Chapitre 6: Méthodes d'approximation en PD

Fabian Bastin

DIRO, Université de Montréal

IFT-6521 – Hiver 2013

Types de difficultés dans la résolution

Modèle stochastique sur horizon fini, rappel:
À l'étape k, l'état est x_k, on prend une décision $u_k \in U_k(x_k)$, puis on paye un coût $g_k(x_k, u_k, w_k)$, et le prochain état sera $x_{k+1} = f_k(x_k, u_k, w_k)$, où la v.a. w_k suit la loi $P_k(\cdot \mid x_k, u_k)$.
(Dans le cas partiellement observé, l'état est I_k.)

Équations de récurrence:

$$\begin{aligned} J_N(x) &= g_N(x) \quad \text{pour } x \in X_N, \\ J_k(x) &= \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right], 0 \leq k < N, x \in X_k, \\ \mu_k^*(x) &= \arg \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right]. \end{aligned}$$

En pratique, la résolution est souvent trop coûteuse ou impossible, pour diverses raisons.

$$J_k(x) = \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right], x \in X_k$$

Difficultés et quelques idées.

A. Quand l'espace d'états est trop grand (continu, multidimensionel,...), on ne peut pas calculer et mémoriser J_k partout. Malédiction de la dimensionalité: la taille de l'espace d'états croît exponentiellement en fonction du nombre de variables d'états (dimension).
Pire dans le cas partiellement observé.

Principe de solution: remplacer J_{k+1} par une approximation $\tilde{J}_{k+1}$.
Plusieurs types d'approximations possibles.

– choisir une forme paramétrique et optimiser les paramètres;
– combinaison linéaire de fonctions de base;
– prendre la fonction de valeur associée à une solution sous-optimale; etc.

Utiliser $\tilde{J}_{k+1}$ à la place de J_{k+1} à l'étape k.

Discrétisation de l'espace d'états

Choisir un ensemble fini d'états $\bar{S} = \{x^1, \ldots, x^M\} \subset S$.
Calculer (ou approximer) $J_k(x)$ par $\bar{J}_k(x)$, seulement pour $x^m \in \bar{S}$.
Définir une fonction $\tilde{J}_k(x)$ qui interpole ou approxime les points $\{(x^m, \bar{J}_k(x^m)), 1 \leq m \leq M\}$. Par exemple, combinaison convexe:

$$\begin{aligned} \tilde{J}_k(x) &= \sum_{x^m \in \bar{S}} w^m(x) \bar{J}_k(x^m) \qquad \text{si} \\ x &= \sum_{x^m \in \bar{S}} w^m(x) x^m, \quad w^m(x) \geq 0, \sum_m w^m(x) = 1. \end{aligned}$$

Ou encore choisir une classe de fonctions $\{\tilde{J}_k(x, r_k),\ r_k \in \mathbb{R}^d\}$, puis trouver r_k qui optimise l'ajustement aux points d'évaluation $\{(x^m, \bar{J}_k(x^m)),\ x^m \in \bar{S}\}$ (apprentissage des paramètres). Par exemple, par moindres carrés:

$$\min_{r_k} \sum_{x^m \in \bar{S}} \|\tilde{J}_k(x^m, r_k) - \bar{J}_k(x^m)\|^2.$$

Discrétisation de l'espace d'états (suite)

Modèle linéaire: $\tilde{J}_k(x, r_k) = \sum_{i=1}^{d} r_{k,i} \Psi_i(x)$ où $\Psi_1, \ldots, \Psi_d$ sont des fonctions de base choisies à l'avance.

Approximation de la politique

Même chose pour la politique: on doit approximer chaque fonction μ_k^*.

$$\mu_k^*(x) \quad = \quad \arg \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right].$$

On peut ici aussi approximer de la même façon que pour J_k: Calculer $\mu_k^*(x^i)$ seulement pour $x^i \in \bar{S}$, puis interpoler ou approximer.

Une possibilité: restreindre la politique à une classe paramétrisée par un vecteur de paramètres θ de petite dimension, puis optimiser θ (par exemple par des méthodes de descente, en estimant le gradient de la performance par rapport à θ). On peut faire cela sans approximer la fonction de valeur J_k.

Approximation d'espérance

$$J_k(x) = \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right], x \in X_k,$$

Parfois, le calcul de l'espérance pour u fixé est trop complexe et on doit l'approximer. On devra dans certains cas se rabattre sur la simulation Monte Carlo. On estimera l'espérance par simulation pour plusieurs valeurs de u, en utilisant des v.a. communes pour les différentes valeurs de u.

Heuristiques

$$J_k(x) = \min_{u \in U_k(x)} \mathbb{E}_{w_k} \left[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k)) \right], x \in X_k,$$

Parfois, la minimisation par rapport à u est trop difficile, et on devra se contenter d'heuristiques ou d'approximations à ce niveau.

Les quatre méthodes d'approximation considérées sont quatre types d'effets de la malédiction de la dimensionalité.

Dans certains cas, le problème doit être résolu en temps réel (très rapidement), souvent parce que les données arrivent à la dernière minute, ou encore parce que certaines données sont recueillies au fur et à mesure que le système évolue.

Dans ce chapitre, nous examinons différentes techniques pour attaquer ces difficultés.

Commande à équivalent certain (CEC)

L'idée est de remplacer le modèle stochastique par un modèle déterministe, en remplaçant toutes les variables aléatoires par des valeurs "moyennes" ou "typiques".

