Apprentissage par renforcement
Notes de cours

philippe.preux@univ-lille3.fr
GRAPPA

Version du 26 janvier 2008

Résumé

Ces quelques pages résument mon cours sur les problèmes de décision de Markov et l'apprentissage par renforcement.

Avertissement (14 juin 2007) : une mise à jour de ces notes va être réalisée dans les semaines et mois à venir. On y traitera notamment des méthodes de recherche directe de politique, des architectures acteur-critique, du problème des grands espaces d'états et l'utilisation d'approximateurs de fonctions.

1 Séance 1 : introduction

On va :

1. définir précisément le problème que l'on veut résoudre, le « problème de contrôle optimal » ;
2. démontrer des propriétés importantes liées à ce problème et à sa solution.

Ce faisant, on essaie de donner de l'intuition concernant ce problème afin de comprendre à quoi sert ce que l'on essaie de faire. Disons-le tout de suite, le problème que l'on va étudier est extrêmement général, les applications en sont très variées et l'utilisation de ce qui suit est grande.

1.1 Définition du problème et de l'objectif du cours

On définit dans cette section des notions essentielles pour l'ensemble du cours et, ceci étant posé, on termine par préciser l'objectif du cours.

On considère un temps s'écoulant de manière discrète. Un instant est noté $t \in \mathbb{N}$.

La situation : un agent interagit avec son environnement. À chaque instant t, il perçoit un état s_t et émet une action a_t. Suite à cela, il reçoit une conséquence de son action que l'on nomme un retour r_t, et perçoit un nouvel état s_{t+1}.

Cette situation décrit un problème très général. Par exemple :

- un agent logiciel qui explore la Toile afin de trouver des informations pertinentes sur un sujet donné ;
- un agent logiciel qui joue aux échecs ;
- un robot qui ramasse les boîtes de boisson vides dans des bureaux ;
- un robot qui peint des carrosseries de voiture ;
- un robot qui explore une planète lointaine.

Définition 1 *On définit un problème de décision de Markov (PDM) par un quadruplet :* $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ *où :*

– *$\mathcal{S}$ est un ensemble d'états fini ;*
– *$\mathcal{A}$ est un ensemble d'actions fini. On note $\mathcal{A}(s)$ l'ensemble des actions possibles dans l'état s. $\mathcal{A} = \cup_{s \in \mathcal{S}} \mathcal{A}(s)$;*
– *$\mathcal{P}$ la fonction de transition décrivant la dynamique de l'environnement :*

$$\begin{aligned} \mathcal{P} \quad : \quad & \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1] \\ & (s, a, s') \mapsto \mathcal{P}(s, a, s') = Pr[s_{t+1} = s' | s_t = s, a_t = a] \end{aligned}$$

– *$\mathcal{R}$ la fonction de retour :*

$$\begin{aligned} \mathcal{R} \quad : \quad & \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R} \\ & (s, a, s') \mapsto \mathcal{R}(s, a, s') = E[r_t | s_{t+1} = s', s_t = s, a_t = a] \end{aligned}$$

La notation $E[.]$ représente l'espérance mathématique (la valeur moyenne) de ..

Il peut exister un ensemble d'états finaux $\mathcal{F} \subset \mathcal{S}$: quand l'agent atteint l'un de ces états, sa tâche est terminée. On parle alors de tâche épisodique, ou à horizon fini.

On suppose que ces 4 données sont fixes au cours du temps (sinon, on dirait que le problème est « non stationnaire »).

Remarquons que l'environnement est implicitement considéré comme stochastique (ou non déterministe). Le cas déterministe est un cas particulier inclus dans tout ce que l'on va dire dans la suite.

On suppose (on l'a fait implicitement dans les définitions ci-dessus de $\mathcal{P}$ et $\mathcal{R}$) que l'état courant et le retour courant ne dépendent que de l'état précédent et de l'action qui vient d'être émise. Ceci est une propriété fondamentale dans ce que nous allons faire. C'est la « propriété de Markov » (on dit aussi que l'environnement est markovien[1]).

Définition 2 *On définit une politique (ou stratégie) π comme une application :*

$$\begin{aligned} \pi \quad : \quad & \mathcal{S} \times \mathcal{A} \to [0,1] \\ & (s, a) \mapsto \pi(s, a) = Pr[a_t = a | s_t = s] \end{aligned}$$

Par la suite, on utilisera également la notion de politique déterministe qui associe une action à un état (et non une probabilité d'émission des actions pour chaque état). On utilisera alors la définition suivante :

$$\begin{aligned} \pi \quad : \quad & \mathcal{S} \to \mathcal{A} \\ & s \mapsto \pi(s) = a \end{aligned}$$

L'utilisation de l'une ou l'autre n'est pas ambiguë : il suffit de compter le nombre d'arguments que l'on associe à π.

On définit le retour R reçu par l'agent à partir de l'instant t par[2] :

[1] Pour être tout à fait précis, ce qui est indiqué dans le texte, « l'état suivant et le retour obtenu ne dépendent que de l'état courante et de l'action qui vient d'être émise », caractérise un processus de Markov d'ordre 1. On peut généraliser à des processus d'ordre k dans lesquels l'état suivant et le retour obtenu dépendent des k états précédents et des k dernières actions émises. On sait que tout processus d'ordre k peut être transformé en un processus d'ordre 1, en augmentant le nombre d'états. Dans le texte ici, on suppose tout le temps que l'on a affaire à un processus de Markov d'ordre 1, à moins que cela ne soit explicitement indiqué.

[2] d'autres définitions peuvent être proposées.

$$R_t = \sum_{k=0} \gamma^k r_{t+k}$$

où $\gamma \in [0, 1]$ est le facteur de dépréciation. Il permet de régler l'importance que l'on donne aux retours futurs par rapport aux retours immédiats : γ près de 1 : les retours futurs sont pris en compte ; γ près de 0 : les retours futurs sont négligés.

Objectif du cours : trouver la politique que doit suivre l'agent pour maximiser R_0[3].
Cette politique est qualifiée d'optimale. On la note π^*.

Propriété 1 *π^* existe pour tout PDM.*[4]

On cherche donc à résoudre un problème de contrôle optimal. C'est un problème très général. L'apprentissage par renforcement se compose d'un ensemble de concepts et d'algorithmes qui permettent de résoudre ce problème. Il existe d'autres méthodes. La force de l'apprentissage par renforcement par rapport à d'autres méthodes est que la seule connaissance de $\mathcal{S}$ et $\mathcal{A}$ suffisent à résoudre ce problème : la connaissance de la dynamique de l'environnement ($\mathcal{P}$ et $\mathcal{R}$) n'est pas nécessaire.

