IFT-6521

PROGRAMMATION DYNAMIQUE

Chapitre 2:

Modèles déterministes et plus court chemin

Pierre L'Ecuyer

DIRO, Université de Montréal

Janvier 2015

PDS déterministe et plus court chemin

Pour x_0 fixé, on veut résoudre

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

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

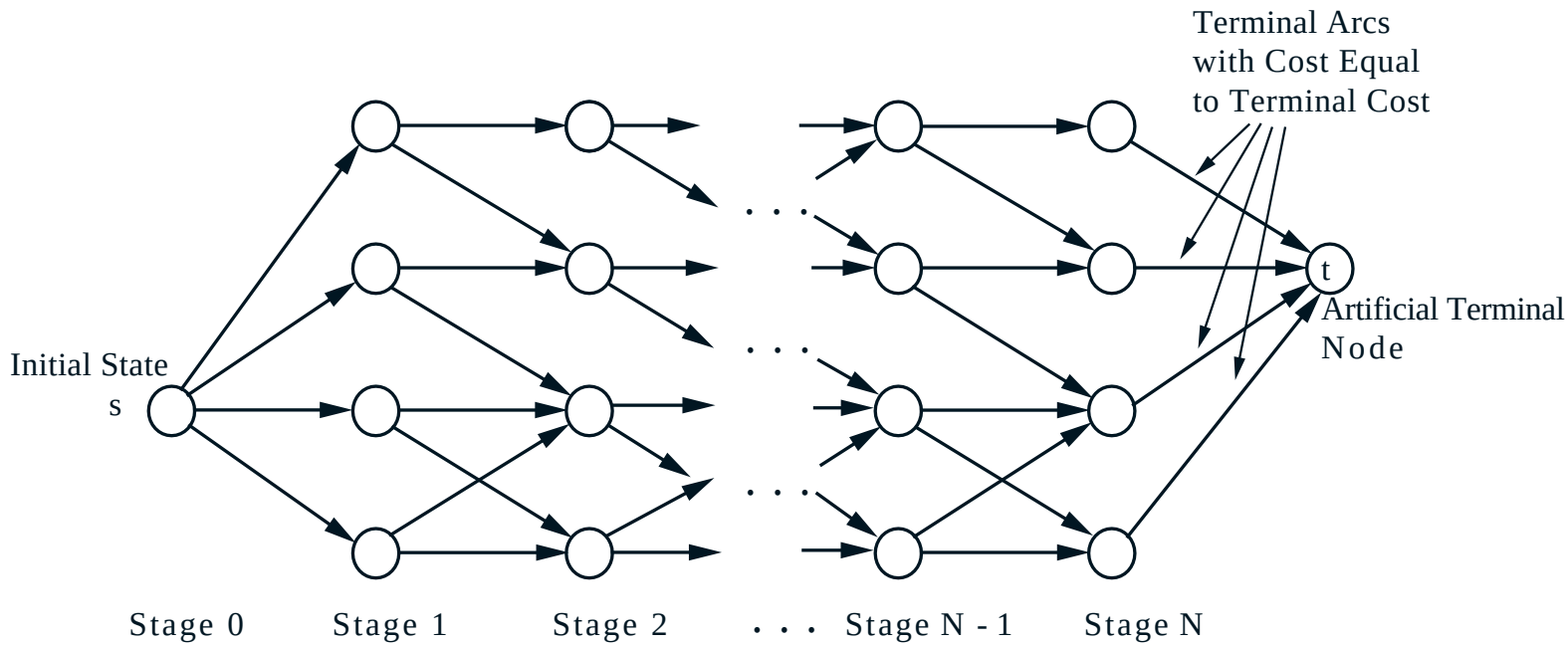
ce qui équivaut à

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

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

Ici, on peut calculer les décisions optimales $u_0, \ldots, u_{N-1}$ 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 $g(x_k, u_k)$ du noeud (k, x_k) au noeud $(k+1,\ x_{k+1} = f_k(x_k, u_k))$.
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 $i < j$).

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. Si on fait cela, il se peut que le réseau résultant ne soit pas ordonné topologiquement.

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. Si on fait cela, il se peut que le réseau résultant ne soit pas 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 flôt, 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 $\{0, \ldots, t\}$ et où chaque arc (i, j) a une longueur $a_{ij} \geq 0$.

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 flôt, 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 $\{0, \ldots, t\}$ 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.

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

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

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

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

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. Récurrence: $D_0 = 0$ et

$$\begin{aligned} D_j &= \min_{\{i|i<j\}} \{D_i + a_{ij}\}; \\ v_j^* &= \arg \min_{\{i|i<j\}} \{D_i + a_{ij}\}. \end{aligned}$$

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 $D_1, D_2, \ldots,$ 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 $\{0, 1, \ldots, i\}$.