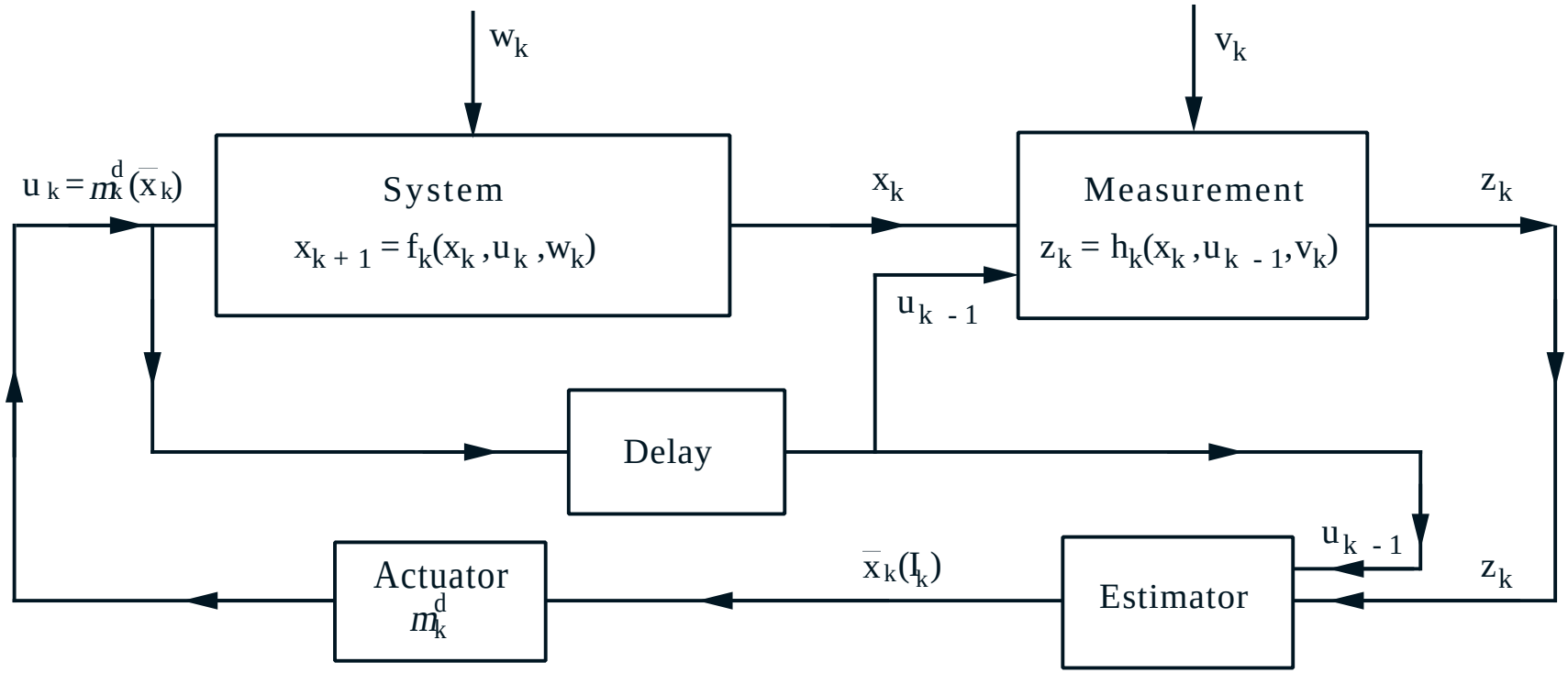
Algorithme (cas partiellement observé). À chaque étape k:

1. Calculer un estimateur $\bar{x}_k(I_k)$ de x_k à partir de l'information I_k.
2. Fixer w_i à $\bar{w}_i(x_i, u_i)$ pour $i \geq k$ et résoudre le problème déterministe:

$$\begin{aligned} \min_{u_k,\dots,u_{N-1}} \quad & g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, u_i, \bar{w}_i(x_i, u_i)) \\ \text{s.l.c.} \quad & x_k = \bar{x}_k(I_k),\ u_i \in U_i,\ x_{i+1} = f_i(x_i, u_i, \bar{w}_i(x_i, u_i)), \quad i \geq k. \end{aligned}$$

3. Utiliser u_k comme décision à l'étape k.

Soit $\{\mu_0^d(x_0), \ldots, \mu_{N-1}^d(x_{N-1})\}$ une politique optimale pour le problème déterministe. À l'étape k, CEC utilise la décision $\bar{\mu}_k(I_k) = \mu_k^d(\bar{x}_k(I_k))$.



Parfois on peut se contenter d'une solution approximative ou heuristique du problème déterministe, dont la valeur est, disons, $H_k(x_k)$.

Amélioration: regarder une étape en avant (one-step lookahead). Au temps k, choisir $u_k \in U_k(x_k)$ qui minimise

$$g_k(x_k, u_k, \bar{w}_k(x_k, u_k)) + H_{k+1}(f_k(x_k, u_k, \bar{w}_k(x_k, u_k))).$$

On peut remplacer seulement une partie des w_k par des valeurs déterministes, et traiter les autres comme stochastiques.

Cas particulier: dans le cas partiellement observé, calculer une estimation $\bar{x}_k(I_k)$ de l'état x_k et l'utiliser comme si c'était le véritable état.

Exemple: Protocole Aloha (primitif) de transmission d'information sur un réseau. Le réseau est commun à plusieurs postes. Le temps est divisé en fenêtres (slots).
Lorsqu'un poste transmet un paquet d'information, il est diffusé sur le réseau et s'il est bien reçu par le destinataire, ce dernier le retransmet et l'expéditeur vérifie si le message est identique.

Si deux ou plusieurs messages sont transmis dans la même "slot", on doit recommencer: collision. Chaque station a une file d'attente de paquets à transmettre. On veut minimiser l'attente moyenne.
Politique optimale: s'il y a en tout x paquets dans les files d'attente, alors chaque station transmet durant le prochain "slot" avec probabilité $\min(1, 1/x)$.

Mais en pratique, les stations ne connaissent pas x_k. Alors chacun calcule une estimation $\bar{x}_k(I_k)$ de x_k, basée sur ce qu'il a pu observer à date (succès de transmission, collisions, périodes libres), puis transmet avec probabilité $\min(1, 1/\bar{x}_k(I_k))$.

Modèle avec paramètres inconnus.

$$x_{k+1} = f_k(x_k, \theta, u_k, w_k)$$

où θ est un vecteur de paramètres inconnus, pour lequel on a une probabilité a priori.

On peut considérer θ comme une variable d'état non observable: $y_k = \theta$ et on a un système avec état élargi $\tilde{x}_k$:

$$\tilde{x}_{k+1} = \begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, w_k) \\ y_k \end{pmatrix} = \tilde{f}_k(\tilde{x}_k, u_k, w_k).$$

Supposons que (1) on peut calculer une politique optimale lorsque θ est connu, et (2) on peut obtenir une estimation $\hat{\theta}_k$ de θ.
Heuristique (CEC partiellement stochastique):

on calcule et utilise la politique optimale pour $\hat{\theta}_k$.

C'est un cas particulier de commande adaptative.

Problème d'identifiabilité

Phase d'identification: calculer une estimation $\hat{\theta}$ de θ;
Phase de commande: appliquer la commande comme si $\hat{\theta} = \theta$.

Danger: il se peut que la commande optimale avec $\theta = \hat{\theta}$ empêche l'identification (l'estimation convergente) de θ.