Remarquons que l'apprentissage par renforcement ne se définit pas par une certaine classe d'algorithmes mais par le problème qu'il cherche à résoudre, celui du contrôle optimal.

Vue d'ensemble du cours : dans la suite du cours, on va étudier la résolution de ce problème. Tout d'abord (séances 1 et 2), on va le résoudre dans le cas où le PDM est complètement spécifié (méthode de « programmation dynamique »). Ensuite (séance 3), on en étudiera la résolution dans le cas où on ne connaît que $\mathcal{S}$ et $\mathcal{A}$. Ensuite (séance 4), on abordera le cas où l'espace d'états ou d'actions, ou le temps, sont continus (prennent une infinité de valeurs). Deux approches seront discutées : discrétisation pour se remplacer dans le cas fini ; résolution directe dans le cas continu. On n'abordera également le cas où l'état perçu n'est pas tout à fait le reflet de l'état réel de l'environnement et le cas où l'environnement est non stationnaire.

1.2 Définitions

Définition 3 *On définit la valeur d'un état s pour une politique fixée π par :*

$$V^\pi(s) = E[R_t | s_t = s]$$

Remarque : cette fonction va jouer un rôle fondamental dans la suite.

Définition 4 *On définit la valeur d'une paire état-action (ou sa qualité) (s, a) pour une politique fixée π par :*

$$Q^\pi(s, a) = E[R_t | s_t = s, a_t = a]$$

Il y a naturellement une forte relation entre ces deux fonctions :

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^\pi(s')]$$

[3] symétriquement, on peut aussi s'intéresser à une politique qui minimise cette fonction en interprétant le retour comme un coût.

[4] On ne donne que très peu de preuves dans ces notes. La plupart des démonstrations sont relativement ardues, voire assez difficiles. Le lecteur intéressé consultera avec avidité l'excellent livre de Bertsekas and Tsitsiklis [1996].

1.3 L'équation de Bellman

On va montrer que la fonction $V^\pi(s)$ peut se définir en fonction de la valeur de tous les états de $\mathcal{S}$:

$$\begin{aligned} V^\pi(s) &= E[R_t|s_t = s] \\ &= E[\sum_{k=0} \gamma^k r_{t+k}|s_t = s] \\ &= E[r_t + \sum_{k=1} \gamma^k r_{t+k}|s_t = s] \\ &= E[r_t + \gamma \sum_{k=1} \gamma^{k-1} r_{t+k}|s_t = s] \\ &= E[r_t] + \gamma \underbrace{E[\sum_{k=1} \gamma^{k-1} r_{t+k}|s_t = s]}_{E[V^\pi(s_{t+1})]} \end{aligned}$$

Dans cette dernière ligne, le calcul de l'espérance doit tenir compte de toutes les situations possibles. Pour le terme $E[r_t]$, $\mathcal{R}(s, a, s')$ est la moyenne des retours pour s, a et s' fixés. Hors, seul s est fixé. Pour un s fixé, $\pi(s, a)$ donne la probabilité d'émettre l'action a dans l'état s ; pour un couple (s, a), $\mathcal{P}(s, a, s')$ donne la probabilité d'atteindre chacun des états $s' \in \mathcal{S}$. Aussi, on a :

$$E[r_t] = \sum_{a \in \mathcal{A}(s)} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')\mathcal{R}(s, a, s')$$

Par le même raisonnement, on peut calculer le deuxième terme. On en déduit la relation suivante :

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^\pi(s')] \quad (1)$$

Cette équation se nomme l'*équation de Bellman*. Elle définit la fonction valeur de la politique π pour chaque état comme une fonction linéaire de la valeur des états. Si on écrit cette équation pour l'ensemble des états de $\mathcal{S}$, on obtient un système de $|\mathcal{S}|$ équations linéaires. (Rappelons bien que seul V^π est inconnu dans cette équation.)

1.4 L'équation d'optimalité de Bellman

On peut exprimer la fonction valeur de la politique optimale ; on la note V^*. Par définition, on a :

$$V^*(s) = \max_\pi V^\pi(s) = \max_{a \in \mathcal{A}(s)} Q^\pi(s, a)$$

Donc :

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^*(s')] \quad (2)$$

Ce qui constitue l'*équation d'optimalité de Bellman*.

Propriété 2 *Pour un PDM, il peut exister plusieurs politiques optimales. Toutes sont associées à la même fonction valeur V^*.*

1.5 Exemples ; sujets de réflexion

Mettre sous forme d'un PDM les problèmes suivants :
- sortir d'un labyrinthe (on voit l'ensemble du labyrinthe : on est au-dessus) ;
- jouer au tétris ;
- jouer aux échecs, ou aux dames, ou à othello, ...
- jouer au 421 ;
- jouer au Black-Jack (21) ;
- jouer au tic-tac-toe ;
- problème du *cart-pole* : un wagonnet sur lequel est posé en équilibre un piquet. Le wagonnet peut se déplacer entre deux limites. L'objectif est de parcourir la distance maximale sans que le piquet ne tombe ;
- problème de la voiture dans la montagne : un mobile est coincé dans une vallée entre deux montagnes. Elle doit atteindre l'un des deux sommets (fixés à l'avance), s'y arrêter, en minimisant sa dépense énergétique. (Elle peut prendre de l'élan en montant d'un côté et en se laissant rouler pour monter de l'autre côté.)

Ces problèmes sont-ils déterministes ou pas ?

Réfléchir à la mise sous forme d'un PDM des problèmes suivants (ce n'est pas toujours possible) :
- jouer au démineur ;
- sortir d'un labyrinthe en étant plongé dedans : on voit les murs autour de soi et non pas le labyrinthe selon une vue de dessus ;
- jouer au Pacman ;
- jouer au Tarot (ou à la belote, au bridge, ...).

2 Séance 2 : programmation dynamique

On va :

1. apprendre à calculer V^π pour une politique π et un PDM donnés ;
2. à partir de là, apprendre comment trouver une politique optimale π^*.