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 peut éliminer tout ce qui passe par i. 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:

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

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 $d_j \leftarrow a_{0j}$; $v_j \leftarrow 0$;
$T \leftarrow \{1, 2, \ldots, t\}$;
TANTQUE $T \neq \phi$ 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 // Mise à jour: on peut passer par i
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 $n = 0, \ldots, t$, 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, il sera le dernier sommet intermédiaire, car il est le plus éloigné parmi tous les sommets hors de T (on ne passera pas par i pour aller à un sommet moins loin que i). Ainsi, après la boucle, on a $H(n+1)$. Par induction, on a donc $H(1), H(2), \ldots, H(t)$. $\square$

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 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

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

"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 $O \neq \phi$ FAIRE

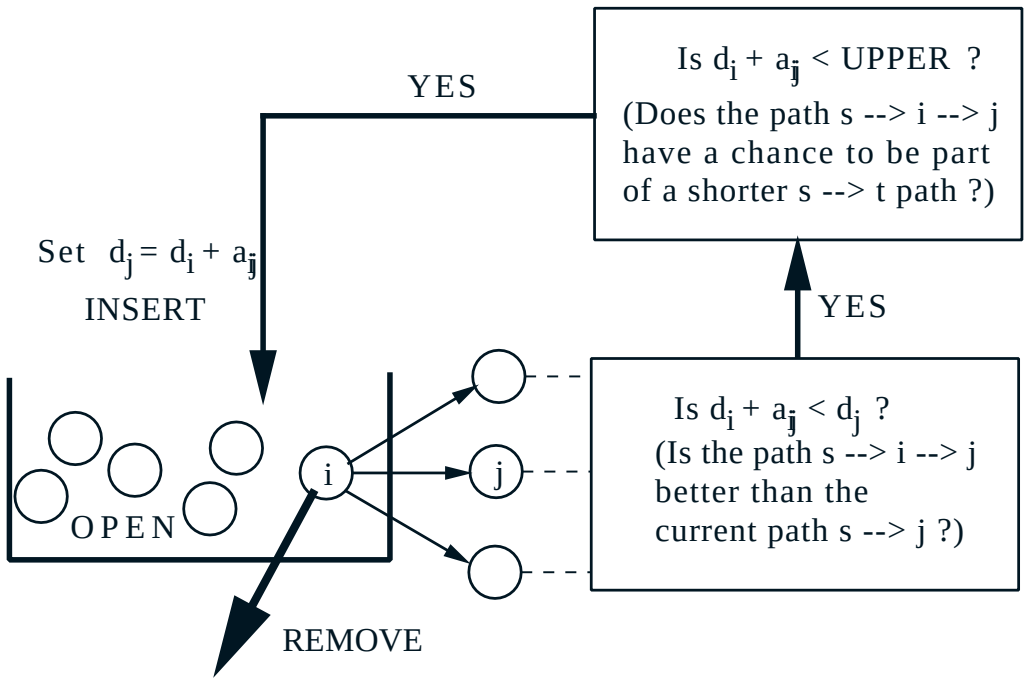
Choisir un i dans O; $O \leftarrow O - \{i\}$;

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 ℓ 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 $(0 = j_0, j_1, j_2, \ldots, j_k, j_{k+1} = t)$, 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 $n = 0, \ldots, k$.

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 $U = d_t$. □

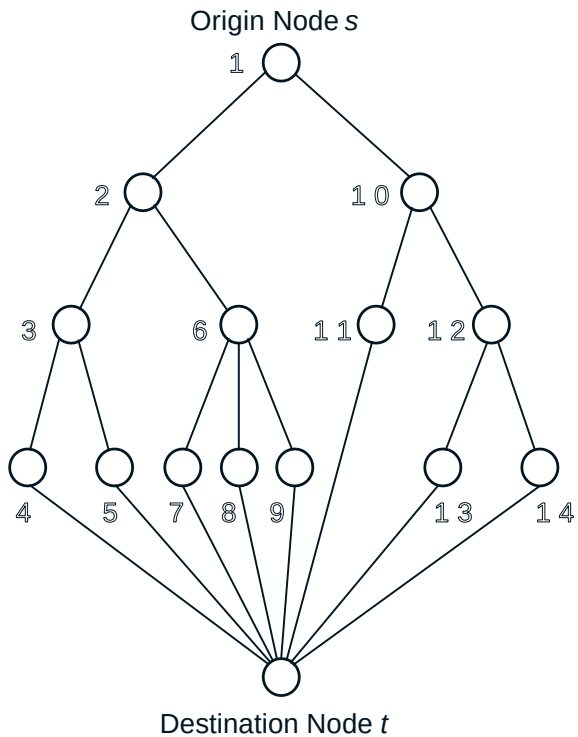
Exemples de façons de choisir i dans O.

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'accession.

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'accession.

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.

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'Esposo-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.

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'Esposo-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 ϵ-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 $O \neq \phi$ 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.