Exemple. Deux stratégies de vente, ou d'investissement, ou de jeu, etc., disons A et B. Vous essayez chacune 3 fois et vous avez 1 succès avec A et 0 succès avec B. Vous estimez à ce moment que la probabilité de succès est 1/3 pour A et 0 pour B. Par la suite, si vous utilisez toujours la stratégie ayant la plus grande probabilité estimée de succès, vous utiliserez toujours A, et votre probabilité estimée pour B restera à 0 à tout jamais. Mais il se peut bien que la véritable probabilité de succès soit plus grande pour B.

Exemple. Soit le système linéaire unidimensionnel

$$x_{k+1} = ax_k + bu_k + w_k, \qquad k = 0, 1, \ldots, N-1,$$

pour lequel on veut minimiser le coût espéré

$$\mathbb{E}\left[\sum_{k=1}^{N} x_k^2\right].$$

On a un système linéaire à coût quadratique. Si a et b sont connus, la commande optimale est $\mu_k^*(x_k) = -(a/b)x_k$.

Mais si a et b sont inconnus et on applique une commande de la forme $\tilde{\mu}_k^*(x_k) = \gamma x_k$, le système va évoluer selon

$$x_{k+1} = (a + b\gamma)x_k + w_k.$$

On pourra au mieux identifier $a + b\gamma$, mais pas a et b, ni a/b.

Trois versions du système, à distinguer.

Soit $\mu_k^*(x_k, \theta)$ la commande optimale dans l'état x_k, pour θ connu. En pratique, on utilisera plutôt la commande $\hat{\mu}_k(I_k) = \mu_k^*(x_k, \hat{\theta}_k)$ à l'étape k. On distingue:

(a) Le système estimé, que l'on croit (souvent à tort) être le bon, qui évolue selon

$$\mathbb{P}\left[x_{k+1} \in \cdot \mid x_k, \mu_k^*(x_k, \hat{\theta}_k), \hat{\theta}_k\right].$$

(b) Le véritable système avec la commande utilisée, qui évolue selon

$$\mathbb{P}\left[x_{k+1} \in \cdot \mid x_k, \mu_k^*(x_k, \hat{\theta}_k), \theta\right].$$

(c) Le véritable système avec sa commande optimale, qui évolue selon

$$\mathbb{P}\left[x_{k+1} \in \cdot \mid x_k, \mu_k^*(x_k, \theta), \theta\right].$$

Dans certains cas, même si $\hat{\theta}_k$ est très différent de θ, la fausse croyance que l'on a le bon paramètre ne fait que se rencorcer!

Solution: Ajouter des mécanismes qui favorisent l'exploration, pour améliorer l'apprentissage (par ex., ajouter du bruit dans le choix des décisions). Il faut partager les efforts entre l'estimation et la minimisation des coûts à court terme.

Domaine de la commande adaptative: vaste et complexe.

Politiques à anticipation limitée

Anticipation d'une étape ("one-step lookahead policy", 1SL):
À l'étape k, on utilise la commande

$$\bar{\mu}_k(x_k) = \arg \min_{u_k \in U_k(x_k)} \mathbb{E}\left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k)) \right].$$

où $\tilde{J}_N = g_N$ et $\tilde{J}_{k+1}$ est une approximation de J_{k+1}.
On peut aussi restreindre a priori les décisions admissibles au sous-ensemble $\bar{U}_k(x_k) \subset U_k(x_k)$.

Si $\tilde{J}_{k+1}$ est déjà disponible ou rapide à calculer, et si la minimisation est assez facile, on peut implanter cette politique "en-ligne".

Parfois, on doit résoudre un problème d'optimisation pour calculer $\tilde{J}_{k+1}(x_{k+1})$: lien avec la programmation stochastique.

Anticipation de ℓ étapes ("ℓ-step lookahead policy", ℓ-SL): à l'étape $k \leq N - \ell$, on résoud un problème de PD à ℓ étapes, avec fonction de coût terminal $\tilde{J}_{k+\ell}$. Dans le cas stochastique, $\ell > 1$ est souvent trop coûteux.

Bornes sur la performance pour 1SL.

$\tilde{J}_k(x_k) =$ approximation de base;
$\bar{J}_k(x_k) =$ véritable coût espéré avec la politique 1SL;
$\hat{J}_k(x_k) =$ fonction de valeur calculée durant 1SL: $\hat{J}_N = g_N$ et

$$\hat{J}_k(x_k) = \min_{u_k \in \overline{U}_k(x_k)} \mathbb{E}\left[g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}(f_k(x_k, u_k, w_k))\right].$$

Proposition. Si $\hat{J}_k(x_k) \leq \tilde{J}_k(x_k) + \delta_k$ pour tout (k, x_k), alors $\bar{J}_k(x_k) \leq \hat{J}_k(x_k) + \delta_k + \cdots + \delta_{N-1}$ pour tout (k, x_k).

Cette propriété tient souvent avec $\delta_k = 0$.

Cas particulier important: $\tilde{J}_k(x_k)$ est le coût pour une politique heuristique $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$. Cela s'appelle une politique de déroulement ("rollout policy"). Dans ce cas, lorsque $\hat{J}_k(x_k) > \tilde{J}_k(x_k)$ (si $\mu_k \notin \overline{U}_k(x_k)$), on peut se rabattre sur l'heuristique, auquel cas on aura toujours $\hat{J}_k(x_k) \leq \tilde{J}_k(x_k)$.

Proposition. Si $\hat{J}_k(x_k) \leq \tilde{J}_k(x_k) + \delta_k$ pour tout (k, x_k), alors $\bar{J}_k(x_k) \leq \hat{J}_k(x_k) + \delta_k + \cdots + \delta_{N-1}$ pour tout (k, x_k).

Preuve. Par induction sur k pour $\delta_k = \cdots = \delta_{N-1} = 0$. La généralisation est facile.

On a $\bar{J}_N(x) = \hat{J}_N(x) = \tilde{J}_N(x) = g_N(x)$ pour tout x.
En supposant que $\bar{J}_{k+1}(x) \leq \hat{J}_{k+1}(x)$ pour tout x, on obtient

$$\begin{aligned}
\bar{J}_k(x) &= \mathbb{E}\left[g_k(x, \bar{\mu}_k(x), w_k) + \bar{J}_{k+1}(f_k(x, \bar{\mu}_k(x), w_k))\right] \\
&\leq \mathbb{E}\left[g_k(x, \bar{\mu}_k(x), w_k) + \hat{J}_{k+1}(f_k(x, \bar{\mu}_k(x), w_k))\right] \\
&\leq \mathbb{E}\left[g_k(x, \bar{\mu}_k(x), w_k) + \tilde{J}_{k+1}(f_k(x, \bar{\mu}_k(x), w_k))\right] \\
&= \hat{J}_k(x).
\end{aligned}$$

Stratégies d'approximation

Le choix de la méthode d'obtention de $\tilde{J}_k$ est un aspect très important. Il y a plusieurs classes de méthodes.