2.1 Algorithme d'évaluation de la politique

On veut calculer la fonction valeur associée à une politique π fixée. Pour cela, on va appliquer directement l'équation de Bellman (1). Cette équation se transcrit en une équation définissant une suite récurrente $\{V_i^\pi\}$ qui converge vers V^π :

$$V_{i+1}^\pi(s) \quad = \quad \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_i^\pi(s')] \tag{3}$$

Plus précisément, l'évaluation de la politique est donnée dans l'algorithme 1. À l'issue de cet algorithme V_i^π contient une estimation de V^π à ϵ près. Cet algorithme fonctionne bien grâce à la propriété suivante :

Propriété 3 *Soit $\{V_i^\pi\}$ la suite des estimations de V^π engendrées par l'algorithme d'évaluation de la politique. On a :*

$$\exists k \in]0,1[, ||V_i^\pi - V^\pi|| \leq ||V_{i-1}^\pi - V^\pi||$$

(et donc, on a aussi :

$$||V_{i+1}^{\pi} - V_i^{\pi}|| \leq ||V_i^{\pi} - V_{i-1}^{\pi}||$$

)

(Autrement dit, à chaque itération, l'algorithme d'évaluation de la politique s'approche de la valeur recherchée.)

L'algorithme est arrêté quand la précision obtenue est inférieure à un seuil fixé car la convergence vers V^{π} est asymptotique, *i.e.* il faut une infinité d'itérations pour obtenir la vraie valeur de V^{π}.

Algorithme 1 L'algorithme d'évaluation de la politique (stochastique ici).

Nécessite: un PDM : $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
Nécessite: une politique π
Nécessite: un seuil de précision ϵ
  initialiser V_0^{π} aléatoirement
  $i \leftarrow 0$
  répéter
    pour tout état $s \in \mathcal{S}$ **faire**
      $V_{i+1}^{\pi}(s) \leftarrow \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_i^{\pi}(s')]$
    fin pour
    $i \leftarrow i + 1$
  jusque $||V_i^{\pi} - V_{i-1}^{\pi}|| \leq \epsilon$

La norme intervenant dans le **jusque** de l'algorithme peut être calculée de différentes manières sans que cela change quoi que ce soit à la convergence de l'algorithme : normes usuelles :

- norme L_1 : $||V_i^{\pi} - V_{i-1}^{\pi}||_1 = \sum_{s \in \mathcal{S}} |V_i^{\pi}(s) - V_{i-1}^{\pi}(s)|$
- norme L_2 (norme euclidienne) : $||V_i^{\pi} - V_{i-1}^{\pi}||_2 = \sum_{s \in \mathcal{S}} (V_i^{\pi}(s) - V_{i-1}^{\pi}(s))^2$
- norme L_{∞} (norme max) : $||V_i^{\pi} - V_{i-1}^{\pi}||_{\infty} = \max_{s \in \mathcal{S}} |V_i^{\pi}(s) - V_{i-1}^{\pi}(s)|$

Propriété 4 *Pour un PDM fixé et une politique π fixée, l'algorithme d'évaluation de la politique converge asymptotiquement vers V^{π}.*

2.2 Algorithme d'amélioration de la politique

Ayant une politique, on veut en obtenir, si possible, une meilleure, jusqu'à obtenir une politique optimale.

On a précisé plus haut ce que l'on entend par « politique optimale ». Par contre, on doit préciser ce que l'on entend par le fait qu'une politique est « meilleure » qu'une autre. On définit donc un ordre entre les politiques qui nous permettra de les comparer :

Définition 5 *Une politique π est meilleure qu'une politique π', et on le note $\pi \geq \pi'$, si $\forall s \in \mathcal{S}$, $V^{\pi}(s) \geq V^{\pi'}(s)$.*

Le principe de l'amélioration de la politique va consister très simplement à choisir pour chaque état $s \in \mathcal{S}$, l'action qui maximise l'espérance de retours calculé à partir de la politique courante π, soit :

$$\pi'(s,a) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')] \tag{4}$$

Partant d'une politique π pour laquelle on a calculé V^{π}, ayant ainsi traité chaque état et obtenu une nouvelle politique π', on est assuré que $V^{\pi'} \geq V^{\pi}$ en tout état. On a une nouvelle propriété très intéressante :

Propriété 5 *Dans un PDM, ayant appliqué l'équation (4), si $\pi = \pi'$, alors $\pi = \pi^*$.*

2.3 Algorithme d'itération de la politique

Donc, nous avons vu comment, disposant d'une politique, on peut estimer sa valeur et nous venons de voir comment, disposant d'une fonction valeur, on peut produire une politique meilleure que celle associée à cette fonction valeur, à moins que l'on ne puisse tout simplement pas l'améliorer car étant déjà optimale. Cela nous fournit immédiatement l'idée d'un algorithme pour produire une politique optimale pour un PDM :

1. démarrer avec une politique quelconque
2. calculer sa valeur
3. construire une politique meilleure que la précédente
4. retourner à l'étape tant que l'on arrive à produire une politique strictement meilleure.

Formalisé, cela nous donne l'algorithme d'itération de la politique 2. La seule difficulté est que pour améliorer la politique, nous travaillons avec des politiques déterministes alors que dans l'évaluation de la politique, nous travaillons avec des politiques non déterministes. Pour mettre cela en cohérence, nous adaptons l'algorithme d'évaluation de la politique à une politique déterministe.

Propriété 6 *L'algorithme d'itération de la politique converge vers π^* au bout d'un nombre fini d'itérations.*

Algorithme 2 L'algorithme d'itération de la politique.

Nécessite: un PDM : $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
Nécessite: un seuil de précision ϵ
  initialiser π_0 aléatoirement
  $k \leftarrow 0$
  répéter
    initialiser V_0^{π} aléatoirement
    $i \leftarrow 0$
    répéter
      pour tout état $s \in \mathcal{S}$ **faire**
        $V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}(s, \pi_k(s), s')[\mathcal{R}(s, \pi(s), s') + \gamma V_i^{\pi_k}(s)]$
      fin pour
      $i \leftarrow i + 1$
    jusque $||V_i^{\pi_k} - V_{i-1}^{\pi_k}|| \leq \epsilon$
    pour tout état $s \in \mathcal{S}$ **faire**
      $\pi_{k+1}(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^{\pi_k}(s')]$
    fin pour
    $k \leftarrow k + 1$
  jusque $\pi_k = \pi_{k-1}$

Remarquons que l'évaluation de la politique se faisant avec une précision fixée, si ce seuil est mal choisi, la fonction valeur de la politique π_k est trop approximative et il peut arriver alors que $\pi_k = \pi_{k-1}$ alors que π_k n'est pas encore la politique optimale.