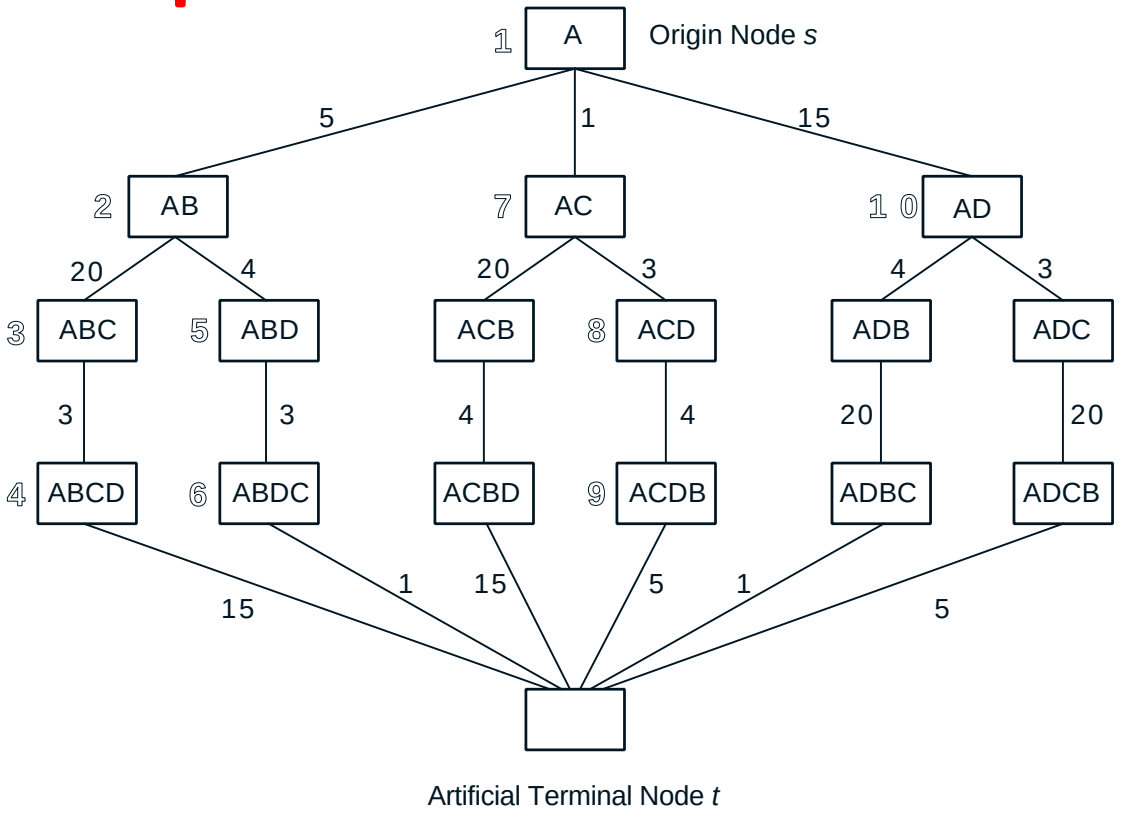
L'algorithme de "branch-and-bound" pour optimiser une fonction de variables entières est un cas particulier de ce dernier algorithme.
Le réseau est un arbre dont les noeuds sont toutes les solutions partielles ou complètes, et les feuilles sont les solutions complètes (toutes les variables sont fixées).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

Note: Dans le cas stochastique, aucune méthode de type chaînage avant, accession, ou correction d'étiquette ne peut s'appliquer.

On ne peut pas formuler le problème comme un de plus court chemin, à cause de l'incertitude dans les transitions et les coûts. Pour cette raison, la notion de "coût optimal pour se rendre à l'état x" n'a pas de sens.

On va plutôt utiliser la notion de "coût espéré à partir de l'état x" et un algorithme de chaînage arrière.

Exemples



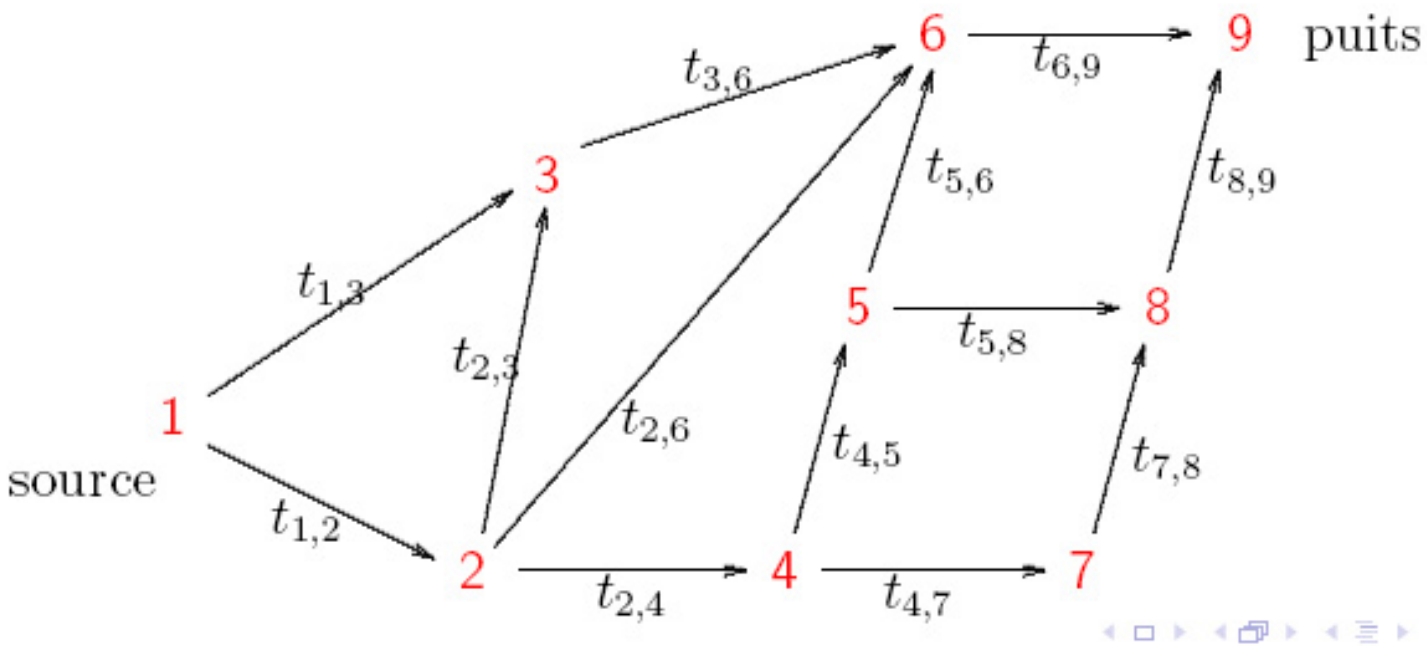
	i	OPEN	UPPER
0	—	1	∞
1	1	2, 7, 10	∞
2	2	3, 5, 7, 10	∞
3	3	4, 5, 7, 10	∞
4	4	5, 7, 10	43
5	5	6, 7, 10	43
6	6	7, 10, 13	13
7	7	8, 10, 13	13
8	8	9, 10, 13	13
9	9	10, 13	13
10	10	$\emptyset$	13

Note: certains noeuds ne sont jamais entrés dans OPEN.

