Modèles déterministes et plus court chemin: exemples.

Fabian Bastin

DIRO, Université de Montréal

IFT-6521 – Hiver 2011

Production-inventaire

Une compagnie doit fournir à son meilleur client trois unités du produit P durant chacune des trois prochaines semaines. Les coûts de production sont donnés dans la table ci-dessous.

Semaine	Production max, temps régulier	Production max, temps supp.	Coût unitaire, temps régulier
1	2	2	300$
2	3	2	500$
3	1	2	400$

Le coût pour chaque unité produite en temps supplémentaire est 100$ de plus que le coût par unité produite en temps régulier. Le coût unitaire d inventaire est de 50$ par semaine

Production-inventaire (2)

Au début de la première semaine, il y a 2 unités du produit dans l inventaire. La compagnie ne veut plus rien avoir dans son inventaire au bout des trois semaines. Combien doit-on produire d unités à chaque semaine afin de rencontrer la demande du client, tout en minimisant les coûts?

Une étape correspond ici à une semaine, $N = 3$, $n = 1, 2, 3$, et

- s_n, l'état au début de la semaine n, est le nombre d'unités de produit dans l inventaire;
- x_n: nombre d unités produites lors de la semaine n;
- $s_{n+1} = s_n + x_n - 3$ (puisque nous devons livrer 3 unités au client à chaque semaine);
- $s_1 = 2$ (puisqu il y a 2 unités au début).

Production-inventaire (3)

Soient c_n, le coût unitaire de production au cours de la semaine n, r_n la production maximale en temps régulier pendant la semaine n, et m_n la production maximale durant la semaine n. Le coût au cours de la semaine n est

$$p_n(s_n, x_n) = c_n x_n + 100 \max(0, x_n - r_n) + 50 \max(0, s_n + x_n - 3).$$

Le coût total vaut dès lors

$$f_n(s_n, x_n) = p_n(s_n, x_n) + f^*_{n+1}(s_{n+1}),$$

et le coût optimal répond à la récurrence

$$f^*_n(s_n) = \min\{p_n(s_n, x_n) + f^*_{n+1}(s_{n+1}) \,|\, 3 - s_n \leq x_n \leq m_n\}.$$

Production-inventaire (4)

Si $n = 3$, nous devons poser

$$f_4^*(s_4) = 0.$$

De plus,

$$s_4 = 0 = s_3 + x_3 - 3,$$

aussi

$$x_3 = 3 - s_3.$$

Calculons d abord les valeurs $f_3^*(s_3)$ et x_3^*. Nous obtenons le tableau

s_3	$f_3(s_3, 3 - s_3)$	$f_3^*(s_3)$	x_3^*
0	1400	1400	3
1	900	900	2
2	400	400	1
≥ 3	0	0	0

Production-inventaire (5)

Voyons maintenant comment nous pouvons calculer les valeurs $f_2^*(s_2)$ et x_2^*, lorsque $s_2 = 0$. Nous devons avoir $x_2 \geq 3$ (car nous devons livrer au moins 3 unités du produit). D'autre part,

$$\begin{aligned} f_2(0,3) &= p_2(0,3) + f_3^*(0) = 1500 + 1400 = 2900, \\ f_2(0,4) &= p_2(0,4) + f_3^*(1) = 2150 + 900 = 3050, \\ f_2(0,5) &= p_2(0,5) + f_3^*(2) = 2800 + 400 = 3200, \\ f_2^*(0) &= \min\{f_2(0,3), f_2(0,4), f_2(0,5)\} = f_2(0,3). \end{aligned}$$

Par conséquent, $x_2^* = 3$. Nous procédons de la même manière pour $s_2 = 1, 2, 3$.

s_2	$x_2 = 0$	$x_2 = 1$	$x_2 = 2$	$x_2 = 3$	$x_2 = 4$	$x_2 = 5$	$f_2^*(s_2)$	x_2^*
0	-	-	-	2900	3050	3200	2900	3
1	-	-	2400	2450	2600	2850	2400	2
2	-	1900	1950	2000	2250	2900	1900	1
≥ 3	1400	1450	1500	1650	2300	2950	1400	0

Production-inventaire (6)

Pour la première étape ($n = 1$), nous avons $s_1 = 2$ (il y a 2 unités au départ dans l'inventaire). Nous devons donc avoir $x_1 \geq 1$. De plus,

$$f_1(2,1) = p_1(2,1) + f_2^*(0) = 300 + 2900 = 3200,$$
$$f_1(2,2) = p_1(2,2) + f_2^*(1) = 650 + 2400 = 3050,$$
$$f_1(2,3) = p_1(2,3) + f_2^*(2) = 1100 + 1900 = 3000,$$
$$f_1(2,4) = p_1(2,4) + f_2^*(3) = 1550 + 1400 = 2950.$$

