin de 0 à  $t$ , soit  $\ell$  la longueur de l'un d'entre eux. Comme il n'y a qu'un nombre fini de chemins de 0 à  $t$  de longueur  $\leq \ell$ , il en existe un plus court, disons  $(0 = j_0, j_1, j_2, \dots, j_k, j_{k+1} = t)$ , de longueur  $d^* = D_t$ .

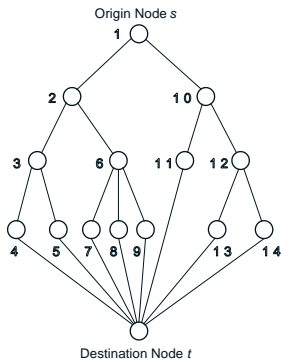
On montre que l'hypothèse  $H(n)$ : " $j_n$  entrera éventuellement dans  $O$  et lorsqu'il en sortira pour la dernière fois, on aura  $d_{j_{n+1}} = D_{j_{n+1}}$ " tient, par induction sur  $n$ , pour  $n = 0, \dots, k$ .

Après le premier tour de la boucle TANTQUE, 0 sera entré puis sorti de  $O$  et on aura  $d_{j_1} = a_{0,j_1} = D_{j_1}$ . Donc  $H(0)$  tient. Montrons maintenant que  $H(n)$  implique  $H(n+1)$ . En supposant  $H(n)$ , lorsque  $j_n$  sortira de  $O$  pour la dernière fois, l'étiquette  $d_{j_{n+1}}$  prendra la valeur  $d_{j_n} + a_{j_n,j_{n+1}} = D_{j_{n+1}}$  et  $j_{n+1}$  entrera dans  $O$  pour la dernière fois, à moins que son étiquette n'ait déjà cette valeur, auquel cas  $j_{n+1}$  sera déjà entré dans  $O$  auparavant lorsque son étiquette a pris cette valeur, et n'y entrera plus. Dans les deux cas, on a  $H(n+1)$ . À la fin,  $H(k)$  tient, et donc  $d_t = D_t$  et  $U = d_t$ .  $\square$

## Exemples de façons de choisir $i$ dans $O$ .

**FIFO:**  $O$  est géré comme une file d'attente, premier arrivé premier servi. Algorithme de Bellman-Ford. Recherche en largeur: les noeuds sont traités couche par couche. Si le réseau est ordonné topologiquement, cela donne notre première procédure d'accèsion.

**LIFO:**  $O$  est géré comme une pile, dernier arrivé premier servi. Recherche en profondeur. On essaie d'atteindre  $t$  le plus vite possible. Moins de mémoire que FIFO et réduit  $U$  plus rapidement.



Plus petit  $d_j$  d'abord (Dijkstra).  $O$  est trié selon les valeurs de  $d_j$ . Équivaut à l'algorithme de Dijkstra, où  $O$  ne contient que les noeuds de  $T$  qui ont une étiquette finie.

Méthode de D'Esposito-Pape.  $O$  est géré comme une file. Lorsqu'on insère un noeud  $j$  dans  $O$ , on le met au début de la file s'il a déjà été dans  $O$ , et à la fin de la file si c'est la première fois.

SLF ("small-label-first"). Lorsqu'on insère un noeud  $j$  dans  $O$ , on le met au début de la file  $O$  si son  $d_j$  est inférieur au  $d_i$  du premier noeud  $i$  de  $O$ , et à la fin de la file sinon. On peut combiner cela avec LLL ("large-label-last"): Chaque fois que l'étiquette  $d_i$  du premier noeud  $i$  de  $O$  est plus grande que la moyenne des étiquettes de  $O$ , on renvoie  $i$  à la fin de la file.

Ces heuristiques ont pour but de favoriser le choix des plus petits  $d_j$  mais sans trop payer en “overhead”. Il y en a d'autres. Ce qui est le plus efficace dépend du problème.