Méthode du chemin critique (PERT/CPM).

PERT: Program Evaluation and Review Technique.
CPM : Critical Path Method.

Un projet est divisé en tâches.
Un graphe représente les relations de précédence entre les tâches.
Noeud i : étape du projet. Arc (i, j): tâche de durée t_{ij}.
Une tâche (i, j) doit se terminer avant que (j, k) débute.



Un plus long chemin dans le réseau s'apelle un chemin critique et sa longueur correspond à la durée minimale du projet.

Pour chaque noeud i, on note
T_i = longueur du plus long chemin de 1 à i.
Correspond au temps minimal pour se rendre à l'étape i.
Équations de récurrence: $T_1 = 0$, et pour $i = 2, \ldots, N$,

$$T_i = \max_{\text{arcs } (j,i)} (T_j + t_{j,i}) .$$

On peut numéroter les étapes de manière à ce que le réseau soit ordonné topologiquement, i.e., pas d'arc (j, i) pour $j > i$. Il suffit alors de calculer $T_1 = 0, T_2, \ldots, T_N$.

Un plus long chemin dans le réseau s'apelle un chemin critique et sa longueur correspond à la durée minimale du projet.

Pour chaque noeud i, on note
T_i = longueur du plus long chemin de 1 à i.
Correspond au temps minimal pour se rendre à l'étape i.
Équations de récurrence: $T_1 = 0$, et pour $i = 2, \ldots, N$,

$$T_i = \max_{\text{arcs } (j,i)} (T_j + t_{j,i}) .$$

On peut numéroter les étapes de manière à ce que le réseau soit ordonné topologiquement, i.e., pas d'arc (j, i) pour $j > i$. Il suffit alors de calculer $T_1 = 0, T_2, \ldots, T_N$.

Ensuite, on peut aussi poser $Y_N = T_N$ et calculer:
Y_i = date au plus tard de l'étape i (sans retarder le projet)

$$= \min_{\{j|(i,j) \text{ existe}\}} Y_j - t_{i,j}, \qquad \text{puis}$$

$E_i = Y_i - T_i$ = écart permis pour l'étape i, pour $i = N - 1, \ldots, 1$.

Exemple: allocation d'une ressource.

On a b unités d'une ressource à allouer à N activités. Posons:

u_k = nombre d'unités de ressource allouées à l'activité k;
$r_k(u_k)$ = revenu pour l'activité k si u_k unités de ressource lui sont allouées.

Formulation:

$$\max \sum_{k=1}^{N} r_k(u_k)$$

$$\text{s.l.c.} \sum_{k=1}^{N} u_k \leq b; \qquad 0 \leq u_k \leq b_k \text{ et } u_k \text{ entier, pour tout } k.$$

Le nombre de solutions possibles est dans $O(b^N)$.

On se ramène à notre cadre de PDS en posant:

$$\begin{array}{lcl} J_k(x) & = & \text{revenu optimal pour les activités } k \text{ à } N \text{ si} \\ & & x \text{ unités de ressource leur sont disponibles;} \\ x_k & = & \text{nombre d'unités disponibles pour les activités } k \text{ à } N. \end{array}$$

Équations fonctionnelles:

$$\begin{array}{lcl} J_{N+1}(x) & = & 0 \quad \forall x \in X_{N+1} \\ J_k(x) & = & \max\limits_{0 \le u \le x} \{r_k(u) + J_{k+1}(x-u)\}, \ k = N, \ldots, 1, \ 0 \le x \le b. \end{array}$$

On se ramène à notre cadre de PDS en posant:

$$\begin{array}{lcl} J_k(x) & = & \text{revenu optimal pour les activités } k \text{ à } N \text{ si} \\ & & x \text{ unités de ressource leur sont disponibles;} \\ x_k & = & \text{nombre d'unités disponibles pour les activités } k \text{ à } N. \end{array}$$

Équations fonctionnelles:

$$\begin{array}{lcl} J_{N+1}(x) & = & 0 \quad \forall x \in X_{N+1} \\ J_k(x) & = & \max\limits_{0 \le u \le x} \{r_k(u) + J_{k+1}(x-u)\}, \ k = N, \ldots, 1, \ 0 \le x \le b. \end{array}$$

Chaînage arrière: fixer dans l'ordre $J_N, J_{N-1}, \ldots, J_1$, et en mémorisant, à chaque noeud, la valeur de u qui fait atteindre la maximum.

On se ramène à notre cadre de PDS en posant:

$$\begin{array}{lcl} J_k(x) & = & \text{revenu optimal pour les activités } k \text{ à } N \text{ si} \\ & & x \text{ unités de ressource leur sont disponibles;} \\ x_k & = & \text{nombre d'unités disponibles pour les activités } k \text{ à } N. \end{array}$$

Équations fonctionnelles:

$$\begin{array}{lcl} J_{N+1}(x) & = & 0 \quad \forall x \in X_{N+1} \\ J_k(x) & = & \max\limits_{0 \le u \le x} \{r_k(u) + J_{k+1}(x-u)\}, \ k = N, \ldots, 1, \ 0 \le x \le b. \end{array}$$

Chaînage arrière: fixer dans l'ordre $J_N, J_{N-1}, \ldots, J_1$, et en mémorisant, à chaque noeud, la valeur de u qui fait atteindre la maximum.