Par conséquent, $f_1^*(2) = f_1(2,4)$ et $x_1* = 4$. Sous forme de tableau, cela donne:

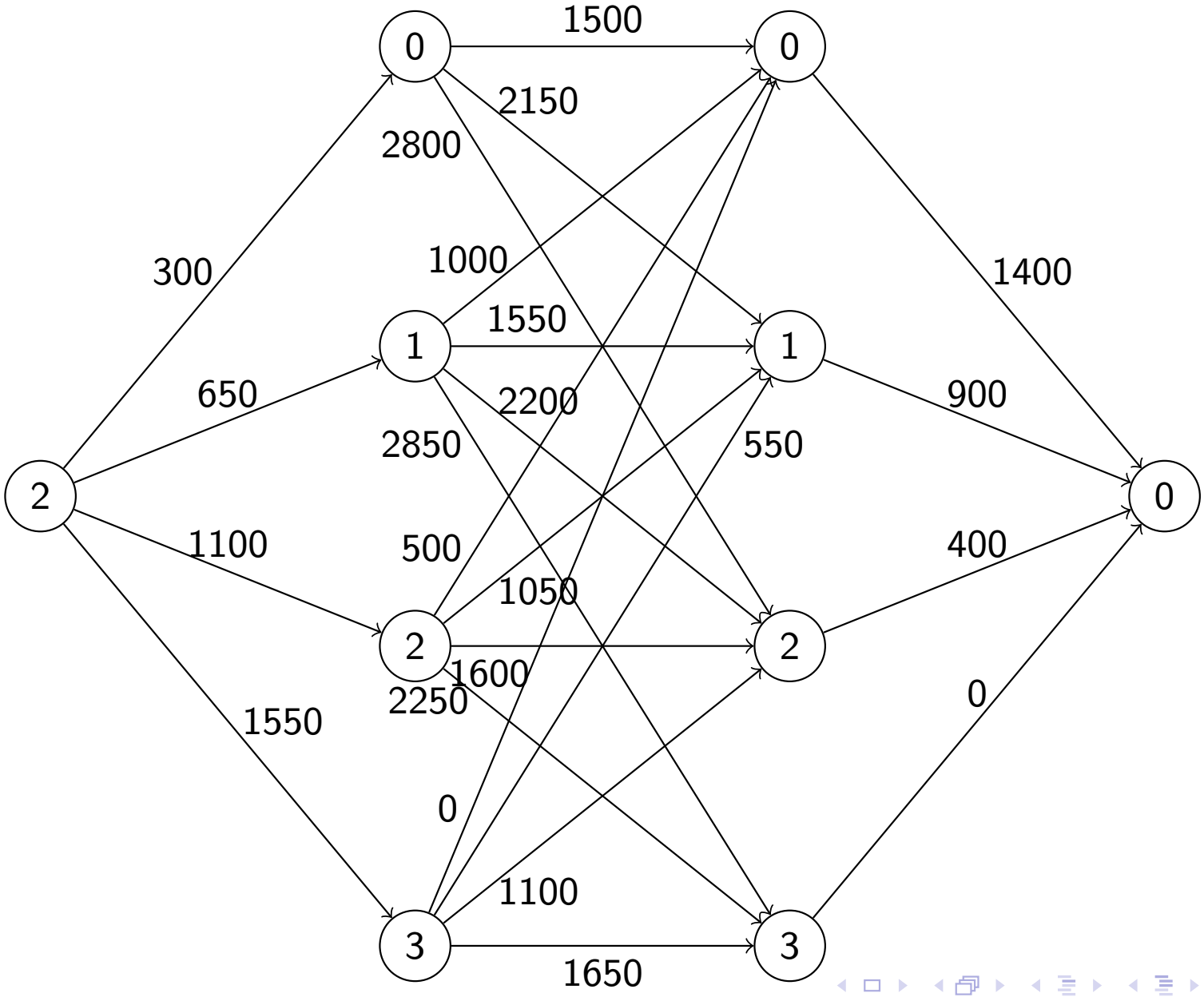
s_1	$x_1 = 1$	$x_1 = 2$	$x_1 = 3$	$x_1 = 4$	$f_1^*(s_1)$	x_1^*
2	3200	3050	3000	2950	2950	4

La politique optimale est donc:

$$x_1^* = 4,\ x_2^* = 0,\ x_3^* = 3$$

donnant lieu à $s_2 = 3$, $s_3 = 0$. Le coût total est $f_1^*(2) = 2950\$$.

Production-inventaire (7)



Exemple: Méthode du chemin critique

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.
Les nœuds 1 et N représentent le début et la fin du projet. On suppose de plus qu'il y a au moins un chemin du nœud 1 à n'importe quel autre nœud.