(a) Remplacer le problème original par un problème plus facile à résoudre et prendre la solution du problème simplifié pour $\tilde{J}_k$.

(b) Approximation paramétrique de J_k: on approxime chaque fonction J_k par une forme paramétrique dont les paramètres sont appris, ou estimés.

(c) Algorithme rollout: $\tilde{J}_k(x)$ est la fonction de valeur associée à une politique sous-optimale donnée, ou encore une estimation de cette fonction (calculée par simulation, par exemple). Heuristique combinée: essayer plusieurs heuristiques et prendre la meilleure solution obtenue.

Simplifier le problème original

Par ex., remplacer des variables aléatoires par des valeurs déterministes “typiques”, remplacer des espérances par des approx., simplifier des contraintes complexes ou la dynamique du système, etc.

Exemple: problème de tournée de véhicules

On a m véhicules pour parcourir un graphe. Chaque noeud donne un revenu (connu) la première fois qu'un véhicule y passe (aucun revenu supplémentaire n'est collecté si un autre véhicule passe par ce noeud ultérieurement). On veut maximiser le total recueilli, en respectant des contraintes sur l'instant et le lieu de départ et d'arrivée de chaque véhicule, etc. En particulier, nous supposons que chaque véhicule part d'un noeud donné, et doit revenir à un noeud donné (possiblement différent) en au plus un nombre donné de mouvements d'arcs.

C'est un problème combinatoire difficile, qui peut, en principe, être traité avec la programmation dynamique.

Exemple: problème de tournée de véhicules

Etat: ensemble des positions des véhicules ainsi que la liste des noeuds déjà visités (et donc ne pouvant plus donner de revenu). Le nombre d'états croît exponentiellement avec le nombre de noeuds et de véhicule.

Pour un seul véhicule, le problème est beaucoup plus simple (sans pour autant être facile), et peut être résolut en temps raisonnable, de manière exacte ou approchée.

Exemple: problème de tournée de véhicules

Simplification (sous-optimale): on peut résoudre une suite de problèmes à un seul véhicule, comme suit. À l'étape k et dans l'état x_k (positions des véhicules et ensemble des noeuds non visités), pour chaque véhicule, on considère chaque déplacement possible. Pour l'état résultant x_{k+1} (dans chaque cas) on approxime la fonction de valeur $J_{k+1}(x_{k+1})$ par la valeur $\tilde{J}_{k+1}(x_{k+1})$ obtenue en optimisant pour un véhicule à la fois (les autres véhicules ne bougent pas), dans un ordre préalablement défini.

On peut recommencer en changeant l'ordre des véhicules, et en gardant l'ensemble de chemins (associé à un ordre donnée) délivrant le gain maximal.

Exemple: gestion du revenu.

Un hôtelier fixe le prix de ses chambres (pour une nuit donnée) en fonction du nombre encore disponibles, de la loi de prob. du nombre de demandes à venir pour cette nuit-là, et des prob. d'acception des clients en fonction du prix. Il veut maximiser son revenu espéré.

Supposons qu'un client choisi au hasard acceptera un prix r avec probabilité $p(r)$, une fonction décroissante de r. À un instant donné, soit x le nombre de chambres disponibles et soit y (une variable aléatoire) le nombre de demandes à venir.

Considérons une version simplifiée du problème où l'hôtelier connaît toujours y. Dans ce cas, l'état est (x, y), et si $J(x, y)$ dénote le revenu espéré optimal dans cet état, on a la récurrence:

$$J(x, y) = \max_{r \geq 0} [p(r)(r + J(x - 1, y - 1)) + (1 - p(r))J(x, y - 1)]$$

avec conditions initiales $J(x, 0) = J(0, y) = 0$ pour tout x, y.
Supposons que l'on sait résoudre ce problème facilement, par PD.

En réalité, on ne connait pas y, mais on peut utiliser une estimation $\bar{y}$ (arrondie en entier) à la place de y dans une politique 1SL.

Approximation par scenarios.

On considère M réalisations différentes de $w_{k+1}, \ldots, w_{N-1}$ (au lieu d'une seule comme dans CEC), qui correspondent à M scénarios.

Pour chaque scénario, on calcule $C_m(x_{k+1})$, une approximation du coût à partir de l'étape $k+1$, puis on approxime $J_{k+1}(x_k)$ par

$$\tilde{J}_{k+1}(x_{k+1}) = r_0 + \sum_{m=1}^{M} r_m C_m(x_{k+1}).$$

Les paramètres $r_0, \ldots, r_M$ constituent une pondération. On peut les interpréter en gros comme les probabilités des différents scénarios.

Approximation paramétrique de J_k

On peut choisir une classe de fonctions $\{\tilde{J}_k(x, r),\ r \in \mathbb{R}^d\}$, puis trouver une fonction dans cette classe (i.e., un vecteur de paramètres $r_k = (r_{k,1}, \ldots, r_{k,d})$) qui est le plus proche possible de J_k.

Le modèle d'approximation peut être linéaire en r (plus facile à estimer) ou non linéaire (plus riche, plus flexible).

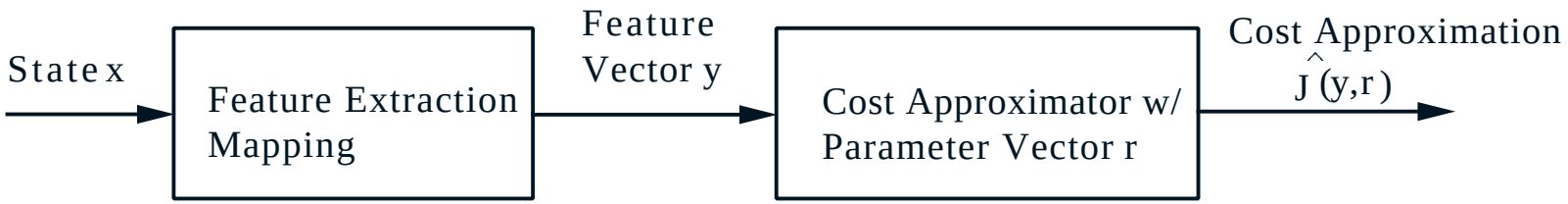
Le choix de la classe de fonctions est très important et dépend du problème.
Optimisation du vecteur r_k: c'est l'entraînement du modèle.
Le succès dépend beaucoup de ces deux aspects.