Chaînage avant: Définir $D_i(x)$ = revenu optimal pour les activités 1 à i si on leur alloue x unités de ressource.

Cas de plusieurs ressources.

Au lieu d'avoir un seul type de ressource, on en a m types.
Dans ce cas, le problème se formule et se résoud (théoriquement) exactement de la même façon, sauf que l'on doit interpréter b, x_k, et u_k comme des vecteurs. On pose

b_i = nombre d'unités de ressource de type i disponibles.
x_{ik} = nombre d'unités de ressource de type i disponible pour les activités k à N;
u_{ik} = nombre d'unités de ressource de type i allouées à l'activité k;

$$b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}, \quad x_k = \begin{pmatrix} x_{1k} \\ \vdots \\ x_{mk} \end{pmatrix}, \quad u_k = \begin{pmatrix} u_{1k} \\ \vdots \\ u_{mk} \end{pmatrix}.$$

r_k et J_k sont définis comme pour le cas où $m = 1$, et les équations fonctionnelles sont les mêmes.

Mais ici, le nombre de noeuds dans le réseau devient (si les b_i et u_{ik} sont entiers): $1 + N \times (b_1 + 1) \times \cdots \times (b_m + 1)$.

Par exemple, pour $N = 100$, $m = 1$ et $b = 99$, on a $1 + 10^4$ noeuds.
Mais pour $N = 100$, $m = 100$ et $b_i = 99$, on a $1 + 10^{202}$ noeuds!

Second cas: impensable de résoudre par fixation itérative en pratique.
C'est la malédiction des grandes dimensions!

En pratique, on ne peut traiter que les petites valeurs de m.

Un problème de sac alpin avec variables non bornées.

On place des objets de types $1, \ldots, N$ dans un sac de volume b.
Chaque objet de type k occupe un volume a_k et rapporte un profit c_k.
Soit u_k le nombre d'objets de type k dans le sac. Formulation:

$$\begin{array}{ll} \max & \sum_{k=1}^{N} c_k u_k \\ \text{s.l.c.} & \sum_{k=1}^{N} a_k u_k \leq b; \\ & u_k \geq 0 \text{ et entier, pour } k = 1, \ldots, N. \end{array}$$

Il s'agit d'un problème de programmation linéaire en nombres entiers, avec une seule contrainte.

Si les u_k n'avaient pas à être entiers, le problème deviendrait trivial: il suffirait de remplir le sac de b/a_k unités de l'objet qui a la plus grande valeur de c_k/a_k (profit par unité de volume occupé).

Se résoud comme le problème d'allocation de ressources précédent: à l'étape k, on fixe u_k. Si b et les a_k sont entiers, on a un problème de plus long chemin dans un réseau de $1 + N \times (b + 1)$ noeuds. Mais dans ce cas-ci, on peut faire beaucoup mieux.

Posons $D(x)$ = valeur optimale d'un sac de volume x.
On a $D(0) = 0$ et on va calculer $D(1), D(2), \ldots, D(b)$ par:

$$D(x) = \max_{\{j | a_j \leq x\}} (D(x - a_j) + c_j)\,. \tag{1}$$

On construit un réseau de $b + 1$ noeuds, dans lequel le noeud x correspond à un sac rempli au niveau x. Un arc $(x, x + a_j)$, de "longueur" c_j, correspond à l'ajout d'un objet de type j au sac.
$D(x)$ est la longueur d'un plus long chemin de 0 à x.

On peut faice ceci car il n'y pas de bornes supérieures sur les u_i.

On peut aussi supposer que les objets sont placés dans le sac par ordre décroissant de valeur de a_k (on les trie dans cet ordre), et qu'en cas d'égalité dans (1), on choisit l'objet ayant le plus petit indice j. On cherchera ainsi le plus long chemin de 0 à b, mais seulement parmi les chemins dont les "types" des arcs sont en ordre décroissant. Soit

$$\begin{aligned} v(x) &= \text{le plus petit type d'objet qu'il est optimal de placer dans un sac de volume } x \\ &= \text{le type du dernier arc sur le chemin optimal de 0 à } x \\ &= \min\{j \mid D(x) = c_j + D(x - a_j)\}. \end{aligned}$$

Le j qui fait atteindre le min doit satisfaire $j \leq v(x - a_j)$.
Il suffit de condidérer les objets qui satisfont cette condition.

$$D(x) = \max_{\{j : a_j \leq x \text{ et } j \leq v(x-a_j)\}} (D(x - a_j) + c_j)\,.$$

On calcule $(D(x), v(x))$, pour $x = 1, \ldots, b$, par la méthode d'accession.

PROCÉDURE Accession pour sac alpin;
POUR $x \leftarrow 0$ À b FAIRE $D(x) \leftarrow 0$; $v(x) \leftarrow N$;
POUR $x \leftarrow 0$ À $b-1$ FAIRE
    POUR $j \leftarrow 1$ À $v(x)$ FAIRE
        SI $c_j + D(x) > D(x+a_j)$ ET $x + a_j \leq b$ ALORS
            $D(x+a_j) \leftarrow c_j + D(x)$; $v(x+a_j) \leftarrow j$;
        SI $c_j + D(x) = D(x+a_j)$ ET $x + a_j \leq b$ ET $j < v(x+a_j)$
            ALORS $v(x+a_j) \leftarrow j$;