On veut connaître le temps requis pour terminer le projet (i.e. avoir effectué toutes les tâches, certaines pouvant s'accomplir en parallèle), ainsi que les activités critiques.

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 requis 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:

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

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

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:

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

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

$$\begin{array}{lcl} b_i & = & \text{nombre d'unités de ressource de type } i \text{ disponibles.} \\ u_{ik} & = & \text{nombre d'unités de ressource de type } i \text{ allouées à l'activité } k; \\ x_{ik} & = & \text{nombre d'unités de ressource de type } i \text{ disponible} \\ & & \text{pour les activités } k \text{ à } N; \end{array}$$

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

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.

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 faire 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

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

Pour les noeuds $> x$, il suffit de considérer les objets de types $\leq v(x)$.

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

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

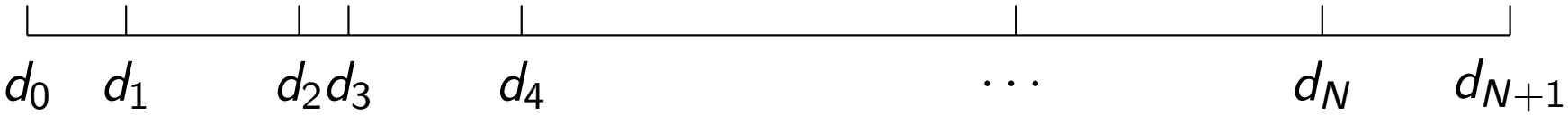
      $D(x+a_j) \leftarrow c_j + D(x)$; $v(x+a_j) \leftarrow j$;

    SI $c_j + D(x) = D(x+a_j)$ ET $x + a_j \leq b$ ET $j < v(x+a_j)$

      ALORS $v(x+a_j) \leftarrow j$;

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).

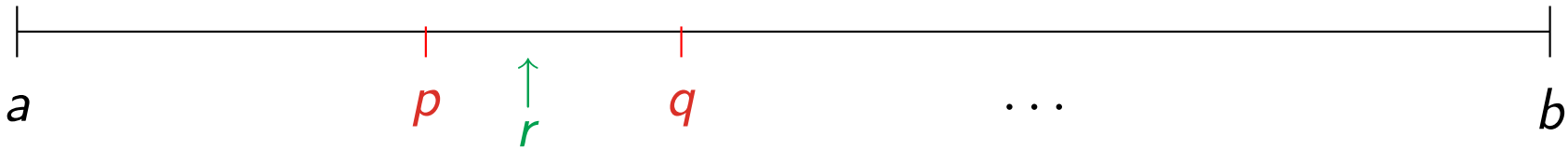
Récurrence:

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

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: 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:

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

Interprétation: on a k soldats pour protéger $[p, b)$. Le premier soldat défendra le segment $[p, u)$. Les $k - 1$ autres défendront $[u, b)$ (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, ...

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'entraînement 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

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

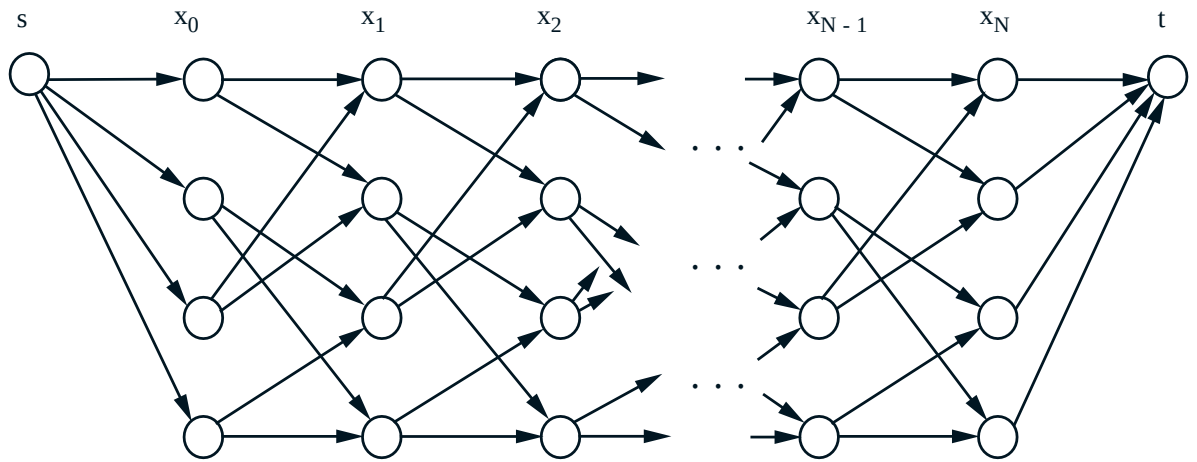
Cela équivaut à minimiser, p.r. à $x_0, \ldots, x_N$,

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

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.