Donc, ce seuil doit être choisi judicieusement. Sa valeur doit dépendre de l'ordre de grandeur des retours.

2.4 Algorithme d'itération de la valeur

L'algorithme d'itération de la politique peut être optimisée de différentes manières. En particulier, des propriétés des PDM font que l'on peut calculer directement V^* et en déduire π^*. C'est l'algorithme d'itération de la valeur 3.

Algorithme 3 L'algorithme d'itération de la valeur.

Nécessite: un PDM : $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$
Nécessite: un seuil de précision ϵ
  initialiser V_0 aléatoirement
  $i \leftarrow 0$
  répéter
    pour tout état $s \in \mathcal{S}$ **faire**
      $V_{i+1}(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V_i(s')]$
    fin pour
    $i \leftarrow i + 1$
  jusque $||V_i - V_{i-1}|| \leq \epsilon$
  pour tout état $s \in \mathcal{S}$ **faire**
    $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V(s')]$
  fin pour

Propriété 7 *Pour tout PDM, l'algorithme d'itération de la valeur converge vers π^*.*

Remarque : ce que l'on a dit à propos du seuil dans l'algorithme d'itération de la politique demeure vrai ici.

2.5 Remarques sur l'implantation

Les algorithmes que l'on vient de rencontrer sont très faciles à implanter. Le lecteur peut être pour l'instant un peu débordé par les notations qui peuvent faire croire que l'implantation va être ardue. Il n'en est rien et nous montrons ici comment s'implante l'algorithme d'itération de la valeur.

La grande difficulté concerne la mise sous forme d'un problème de décision de Markov du problème que l'on veut résoudre. Ce point est discuté au chapitre 5.

Pour le reste, chaque état est numéroté entre 1 et $|\mathcal{S}|$, chaque action également, entre 1 et $|\mathcal{A}|$. La fonction valeur est donc simplement un tableau de $|\mathcal{S}|$ réels. Une politique déterministe est un tableau de $|\mathcal{S}|$ entiers compris entre 1 et $|\mathcal{A}|$. La fonction de transition est un tableau tridimensionnel de dimensions respectivement $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ et contenant des réels compris entre 0 et 1. La fonction retour est un tableau tridimensionnel de dimensions respectivement $|\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}|$ et contenant des réels.

Dans l'algorithme d'itération de la valeur, on peut, au choix, utiliser deux tableaux distincts pour V_i et V_{i+1} ou confondre les deux. L'algorithme converge dans les deux cas. Si on utilise deux tableaux, on a un algorithme de style Jacobi ; si on en n'utilise qu'un seul, l'algorithme est de style Gauss-Seidel.

2.6 Conclusion

- en pratique : on peut appliquer ces algorithmes à des PDM ayant jusque de l'ordre de 10^6 états ;
- en pratique : la complexité en temps de calcul est $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ pour l'algorithme d'itération de la valeur ;
- en pratique, pour calculer la fonction valeur, les méthodes de programmation linéaire sont moins efficaces que l'algorithme d"itération de la valeur et ne permettent de traiter que des problèmes de taille 100 fois plus petite que ceux traités par les algorithmes vus ici ;
- la recherche directe de la politique optimale est sans intérêt : la complexité est non polynomiale en $\mathcal{O}(|\mathcal{S}|^{|\mathcal{A}|})$ (c'est un problème $\mathcal{NP}$-difficile) ;
- de nombreuses heuristiques sont possibles pour accélérer l'itération de la valeur. On peut ne pas parcourir systématiquement tous les états à chaque balayage ; il faut cependant garantir que tous les états seront visités suffisamment de fois ; voir en particulier la *programmation dynamique temps-réel* (*cf.* Barto et al. [1995]).

3 Séance 3 : algorithmes TD

On va :

1. étudier le cas où l'on ne connaît ni $\mathcal{P}$ ni $\mathcal{R}$
2. définir des algorithmes pour apprendre une politique optimale dans ce cas.

Ne disposant pas d'un modèle de la dynamique de l'environnement, l'agent va maintenant devoir effectuer son apprentissage d'une politique optimale en interagissant avec son environnement. Des conséquences de ses interactions (retours perçus et transitions effectuées), il déduira une estimation de la fonction valeur (ou de la fonction qualité) qu'il raffinera petit à petit, d'où il déduira une politique. On fait ici de l'apprentissage « en-ligne » alors que les algorithmes de programmation dynamique vus précédemment font de l'apprentissage « hors-ligne » (sans interagir avec leur environnement).

3.1 Méthode de Monte Carlo

On ne s'intéresse ici qu'à des tâches épisodiques. On nomme « trajectoire » la suite d'états parcourus depuis un état initial à un état final.

Disposant d'une trajectoire et des retours associés à chaque transition, on peut observer les retours suivants le passage par chaque état de la trajectoire. Si l'on dispose d'un grand nombre de passages par cet état et d'un grand nombre de trajectoires, la moyenne des retours observés pour chaque état tend vers la vraie moyenne (loi des grands nombres). Cela nous donne l'idée de l'algorithme de Monte-Carlo 4 qui évalue une politique à partir des observations effectuées le long de trajectoires engendrées par cette politique.

Il faut que tous les états soient visités un grand nombre de fois pour obtenir une approximation satisfaisante de V^π.

On a donné ici une version très simple de l'algorithme de Monte-Carlo pour évaluer la valeur des états. Plusieurs variantes existent dont la convergence asymptotique vers la valeur des états a été démontrée.

3.2 La notion de différence temporelle

Une fois apprise, $V^\pi(s)$ quantifie l'intérêt d'être dans l'état s. En moyenne, on a l'égalité :

$$V^\pi(s_t) = r_t + \gamma V^\pi(s_{t+1})$$

Algorithme 4 Un algorithme de la famille Monte Carlo.