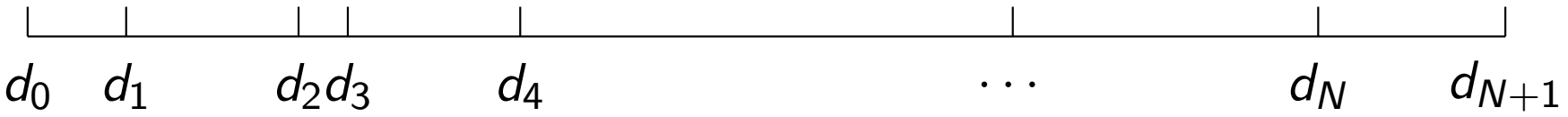
Exemple.

$$\begin{array}{lrcrcrcl} \text{Max} & 6u_1 & + & 7u_2 & + & 3u_3 & & \\ \text{s.l.c.} & 3u_1 & + & 4u_2 & + & 2u_3 & \leq & 10 \\ & u_j \geq 0 \text{ et entier.} & & & & & & \end{array}$$

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$D(x)$														
$v(x)$														

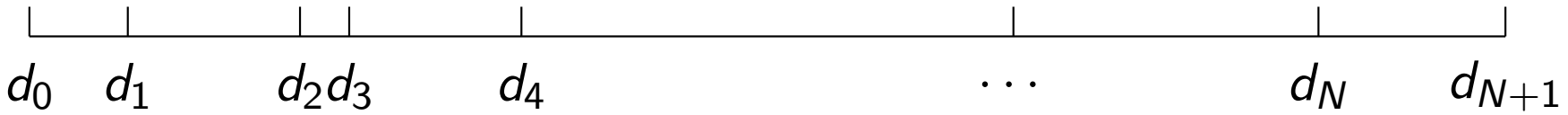
Exemple: un problème de découpe de tissu.

On a une pièce de tissu (en rouleau) de longueur L.
Elle contient des défauts aux points $d_1, d_2, \ldots, d_N$.



Exemple: un problème de découpe de tissu.

On a une pièce de tissu (en rouleau) de longueur L.
Elle contient des défauts aux points $d_1, d_2, \ldots, d_N$.



On veut couper la pièce de tissu pour en vendre les morceaux.
Chaque coupure se fait vis-à-vis d'un défaut et élimine ce dernier.
Soit $V(n, l)$ le prix de vente d'un morceau de tissu de longueur l contenant n défauts. On veut couper la pièce de façon à maximiser le revenu total.
Pour chaque défaut, on a une variable de décision binaire.
Il y a donc 2^N solutions possibles.
Si N est grand, il sera beaucoup trop long de les examiner toutes.

L'approche suivante (programmation dynamique) est plus efficace. Posons
$d_{N+1} = L$;
J_k = la valeur optimale du morceau $[0, d_k]$;
u_k = numéro du dernier défaut où couper avant d_k,
si on dispose du morceau $[0, d_k]$ (i.e. si on coupe à d_k).

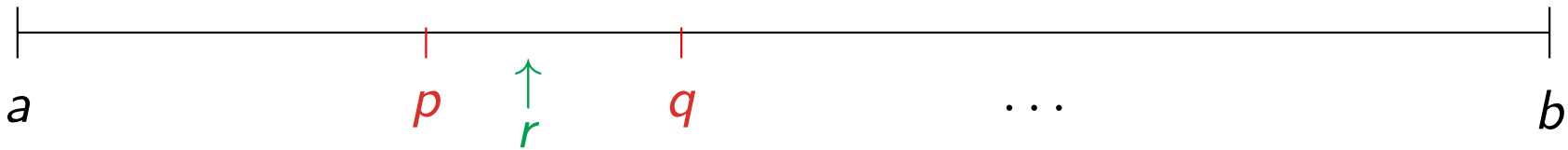
Récurrence:

$$\begin{aligned} J_0 &= 0; \\ J_k &= \max_{0 \le u \le k-1} \{J_u + V(k-u-1,\ d_k - d_u)\}, \\ &\qquad \text{pour } k = 1, 2, \ldots, N+1. \end{aligned}$$

Facile à résoudre par chaînage avant. La valeur optimale de u_k est celle qui fait atteindre le maximum. Si $u_k = 0$, on ne coupera pas le morceau $[0, d_k]$.

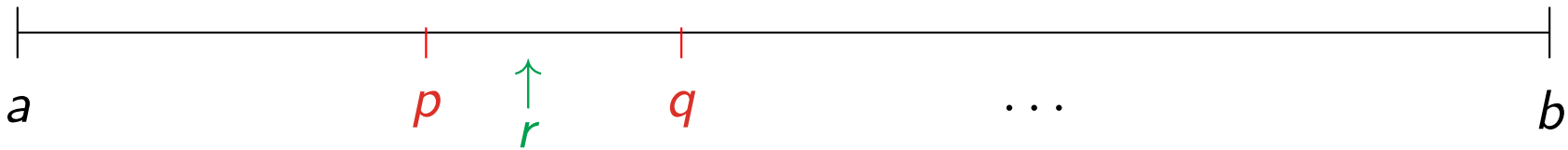
Exemple: la défense d'une frontière.

Le segment $[a, b)$ doit être défendu par N soldats. Chaque soldat va défendre un sous-segment $[p, q)$, en
se plaçant à un point r tel que $p \leq r < q$. Ces sous-segments sont disjoints.



Exemple: la défense d'une frontière.