Si le réseau contient un très grand nombre de noeuds dont la plupart ne sont pas intéressants, il devient important de ne pas les visiter tous. Les méthodes de correction d'étiquettes deviennent très avantageuses par rapport aux chaînages avant et arrière lorsqu'elles permettent de visiter beaucoup moins de noeuds.

# Généralisation et Autres Variantes

Supposons qu'à chaque noeud  $j$ , on peut disposer de bornes inférieure et supérieure sur  $J_j$ , la distance minimale de  $j$  à  $t$ :

$$h_j \leq J_j \leq m_j.$$

Supposons aussi qu'on accepte une solution  $\epsilon$ -optimale, i.e., que l'on cherche un chemin de 0 à  $t$  dont la longueur ne dépasse pas  $D_t + \epsilon$ , pour un  $\epsilon > 0$  fixé. Dans l'algo., on calcule les  $h_j$  et  $m_j$  au besoin.

## PROCÉDURE CorrectionDétiquette2;

$U \leftarrow \infty$ ;  $O \leftarrow \{0\}$ ;  $d_0 \leftarrow 0$ ;

POUR  $j \leftarrow 1$  À  $t$  FAIRE  $d_j \leftarrow \infty$ ;

TANTQUE  $O \neq \emptyset$  FAIRE

    Choisir un  $i$  dans  $O$ ;  $O \leftarrow O - \{i\}$ ;

    POUR CHAQUE  $j$  tel que  $a_{ij} < \infty$  FAIRE

        SI  $d_i + a_{ij} < d_j$  ET  $d_i + a_{ij} + h_j < U - \epsilon$  ALORS

$d_j \leftarrow d_i + a_{ij}$ ;  $v_j \leftarrow i$ ;

            SI  $j = t$  ALORS  $U \leftarrow \min(U, d_t)$

            SINON  $O \leftarrow O + \{j\}$ ;  $U \leftarrow \min(U, d_j + m_j)$ .

Plus les bornes  $h_j$  et  $m_j$  sont serrées, moins on aura de noeuds à visiter. Mais les bornes plus serrées coûtent en général plus cher à calculer. Il faut trouver un bon compromis.

L'algorithme de "branch-and-bound" pour optimiser une fonction de variables entières est un cas particulier de ce dernier algorithme. Le réseau est un arbre dont les noeuds sont toutes les solutions partielles ou complètes, et les feuilles sont les solutions complètes (toutes les variables sont fixées).

Un arc de longueur  $h_j - h_i$  va de  $i$  à  $j$  si  $j$  est obtenu de  $i$  en fixant une variable de plus.

On peut ajouter un noeud artificiel  $t$ , et un arc artificiel de longueur 0 reliant chaque feuille à  $t$ .

# Correction d'étiquette avec des longueurs d'arcs négatives

Considérons le problème de trouver un plus court chemin du noeud  $s$  au noeud  $t$ , et supposons qu'il n'y a pas de cycle de longueur négative. Supposons qu'un scalaire  $u_j$  est connu pour chaque noeud  $j$ , qui est une sous-estimation de la plus courte distance de  $j$  à  $t$  ( $u_j$  peut prendre la valeur  $-\infty$  si aucun sous-estimé n'est connu). Considérons une version modifiée de l'itération typique de l'algorithme de correction d'étiquette, où la seconde étape est remplacée comme suit:

```
POUR CHAQUE  $j$  tel que  $a_{ij} < \infty$  FAIRE
  SI  $d_i + a_{ij} < \min(d_j, U - u_j)$  ALORS
     $d_j \leftarrow d_i + a_{ij}$ ;     $v_j \leftarrow i$ ;
  SI  $j = t$  ALORS  $U \leftarrow \min(U, d_t)$  SINON  $O \leftarrow O + \{j\}$ .
```

Il est facile de montrer que cet algorithme se termine avec un plus court chemin, en supposant qu'il existe au moins un chemin de l'origine à  $t$ . (Exercice 2.7 de DPOC)