Nécessite: une politique π
  pour chaque état s **faire**
    pour $j \in \{0, ..., M\}$ **faire**
      générer une trajectoire en suivant la politique π à partir de l'état $s_0 = s$. On note $\{s_t, a_t, r_t\}_t$ la suite des triplets état, action, retour immédiat de cette trajectoire. On suppose la trajectoire de taille bornée T.
      $v(s_0, j) \leftarrow \sum_{t=0}^{t=T} \gamma^t r_t$
    fin pour
    $V^\pi(s) \leftarrow \sum_{j=0}^{j=M} v(s, j)$
  fin pour

soit :

$$\underbrace{r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)}_{\text{TD}} = 0$$

En fait, pendant l'apprentissage de la fonction V^π, TD n'est pas nul. Si l'estimation de V^π est trop optimiste en s, TD est négative ; si elle est trop pessimiste, TD est positive. Aussi, TD peut être utilisé comme une correction à apporter à $V^\pi(s_t)$. En notant V_t^π l'estimation de la fonction valeur à l'instant t, on peut corriger cette estimation comme suit :

$$V_{t+1}^\pi(s) \leftarrow \begin{cases} V_t^\pi(s) + \alpha[r_t + \gamma V_t^\pi(s_{t+1}) - V_t^\pi(s)] \text{ pour } s = s_t \\ V_t^\pi(s) \text{ sinon} \end{cases}$$

où $\alpha \in [0, 1]$ est un coefficient dénommé le « taux d'apprentissage ».

Dans un environnement déterministe, on prend $\alpha = 1$: on est certain de la valeur de la correction à apporter à $V^\pi(s)$.

Dans un environnement non déterministe, on aura initialement une estimation totalement erronée de la valeur ; au fur et à mesure des expériences, donc des itérations de cette équation de mise à jour, la précision de l'estimation augmente. Aussi, on prend α variable au cours du temps, initialement grand et tendant vers 0.

La correction TD est dénommée la « différence temporelle ».

3.3 L'algorithme TD (0)

En utilisant la différence temporelle, on obtient directement l'algorithme 5 qui évalue une politique fixée π : c'est l'algorithme TD(0).

Le taux d'apprentissage α doit varier : en effet, lors des premiers épisodes, les corrections sont importantes car l'estimation de V^π est mauvaise ; cependant, au fur et à mesure des épisodes, la précision de cette estimation augmente et les corrections deviennent moins significatives. Par contre, du fait du non déterminisme de l'environnement, les corrections peuvent être importantes alors que l'estimation de V^π est bonne. Aussi, cela nous fait penser qu'α doit être initialement important, proche de 1, et diminuer ensuite pour tendre vers 0. Pour que TD(0) converge effectivement vers V^π, on a des conditions plus précises qui doivent être respectées. On doit utiliser un taux d'apprentissage pour chaque état que l'on notera donc $\alpha_t(s)$ pour la valeur à l'itération t du taux d'apprentissage associé à l'état s.

Algorithme 5 L'algorithme TD(0).

Nécessite: une politique π
  $V^\pi \leftarrow 0$
  pour ∞ **faire**
    initialiser l'état initial s_0
    $t \leftarrow 0$
    répéter
      émettre l'action $a_t = \pi(s_t)$
      observer r_t et s_{t+1}
      $V^\pi(s_t) \leftarrow V^\pi(s) + \alpha[r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)]$
      $t \leftarrow t + 1$
    jusque $s_t \in \mathcal{F}$
  fin pour

Propriété 8 *Pour un PDM fixé et une politique π fixée, l'algorithme TD(0) converge asymptotiquement vers V^π à condition que :*

- *chaque état soit visité une infinité de fois,*
- *on ait :*

$$\sum_t \alpha_t(s) = +\infty \text{ et } \sum_t \alpha_t^2(s) < +\infty, \forall s \in \mathcal{S} \tag{5}$$

Concernant ce second point, on peut prendre :

$$\alpha_t(s) = \frac{1}{1 + \text{nombre de visites à l'état } s \text{ depuis le début de l'épisode}} \tag{6}$$

3.4 L'algorithme Q-Learning

L'algorithme TD(0) évalue une politique. Il nous faut maintenant chercher à l'améliorer et aboutir à une politique optimale. On pourrait appliquer la technique d'amélioration de la politique déjà rencontrée. Cependant, on va adopter ici une autre stratégie qui fournira un algorithme plus efficace, l'algorithme *Q-Learning*.

3.4.1 Principe de l'algorithme du *Q-Learning*

Le principe de l'algorithme consiste à apprendre la fonction qualité en interagissant avant son environnement : voir l'algorithme 6.

Le taux d'apprentissage doit varier de la même manière que dans l'algorithme TD(0).

3.4.2 Le choix de l'action

Le choix de l'action doit garantir un équilibre entre l'exploration et l'exploitation de l'apprentissage déjà réalisé : faire confiance à l'estimation courante de Q pour choisir la meilleure action à effectuer dans l'état courant (exploitation) ou, au contraire, choisir une action *a priori* sous-optimale pour observer ses conséquences (exploration). Naturellement, on conçoit intuitivement que l'exploration doit initialement être importante (quand l'apprentissage est encore très partiel, on explore) puis diminuer au profit de l'exploitation quand l'apprentissage a été effectué pendant une période suffisamment longue.

Plusieurs stratégies sont utilisées classiquement :

Algorithme 6 L'algorithme *Q-Learning*.

$Q(s,a) \leftarrow 0, \forall (s,a) \in (\mathcal{S}, \mathcal{A})$
pour ∞ **faire**
  initialiser l'état initial s_0
  $t \leftarrow 0$
  répéter
    choisir l'action à émettre a_t et l'émettre
    observer r_t et s_{t+1}
    $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_{a' \in \mathcal{A}(s_{t+1})} Q(s_{t+1},a') - Q(s_t,a_t)]$
    $t \leftarrow t+1$
  jusque $s_t \in \mathcal{F}$
fin pour

gloutonne : elle consiste à toujours choisir l'action estimée comme la meilleure, soit $a_{\text{gloutonne}} = \arg\max_{a \in \mathcal{A}(s_t)} Q(s_t,a)$;

ϵ-gloutonne : elle consiste à choisir l'action gloutonne avec une probabilité ϵ et à choisir une action au hasard avec une probabilité $1-\epsilon$, soit :

$$a_{\epsilon-\text{gloutonne}} = \begin{cases} \arg\max_{a \in \mathcal{A}(s_t)} Q(s_t,a) \text{ avec probabilité } \epsilon \\ \text{action prise au hasard dans } \mathcal{A}(s_t) \text{ avec probabilité } 1-\epsilon \end{cases} \quad (7)$$