Extraction d'un vecteur d'attributs ("features") $y = y(x)$, suivi d'une approximation par une fonction (paramétrisée) de y.

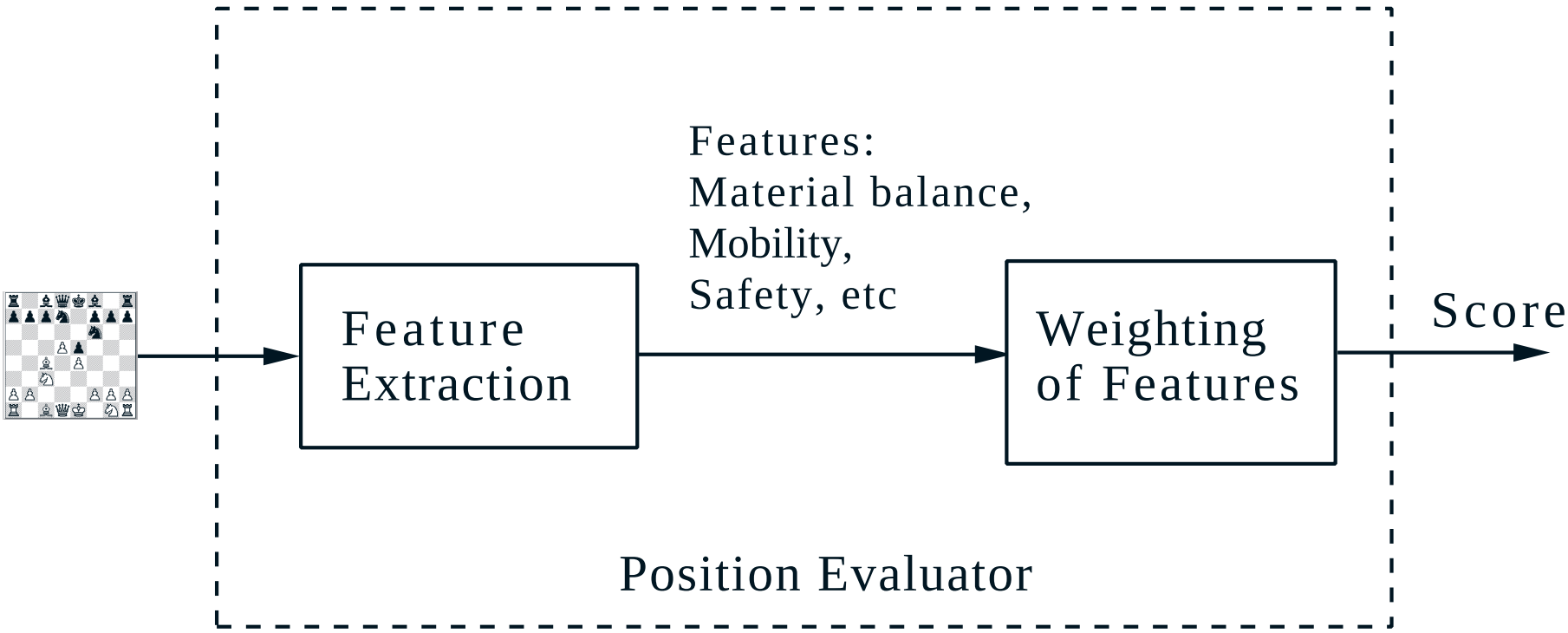
Idée: y peut contenir moins d'information que x, mais une information pertinente et concentrée dans un petit nombre de variables.

Idéalement, y pourra encoder une grande partie de la non-linéarité; ses composantes seront des fonctions non-linéaires de x, bien choisies, de sorte que l'on pourra bien approximer J_k par une fonction linéaire (ou simple) de y et r:

$$\tilde{J}_k(x, r) = \hat{J}(y(x), r) = \sum_{i=1}^{d} r_i y_i(x).$$



Exemple: logiciel joueur d'échecs.



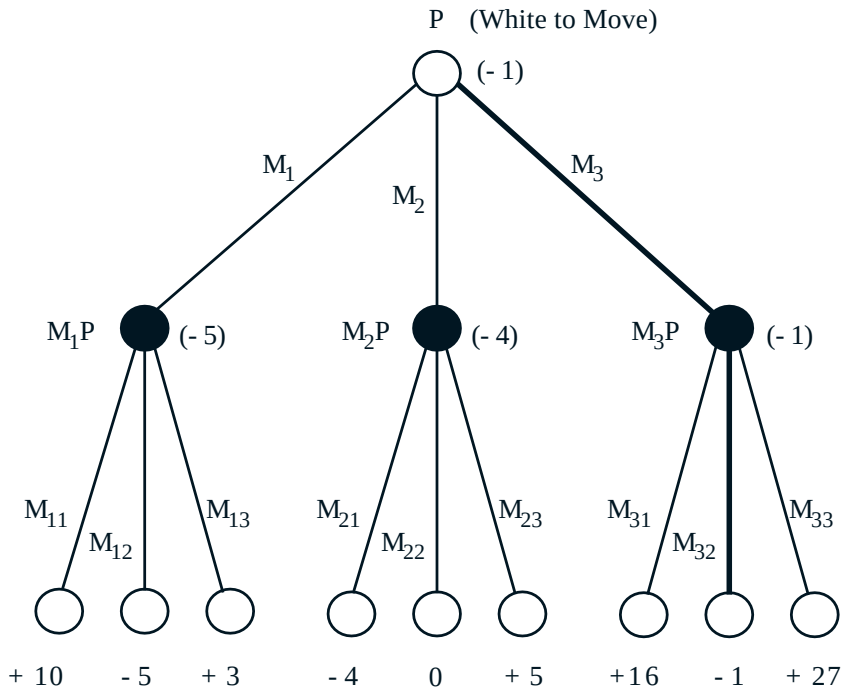
On définit une fonction d'évaluation (score) de chaque position du jeu en termes d'attributs ("features"), fonctions des pièces restantes, leurs positions, etc. Le score est souvent une fonction linéaire (pondération) des attributs.

Les poids sont appris par entraînement (essai-erreur).

Recherche en profondeur dans le graphe; profondeur variable;

On commence par trouver rapidement un coup raisonnable à jouer, en explorant en profondeur l'arbre des possibilités.