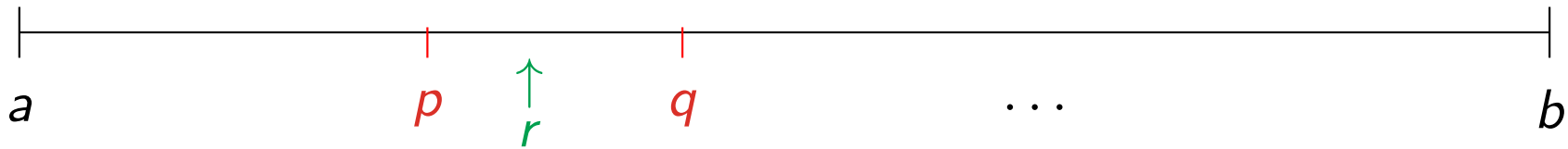
Le segment $[a, b)$ doit être défendu par N soldats. Chaque soldat va défendre un sous-segment $[p, q)$, en
se plaçant à un point r tel que $p \leq r < q$. Ces sous-segments sont disjoints.



Un ennemi qui tente de pénétrer à un point y dans le sous-segment $[p, q)$, défendu par 1 soldat au point r, réussira avec une probabilité $P(p, q, r, y)$. L'ennemi peut savoir où se trouvent nos soldats et choisira une valeur de y de façon à maximiser sa probabilité de pénétration. Nous voulons placer nos soldats de façon à minimiser la probabilité de pénétration de l'ennemi.

Exemple: la défense d'une frontière.

Le segment $[a, b)$ doit être défendu par N soldats. Chaque soldat va défendre un sous-segment $[p, q)$, en
se plaçant à un point r tel que $p \leq r < q$. Ces sous-segments sont disjoints.



Un ennemi qui tente de pénétrer à un point y dans le sous-segment $[p, q)$, défendu par 1 soldat au point r, réussira avec une probabilité $P(p, q, r, y)$. L'ennemi peut savoir où se trouvent nos soldats et choisira une valeur de y de façon à maximiser sa probabilité de pénétration. Nous voulons placer nos soldats de façon à minimiser la probabilité de pénétration de l'ennemi.

Probabilité que l'ennemi réussisse s'il tente de pénétrer par le segment $[p, q)$ et que ce dernier est gardé de façon optimale:

$$G(p, q) = \min_{r \in [p,q]} \left(\max_{y \in [p,q]} P(p, q, r, y) \right).$$

Pour $k = 1, \ldots, N$ et chaque $p \in [a, b)$, soit $J_k(p)$ = probabilité que l'ennemi pénètre par le segment $[p, b)$, s'il est gardé de façon optimale par k soldats. Équations fonctionnelles:

$$\begin{aligned} J_1(p) &= G(p, b) \text{ pour tout } p < b; \quad J_0(b) = 0, \\ J_k(p) &= \min_{q \in [p,b]} \left(\max\left(G(p, q),\ J_{k-1}(q)\right)\right), \\ &\quad \text{pour tout } p, \text{ et } k = 2, \ldots, N. \end{aligned}$$

Interprétation: on a k soldats pour protéger $[p, b)$. Le premier soldat défendra le segment $[p, q)$. Les $k - 1$ autres défendront $[q, b)$ (de façon optimale). Nous choisissons u de façon à minimiser. L'ennemi choisira son segment de façon à maximiser sa probabilité.

Pour $k = 1, \ldots, N$ et chaque $p \in [a, b)$, soit $J_k(p) =$ probabilité que l'ennemi pénètre par le segment $[p, b)$, s'il est gardé de façon optimale par k soldats. Équations fonctionnelles:

$$\begin{aligned} J_1(p) &= G(p, b) \text{ pour tout } p < b; \quad J_0(b) = 0, \\ J_k(p) &= \min_{q \in [p,b]} \left(\max\left(G(p, q),\ J_{k-1}(q)\right)\right), \\ &\quad \text{pour tout } p, \text{ et } k = 2, \ldots, N. \end{aligned}$$

Interprétation: on a k soldats pour protéger $[p, b)$. Le premier soldat défendra le segment $[p, q)$. Les $k - 1$ autres défendront $[q, b)$ (de façon optimale). Nous choisissons u de façon à minimiser. L'ennemi choisira son segment de façon à maximiser sa probabilité.

Pour résoudre, on calcule la fonction J_1, puis J_2, etc.
On fait l'hypothèse que l'on dispose d'une procédure pour calculer $G(p, q)$ au besoin.

Autre difficulté: on a ici un espace d'états continu: il y a une infinité de valeurs de p.
Solution: discrétiser: on ne considére qu'un nombre fini de valeurs possibles pour p et u.

Exemple: les longueurs des segments doivent tous être des multiples de 10 mètres.
Ou encore: on approxime les fonctions G, P et J_k par des polynômes, ou des splines, ou par éléments finis, ...

Autre difficulté: on a ici un espace d'états continu: il y a une infinité de valeurs de p.
Solution: discrétiser: on ne considére qu'un nombre fini de valeurs possibles pour p et u.

Exemple: les longueurs des segments doivent tous être des multiples de 10 mètres.
Ou encore: on approxime les fonctions G, P et J_k par des polynômes, ou des splines, ou par éléments finis, ...

Applications similaires:
— Décider où placer les arrêts d'autobus.
— Quand changer les pneus dans une course automobile.

Algorithme de Viterbi.