Un cas particulier est la sélection 0-gloutonne qui consiste à choisir une action au hasard ;

softmax : elle consiste à choisir une action en fonction de sa qualité relativement à la qualité des autres actions dans le même état, soit : on associe par exemple à chaque action a la probabilité d'être émise selon l'équation suivante :

$$Pr[a_t|s_t] = \frac{Q(s_t,a_t)}{\sum_{a \in \mathcal{A}(s_t)} Q(s_t,a)}$$

boltzmann : c'est un cas particulier de sélection *softmax* : la probabilité est calculée avec l'équation suivante qui produit une distribution dite de Boltzmann :

$$Pr[a_t|s_t] = \frac{e^{\frac{Q(s_t,a_t)}{\tau}}}{\sum_{a \in \mathcal{A}(s_t)} e^{\frac{Q(s_t,a)}{\tau}}}$$

où τ est un réel positif. Si τ est grand, cette méthode tend vers la méthode gloutonne ; si τ est proche de 0, elle tend alors vers une 0-gloutonne. Généralement, on utilise cette méthode en faisant varier τ au cours de l'apprentissage, en démarrant avec une grande valeur assurant l'exploration de l'espace des actions, puis en diminuant progressivement pour augmenter l'exploitation de l'apprentissage déjà effectué.

La sélection de l'action est un thème classique en intelligence artificielle. Beaucoup d'encre a coulé et coule encore. Une idée récurrente est de focaliser l'apprentissage dans les zones de l'espace d'états dans lesquelles l'estimation de Q est la moins bonne ; simple à exprimer, sa formalisation algorithmique l'est beaucoup moins...

3.4.3 Convergence du *Q-Learning*

On a :

Propriété 9 *Pour un PDM fixé, l'algorithme du Q-Learning converge asymptotiquement vers une politique optimale à condition que :*

- *chaque paire (état, action) soit visitée une infinité de fois ;*
- *les conditions décrites par les équations (5) soient respectées.*

3.4.4 L'algorithme SARSA

On donne ici l'algorithme SARSA qui ressemble beaucoup au *Q-Learning* mais présente une différence très importante, même si elle peut sembler mineure au premier coup d'œil : *cf.* algorithme 7.

Algorithme 7 L'algorithme SARSA.

$Q(s,a) \leftarrow 0, \forall (s,a) \in (\mathcal{S}, \mathcal{A})$
pour ∞ **faire**
  initialiser l'état initial s_0
  $t \leftarrow 0$
  choisir l'action à émettre a_t en fonction de la politique dérivée de Q (ϵ-gloutonne par exemple) et l'émettre
  répéter
    émettre a_t
    observer r_t et s_{t+1}
    choisir l'action à émettre a_{t+1} en fonction de la politique dérivée de Q (ϵ-gloutonne par exemple)
    $Q(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
    $t \leftarrow t + 1$
  jusque $s_t \in \mathcal{F}$
fin pour

L'analyse de la convergence de SARSA est plus difficile que celle du *Q-Learning*.

Propriété 10 *SARSA converge vers la politique optimale si :*

- *toutes les paires (état, action) sont visitées une infinité de fois ;*
- *la stratégie de choix de l'action tend vers une stratégie gloutonne.*

Pour remplir cette dernière condition, on peut prendre $\epsilon = 1/t$.

En comparant SARSA et *Q-Learning*, on voit que SARSA utilise dans la mise à jour de Q l'action qui va être émise à l'itération suivante : on dit que c'est un algorithme *on-policy*. Au contraire, *Q-Learning* choisit une action à émettre et met à jour Q en fonction de la meilleure action (celle prédite comme telle) dans l'état suivant : c'est un algorithme *off-policy*.

On n'en dit pas plus ici ; la comparaison de SARSA avec le *Q-Learning* est une activité tout à fait intéressante et formatrice...

3.5 TD (λ) et Q (λ)

Lorsqu'un certain retour est obtenu, cela est du à la dernière transition, mais aussi, dans une moindre mesure à l'avant dernière, dans une mesure encore moindre à l'antépénultième, ... Les algorithmes TD(0) et *Q-Learning* ne prennent en compte le retour immédiat que pour la dernière transition pour modifier la fonction valeur. On peut donc, intuitivement, propager ces conséquences aux transitions précédentes, le long de la trajectoire. C'est exactement l'idée des algorithmes TD (λ) et Q (λ) qui ne sont qu'une généralisation des TD(0) et *Q-Learning*.

Cette généralisation s'appuie sur la notion importante d'éligibilité : est éligibile un état, ou une paire état-action, qui est, au moins en partie, responsable du retour immédiat. Cette éligibilité est quantifiée par le niveau de responsabilité : le dernier état ou la dernière paire état-action est la plus responsable, donc possède l'éligibilité la plus grande, ...

Dans les deux section qui suivent, on précise ces idées et on définit précisément ces deux algorithmes.

3.5.1 TD (λ)

L'algorithme TD (λ) évalue une politique fixée π. Outre l'apprentissage en-ligne via la différence temporelle, TD (λ) maintient à jour l'éligibilité des états et l'utilise pour mettre à jour la valeur des états rencontrés depuis l'état initial. Il est donné par l'algorithme 8.

Pour un état $s \in \mathcal{S}$ donné, son éligibilité $e(s)$ est initialement nulle. Elle est incrémentée à chaque visite de cet état ; sinon, elle diminue d'un facteur $\gamma\lambda$ à chaque itération.

Algorithme 8 L'algorithme TD (λ).

Nécessite: une politique π
  $V^\pi \leftarrow 0$ (valeur arbitraire)
  pour ∞ **faire**
    $e(s) \leftarrow 0, \forall s \in \mathcal{S}$
    initialiser l'état initial s_0
    $t \leftarrow 0$
    répéter
      émettre l'action $a_t = \pi(s_t)$
      observer r_t et s_{t+1}
      $\delta \leftarrow r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$
      $e(s_t) \leftarrow e(s_t) + 1$
      pour $s \in \mathcal{S}$ **faire**
        $V^\pi(s) \leftarrow V^\pi(s) + \alpha\delta e(s)$
        $e(s) \leftarrow \gamma\lambda e(s)$
      fin pour
      $t \leftarrow t + 1$
    jusque $s_t \in \mathcal{F}$
  fin pour

Remarque : TD(1) est équivalent à Monte Carlo visites multiples, donc converge vers V^π.