Considérons par exemple une stratégie à deux coups d'avance (en comptant le coup de l'adversaire). Chaque feuille de l'arbre de la figure ci-dessous donne une valeur un score à l'échiquier. Nous allons essayer de "remonter" une valeur pour obtenir la stratégie optimale sous cette approximation, i.e. le meilleur score, et la séquence de coups associées.



Pour "remonter le score", il suffit de remonter le minimum des scores lorsque l'on passe d'une ligne associée au joueur blanc (en supposant que nous sommes ce joueur), vu que le joueur adverse cherche à nous faire perdre, et donc devrait choisir le coup qui diminue au plus le score. A l'inverse, en passant d'une ligne noire à une ligne blanche, nous remontons le maximum, comme nous cherchons le meilleur coup.

Il est toutefois prohibitif de parcourir l'arbre de manière exhaustive, aussi cherchons-nous à l'élaguer, i.e. a ne pas parcourir les branches qui ne contribuent pas à la solution.

Pour se faire, dans chaque situation, on conserve (et met à jour) notre meilleur coup à jouer et le meilleur coup de l'adversaire. Soient α et β les valeurs pour nous de ces coups.

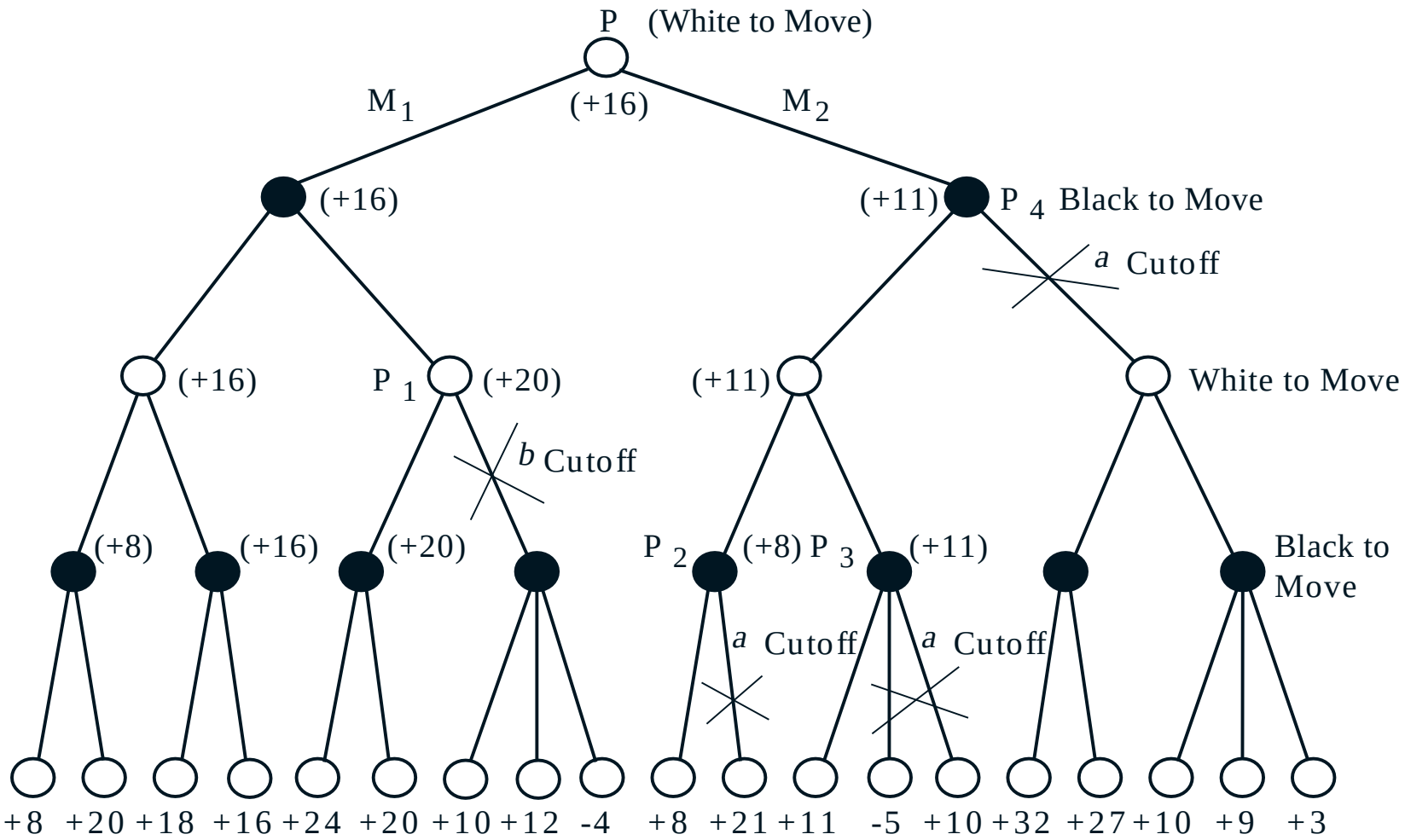
Technique α-β

Lorsqu'on explore l'arbre, dès qu'on se rend compte qu'un coup donné ne peut pas faire mieux que α (si l'adversaire joue bien), on coupe la branche correspondante de l'arbre.

De même, dès qu'on se rend compte qu'un coup de l'adversaire ne peut pas faire pire (pour nous) que β (si on joue bien), on coupe la branche correspondante de l'arbre.

Technique alpha-beta: dès que l'on sait qu'une branche ne peut contenir la suite optimale de coups, on l'élimine.

Ces valeurs sont obtenues en commançant un parcours en profondeur, pour en "remontant" les branches pour explorer les bifurcations non-encore visitées. Dans l'illustration suivante, le principe est d'explorer en profondeur, en allant toujours le plus à gauche possible dans l'arbre.



Extensions: anticipation à plusieurs étapes, en nombre variable, en profondeur.

Approximation par déroulement ("rollout")

$\tilde{J}_k(x) = H_k(x)$ est la fonction de valeur pour une politique sous-optimale donnée $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, appelée politique de base.

Par conséquent, la politique par déroulement est une politique de type 1SL, ou l'approximation du coût est donnée par le coût de la politique de base. L'extension à des politiques ℓ-SL est facile.

Si pas de formule pour évaluer cette valeur, son calcul peut être coûteux. Parfois, on peut utiliser une heurisitque (nous écrirons alors $\hat{J}$ au lieu de ($\tilde{J}$), ou évaluer (estimer) par simulation. Souvent plus facile dans le cas déterministe.