Chaîne de Markov cachée (partiellement observée).

$$
\begin{aligned}
X_N &= (x_0, x_1, \ldots, x_N) = \text{ suite des états visités (cachée);} \\
Z_N &= (z_1, \ldots, z_N) = \text{ suite des observations} \\
&\quad \text{(e.g., état observé avec du bruit);} \\
\pi_i &= P[x_0 = i] = \text{ probabilités de l'état initial;} \\
p_{ij} &= P[x_{k+1} = j \mid x_k = i] = \text{ probabilité de transition de } i \text{ à } j; \\
r(z; i, j) &= P[z_{k+1} = z \mid x_k = i, x_{k+1} = j] \\
&= \text{ probabilité d'observer } z \text{ lorsqu'on passe de } i \text{ à } j;
\end{aligned}
$$

On observe Z_N et on cherche à estimer X_N.
On choisit ici l'estimateur de vraisemblance maximale, i.e., $\hat{X}_N$ sera le X_N qui maximise $P[X_N \mid Z_N]$. On va le calculer par PD.

Exemples d'applications:

A. Reconnaissance de la parole ou de l'écriture.
X_N est la suite des phonèmes réellement prononcées par un interlocuteur,
Z_N est la suite des phonèmes comprises par le système.

Les probabilités π_i, p_{ij} et $r(z; i, j)$ du modèle doivent avoir été estimées auparavant: c'est l'entrainement du modèle.

On peut entrainer le modèle pour un interlocuteur particulier (e.g., systèmes de dictée) ou encore pour un vocabulaire particulier (e.g., un répondeur téléphonique reconnaissant la parole ou un interface vocal pour un site internet spécialisé).

Si le système n'est pas suffisamment certain que $\hat{X}_N = X_N$, il pourra demander à l'interlocuteur de confirmer.

B. Transmission de données sur un canal bruité.

On cherche le X_N qui maximise $P[X_N \mid Z_N] = P[X_N, Z_N]/P[Z_N]$.
Mais puisque $P[Z_N]$ ne dépend pas de X_N, il suffit de maximiser

$$\begin{aligned} P[X_N, Z_N] &= P[x_0, x_1, \ldots, x_N, z_1, \ldots, z_N] \\ &= \pi_{x_0} p_{x_0 x_1} r(z_1; x_0, x_1) p_{x_1 x_2} r(z_2; x_1, x_2) \\ &\quad \cdots\cdots p_{x_{N-1} x_N} r(z_N; x_{N-1}, x_N) \\ &= \pi_{x_0} \prod_{k=0}^{N-1} p_{x_k x_{k+1}} r(z_{k+1}; x_k, x_{k+1}). \end{aligned}$$

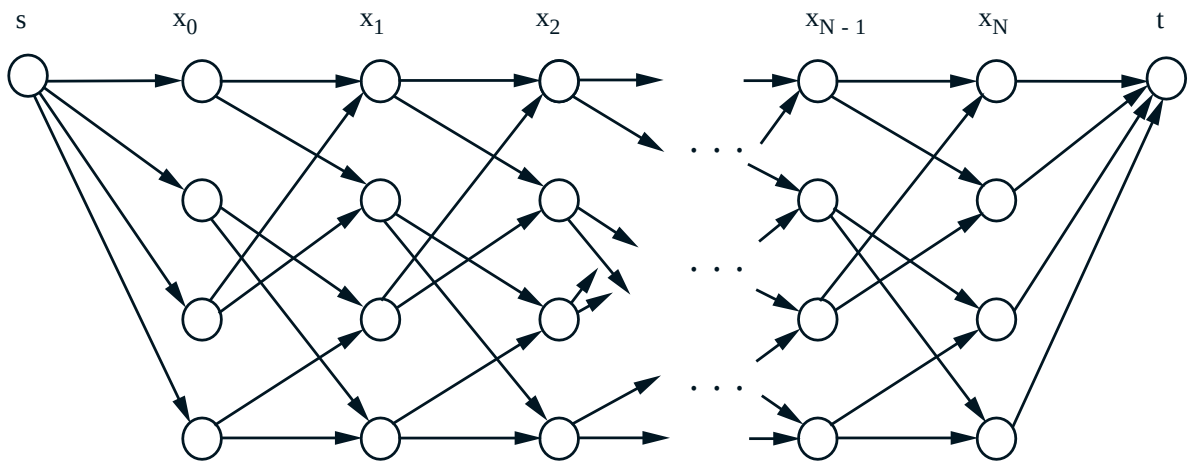
Cela équivaut à minimiser, p.r. à $x_0, \ldots, x_N$,

$$-\ln P[X_N, Z_N] = -\ln(\pi_{x_0}) - \sum_{k=0}^{N-1} \ln[p_{x_k x_{k+1}} r(z_{k+1}; x_k, x_{k+1})].$$

Posons

$$
\begin{aligned}
D_0(x_0) &= -\ln(\pi_{x_0}); \\
D_{k+1}(x_{k+1}) &\stackrel{\text{def}}{=} \min_{x_0,\ldots,x_k} \left(-\ln(\pi_{x_0}) - \sum_{n=0}^{k} \ln[p_{x_n x_{n+1}} r(z_{n+1}; x_n, x_{n+1})] \right) \\
&= \min_{x_k} \left(D_k(x_k) - \ln[p_{x_k x_{k+1}} r(z_{k+1}; x_k, x_{k+1})] \right) \\
&\qquad \text{pour } k = 0, \ldots, N-1.
\end{aligned}
$$

Correspond à trouver un plus court chemin dans le réseau:



On calcule les $D_k(x_k)$ par une méthode de correction d'étiquettes. Avantage p.r. au chainage arrière: on peut débuter l'algorithme dès qu'on a la première observation, et calculer les $D_k(x_k)$ dès que l'on dispose de z_k, pour chaque k, en temps réel.

En pratique on va souvent calculer les n plus courts chemins d'un seul coup.