Propriété 11 *Pour un PDM fixé, TD (λ) converge vers V^π.*

3.5.2 Améliorer la politique suivie par un TD (λ)

L'algorithme TD (λ) évalue la valeur d'une politique fixée π, soit évalue V^π. On peut se poser la question de l'amélioration de cette politique. L'algorithme Q (λ) que l'on va présenter à la section suivante est une solution. Une autre solution simple est l'utilisation de la technique de *rollout*. Son principe repose sur une combinaison de TD (λ) avec méthode de Monte Carlo. Son principe est donné dans l'algorithme 9.

Propriété 12 *TD (λ) avec utilisation de* rollout *converge vers V^*.*

A priori, cet algorithme peut entraîner des temps de convergence non négligeables. Cependant, différentes stratégies peuvent être utilisées pour l'accélérer.

Algorithme 9 L'algorithme TD (λ) avec *rollout*.

Nécessite: une politique π initiale
  pour ∞ **faire**
    initialiser s
    répéter
      pour toute action $a \in \mathcal{A}(s)$ **faire**
        répéter
          exécuter a
          poursuivre l'épisode jusque sa fin en suivant π
          mémoriser les retours ayant suivi l'exécution de a
        jusque un grand nombre de fois
        mettre à jour $\pi(s) \leftarrow \arg\max_a$ la moyenne de ces retours pour l'action a
      fin pour
      exécuter $\pi(s)$
    jusque s est un état final
  fin pour

3.5.3 Q (λ)

L'algorithme Q (λ) est obtenu comme extension du *Q-Learning* à l'utilisaiton de traces d'éligibilité. Il est donné par l'algorithme 10. Il apprend une politique en interagissant avec son environnement.

À propos de la mise à jour de l'éligibilité de (s, a), plusieurs options ont été proposées :

Watkins effectue le traitement suivant : si $a = a^*$, alors $e(s, a) \leftarrow \gamma\lambda e(s, a)$, sinon $e(s, a) \leftarrow 0$;

naïf effectue le traitement suivant : $e(s, a) \leftarrow \gamma\lambda e(s, a)$;

Peng est une variante plus complexe que nous ne décrivons pas.

Remarque : Q(1) est équivalent à l'algorithme de contrôle de Monte Carlo visites multiples dont on n'a pas démontré, à ce jour, la convergence.

Plus généralement, la convergence de l'algorithme Q(λ) est une question ouverte pour $\lambda \neq 0$.

3.5.4 SARSA (λ)

On peut naturellement adapter l'algorithme SARSA à l'utilisation de traces d'éligibilité. ...

3.6 Grands espaces d'états ; utilisation d'approximateurs de fonction

Jusqu'à maintenant, la fonction valeur et la fonction qualité ont toujours étaient implicitement représentées exactement, à l'aide d'un tableau. Pour que la valeur d'un état soit modifiée, les algorithmes précédents imposent que cet état soit visité. Ceci implique que la mise à jour de la valeur d'un état ne modifie pas la valeur d'autres états.

Cette hypothèse n'est pas tenable dans le cas de grands espaces d'états. Dans ce cas, le balayage de l'ensemble des états n'est plus possible et l'apprentissage de la fonction valeur encore moins. La solution classique pour résoudre ce problème consiste à ne plus utiliser une représentation tabulaire de la fonction valeur, mais d'utiliser une représentation plus compacte, ayant des propriétés de généralisation. Ce peut être une représentation par réseau de neurones. On sait qu'un perceptron multi-couches peut approximer n'importe quelle fonction réelle, continue et bornée (propriété d'approximation universelle), ce qui permet un certain optimisme.

Algorithme 10 L'algorithme Q (λ).

$Q^\pi \leftarrow 0$ (valeur arbitraire)
pour ∞ **faire**
  $e(s,a) \leftarrow 0, \forall (s,a) \in \mathcal{S} \times \mathcal{A}$
  $t \leftarrow 0$
  initialiser l'état initial s_0
  choisir l'action à émettre a_0
  répéter
    émettre l'action a_t
    observer r_t et s_{t+1}
    choisir l'action qui sera émise dans s_{t+1}
    $a^* \leftarrow \arg\max_{a \in \mathcal{A}(s_{t+1})} Q(s_{t+1}, a)$
    $\delta \leftarrow r_t + \gamma Q(s_{t+1}, a^*) - Q(s_t, a_t)$
    $e(s_t, a_t) \leftarrow e(s_t, a_t) + 1$
    pour $(s,a) \in \mathcal{S} \times \mathcal{A}$ **faire**
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      mise à jour de l'éligibilité de (s,a)
    fin pour
    $t \leftarrow t + 1$
  jusque $s_t \in \mathcal{F}$
fin pour

Typiquement, lors de l'utilisation d'un réseau de neurones, on place l'état en entrée tandis que la sortie fournit une estimation de la valeur de cet état. En entrée, il est bon que l'état se définisse au travers de plusieurs caractéristiques, chacune fournissant l'une des entrées du réseau de neurones.

Les poids du réseau de neurones constituent les paramètres permettant de calculer l'estimation courante de la valeur d'un état. La valeur estimée d'un état est donc maintenant $\hat{V}(s) = f(s, \theta)$ où θ est la valeur de l'ensemble des paramètres utilisés par la fonction f d'estimation de la valeur. Par exemple, f est un certain réseau de neurones et θ sont ses poids à un certain instant. Le but est de trouver θ^* tel que $||f(s, \theta^*) - V^*(s)||$ soit minimal pour tout état s.

L'algorithme TD (λ) ou Q(λ) fonctionne comme d'habitude si ce n'est que quand il veut connaître l'estimation courante de la valeur d'un état, il place cet état en entrée du réseau et sa sortie fournit cette estimation. La mise à jour de l'estimation de la fonction valeur est ici réalisée par apprentissage du réseau de neurones, c'est-à-dire, modification de ses poids. L'erreur TD est utilisée pour cette modification par l'algorithme de rétro-propagation de l'erreur.

Pour les algorithmes utilisant des traces d'éligibilité, l'éligibilité concerne ici les paramètres (les poids d'un réseau de neurones) puisque c'est à partir des poids qu'est estimée la valeur d'un état. (Dans une représentation tabulaire, les paramètres sont simplement les valeurs des états ; dans ce cas, il y a équivalence entre valeur d'un état et paramètre ; en utilisant une représentation paramétrique, ce n'est plus le cas.)