Amélioration du coût avec l'algorithme rollout

$H_k(x_k)$ = coût espéré pour l'heuristique de base μ_k;
$\bar{J}_k(x_k)$ = coût espéré pour la politique rollout $\bar{\mu}_k$.

Proposition. $\bar{J}_k(x) \leq H_k(x)$ pour tout (k, x).

Preuve. C'est un cas particulier de la proposition précédente sur la performance pour 1SL. Preuve par induction sur k. On a $\bar{J}_N(x) = H_N(x) = g_N(x)$ pour tout x.
En supposant que $\bar{J}_{k+1}(x) \leq H_{k+1}(x)$ pour tout x, on obtient

$$\begin{aligned}
\bar{J}_k(x) &= \mathbb{E}\left[g_k(x, \bar{\mu}_k(x), w_k) + \bar{J}_{k+1}(f_k(x, \bar{\mu}_k(x), w_k))\right] \\
&\leq \mathbb{E}\left[g_k(x, \bar{\mu}_k(x), w_k) + H_{k+1}(f_k(x, \bar{\mu}_k(x), w_k))\right] \\
&\leq \mathbb{E}\left[g_k(x, \mu_k(x), w_k) + H_{k+1}(f_k(x, \mu_k(x), w_k))\right] = H_k(x).
\end{aligned}$$

Autrement dit, la politique rollout améliore la politique de base.
Empiriquement, cette amélioration est souvent très importante.

Exemple: ordonner les questions d'un quiz.

Rappel: N questions à ordonner. On répondra correctement à la question i avec probabilité p_i, et si on le fait on gagne R_i. Dès que l'on échoue à une question, c'est terminé. On veut maximiser notre gain total espéré.

Politique optimale: ordonner les questions par ordre décroissant de leur valeur de $p_i R_i/(1-p_i)$.

Mais si on ajoute des contraintes, par ex. nombre max. de questions, contraintes d'ordonnancement, contraintes de temps, etc., alors cette politique n'est plus nécessairement optimale.

Par contre, on peut l'utiliser pour définir une politique de base pour un algorithme "rollout". Les valeurs de $H_k(x)$ et $\bar{J}_k(x)$ correspondantes sont faciles à calculer.

Problèmes discrets déterministes

Un problème de décision séquentiel déterministe ayant un nombre fini d'étapes et un nombre fini de décisions admissibles peut se représenter par un arbre. Les feuilles représentent toutes les solutions possibles. Chacune a un coût. On cherche un chemin allant de la racine (origine) à une feuille de moindre coût.

Algorithme PD ordinaire: on part de chaque feuille et on recule.

On peut ajouter un noeud t et un arc de coût nul de chaque feuille vers t, puis reformuler comme un problème de plus court chemin de s à t, que l'on peut résoudre en principe par des méthodes d'étiquetage.

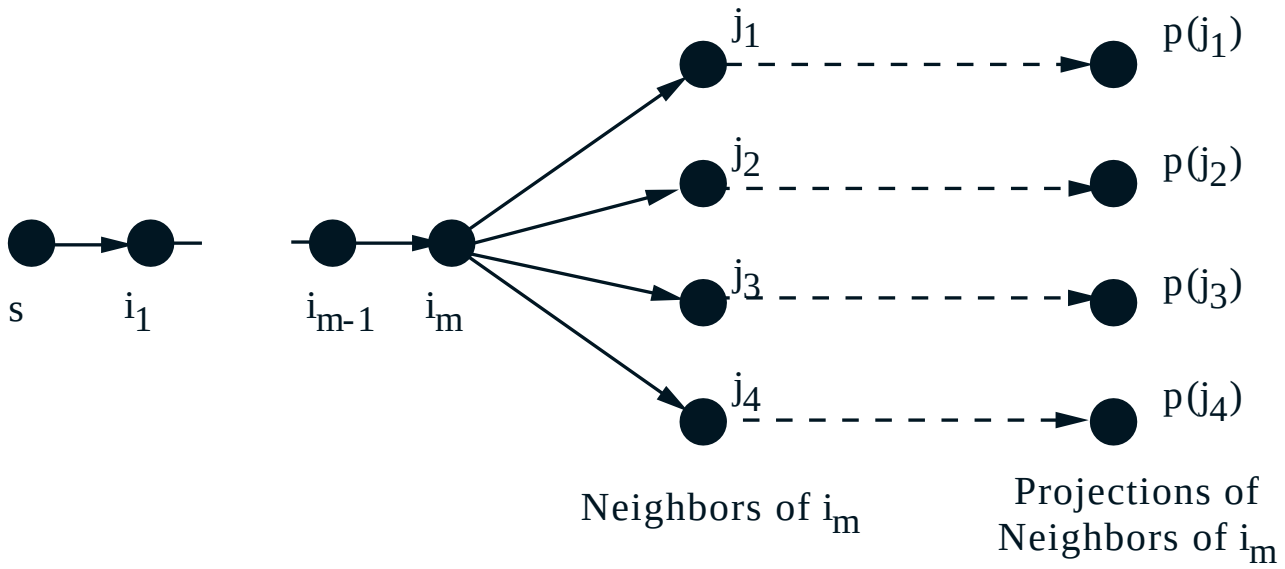
Mais parfois, cela n'est pas possible, car le graphe est trop gigantesque.

Possibilité: se rabattre sur un algorithme rollout.

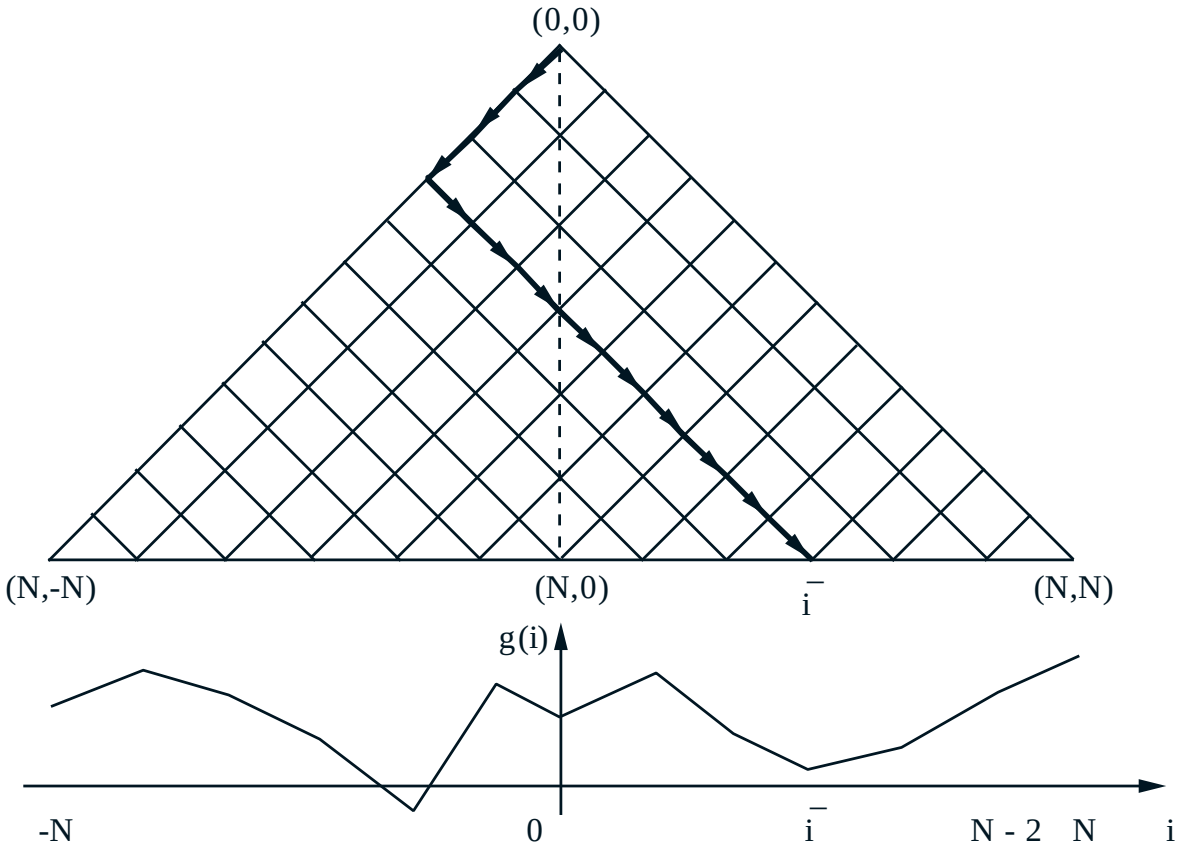
Problème générique: on a un graphe orienté ayant une origine s, un ensemble T de noeuds terminaux (souvent les feuilles dans le cas d'un arbre), un coût $g(i)$ pour chaque noeud terminal i, et on cherche un chemin allant de s à un noeud terminal de moindre coût.

Heuristique de base: pour chaque noeud $i \notin T$, une heuristique nous donne un chemin unique $(i, i_1, \ldots, i_m, \bar{i})$, où $\bar{i} \in T$ s'appelle la projection de i. Le coût correspondant est $H(i) = g(\bar{i})$.

Algorithme rollout: Partant de s, toujours choisir comme successeur le noeud ayant la plus petite projection.



Exemple: une marche unidimensionelle



On part de 0. À chaque étape, on se déplace de 1 à gauche ou à droite. Si i est la destination finale après N étapes, on paye $g(i)$.

Exemple: une marche unidimensionelle (suite)

Heuristique de base A: toujours aller à droite. L'algorithme rollout trouve alors un minimum local.
Heuristique de base B: comparer aller toujours à droite vs aller toujours à gauche, et choisir le meilleur des deux. Dans ce cas, l'algorithme rollout trouve alors un minimum global!

Consistance séquentielle. Une heuristique de base est séquentiellement consistante si chaque fois qu'elle retourne le chemin $(i, i_1, \ldots, i_m, \bar{i})$ en partant du noeud i, elle retournera $(i_1, \ldots, i_m, \bar{i})$ en partant du noeud i_1.

Exemple: un algorithme vorace, qui utilise une estimation $F(i)$ du coût optimal en partant de i. L'algorithme construit son chemin comme suit: étant donné une portion de chemin $(i, i_1, \ldots, i_m)$ où $i_m \notin T$, l'algorithme ajoute l'arc (i_m, i_{m+1}) qui minimise $F(i_{m+1})$. Un tel algorithme est séquentiellement consistant.

Proposition. Si l'heuristique de base est séq. consistante, alors l'algorithme rollout se terminera en temps fini, son coût ne dépassera jamais celui de l'heuristique de base, et pour tout chemin $(i, i_1, \ldots, i_m)$ construit par l'algorithme, on aura

$$H(s) \geq H(i_1) \geq \cdots \geq H(i_{m-1}) \geq H(i_m).$$

Approximation des Q-facteurs (cas stochastique)

Le Q-facteur de (x_k, u_k) à l'étape k est

$$Q_k(x_k, u_k) = \mathbb{E}\left[g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k))\right]$$

La décision optimale est

$$\mu_k^*(x_k) = \arg \min_{u_k \in U_k(x_k)} Q_k(x_k, u_k).$$

L'idée du Q-learning est d'apprendre ou estimer les fonctions Q_k. Si les approx. sont $\tilde{Q}_k$, la commande optimale est approximée par

$$\bar{\mu}_k(x_k) = \arg \min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k).$$

On peut par exemple estimer les $Q_k(x_k, u_k)$ par simulation Monte-Carlo, en utilisant les mêmes nombres aléatoires uniformes (v.a. communes) pour tous les u_k.

L'important est de bien estimer les différences $Q_k(x_k, u_k) - Q_k(x_k, u_k')$ pour $u_k \neq u_k'$. Si on se trompe partout par la même constante, cela n'introduit aucune erreur dans le choix des décisions!

Une autre façon d'approximer les Q-facteurs est simplement de remplacer J_{k+1} par une approximation H_{k+1}. On peut l'obtenir par une variante de l'une ou l'autre des techniques vues précédemment (approximation déterministe, scénarios, valeur associée à une heuristique fixée, approximation par moindre carrés, etc.).

Dans le cas d'une approximation déterministe, on fixe $w_{k+1}, \ldots, w_{N-1}$ à $\bar{w}_{k+1}, \ldots, \bar{w}_{N-1}$ (mais pas w_k).

Horizon fuyant ("rolling horizon")

C'est un cas particulier de ℓ-SL, avec $H_k(x) = 0$ pour tout x.
Plus général: $H_k(x) = g(x)$ pour une fonction de coût terminal g.

L'idée est toujours de diminuer la quantité de calculs.

Un horizon plus long donne-t-il toujours une meilleure performance?
La réponse est non!

Exemple: Seulement deux décisions possibles à la première étape, et une seule décision par la suite. Le coût sur le chemin du bas oscille en fonction de ℓ.