Si du point de vue du principe, l'utilisation d'un réseau de neurones dans un TD (λ) ne pose pas de problème réel, du point de vue pratique, de nombreuses précautions doivent être prises pour que le réseau apprenne correctement la fonction valeur. En effet, nous sommes face à une tâche d'apprentissage en-ligne, ce qui ne constitue pas le mode d'utilisation le plus classique des réseaux de neurones. En général, en apprentissage supervisé, tous les exemples sont fournis initialement au réseau de neurones, avant que celui-ci ne soit utilisé pour prédire la classe de données. Dans le cas présent, l'apprentissage se fait en

interagissant avec l'environnement, c'est-à-dire que chaque itération de l'algorithme fournit une nouvelle donnée à intégrer dans l'apprentissage et qu'en même temps qu'il apprend, l'algorithme doit sélectionner les actions à effectuer en se basant sur l'apprentissage partiel qui a déjà été réalisé. De plus, la nature de l'erreur qui est rétro-propagée est très différente : en apprentissage supervisé, on fournit la valeur qui aurait du être produite par le réseau de neurones; en apprentissage par renforcement, on fournit seulement une différence temporelle : c'est beaucoup moins précis. On ne saurait trop conseiller de lire différentes sources avant de se lancer dans une implantation (*cf.* Cun [1998], Neuneier and Zimmermann [1998], Schraudolph [1999], Coulom [2002]).

Une application remarquable de ces idées alliée à la technique de *rollout* est le TD-Gammon décrit dans Tesauro [1995], meilleur programme de jeu de Backgammon, qui a appris à jouer en jouant des parties contre lui-même. Son jeu le place au niveau des meilleurs experts humains.

3.7 Conclusion

Dans cette partie, nous avons décrit des méthodes puissantes pour qu'un agent apprenne un comportement optimal dans un environnement pour lequel il ne dispose pas de modèle. Ces algorithmes ont été mis en œuvre dans des applications réelles de grande taille.

Notons également que l'apprentissage par renforcement fournit un cadre de modélisation de la dynamique comportementale animale. Certains travaux de recherche s'appuient sur cette idée.

4 Séance 4 : extensions du cadre des PDM

Dans cette section, on sort du cadre PDM défini plus haut pour aborder différents points qui changent fondamentalement la nature du problème de contrôle. On examine rapidement :

- le cas où l'espace d'états, l'espace d'actions et le temps sont continus ;
- le cas où l'état perçu n'est pas le reflet exact de l'état réel ;
- le cas où la dynamique du système varie au cours du temps.

On aurait pu également parler du cas où les actions durent dans le temps. En effet, dans tout ce qui est traité ici, on fait comme si les actions avaient une durée nulle : dans le cas d'un PDM discret, une action initiée à l'instant t est terminée à l'instant $t+1$. Cette hypothèse peut être levée si l'on se place dans les semi-PDM. On consultera Sutton et al. [1999].

4.1 PDM dans le cas continu

Dans le cas continu, la différence fondamentale provient du fait que le temps ne s'égrène plus de manière discrète, passant de t à $t+1$ au pas suivant. Ici, le temps évolue continûment, donc passe de t à $t+dt$ où dt est un infiniment petit. Pendant ce laps de temps, l'état varie d'une quantité $s + \frac{ds}{dt}$, où $\frac{ds}{dt}$ s'interprète naturellement comme une vitesse de variation d'état. Une action consiste à faire varier le terme $\frac{ds}{dt}$, soit en un terme de type $\frac{d^2s}{dt^2}$, soit une accélération.

L'équation de Bellman caractérisant une politique et l'équation d'optimalité de Bellman caractérisant la politique optimale s'exprime maintenant sous une forme continue. L'équation de Bellman exprime la valeur d'un état donné en fonction de la valeur des états suivants :

$$V(s_t) = \sum_{k>0} f(V(s_{t+k}))$$

En passant à la limite, s_{t+1} devient s_{t+dt} et la somme devient une intégrale pour l'ensemble des instants futurs, soit :

$$V(s) = \int_{t=\text{instant courant}}^{t=+\infty} f(V(s_t))dt$$

dans le cas continu. Plus précisément, on obtient l'équation de Hamilton-Jacobi-Bellman (HJB) qui généralise l'équation d'optimalité de Bellman dans le cas continu. Dans le cas d'un environnement déterministe, elle s'exprime comme suit :

$$0 = \max_{a \in \mathcal{A}} (r(s,a) - s_\gamma V^*(s) + \frac{\partial V^*}{\partial s} \mathcal{P}(s,a)) \tag{8}$$

dans laquelle $\mathcal{P}(s,a)$ dénote l'état atteint après l'émission de l'action a dans l'état s. s_γ est lié au coefficient de dépréciation par la relation $\gamma = e^{-s_\gamma \delta t}$.

Pour toute fonction V, le terme

$$\mathcal{H} = \max_{a \in \mathcal{A}} (r(s,a) - s_\gamma V(s) + \frac{\partial V}{\partial s} \mathcal{P}(s,a))$$

est dénommé l'hamiltonien de V.

Cette équation différentielle caractérise la fonction valeur associée à la politique optimale. Son intégration fournira cette fonction valeur, de laquelle on déduira la politique optimale :

$$\pi^*(s) = \arg\max_a (\mathcal{R}(s,a) + \frac{\partial V^*}{\partial s} \mathcal{P}(s,a))$$

L'intégration de l'équation d'HJB n'est pas simple car d'une manière générale, la fonction valeur optimale V^* n'est pas dérivable partout ; elle est « généralement » dérivable, c'est-à-dire qu'elle est dérivable sauf en un nombre fini de points. Les méthodes de descente de gradient ne sont pas adaptées dans ce cas : elles convergent vers des optima locaux. Un résultat remarquable établit qu'il existe une et une seule solution dite « solution de viscosité » à l'équation d'HJB qui correspond à la fonction valeur optimale recherchée, ceci pour un grand nombre de problème de contrôle optimal. Deux approches sont possibles pour trouver cette solution :

1. transformer le problème continu en un problème discret en utilisant des techniques de discrétisation ;
2. résoudre le problème directement en continu.

On examine rapidement ces deux approches.

4.1.1 Discrétisation d'un PDM continu

Dans cette première approche, on discrétise le problème : l'ensemble des états est discrétisé, par exemple avec un pas constant. On retrouve le cas d'un PDM discret que l'on peut résoudre avec un algorithme de programmation dynamique classique. La solution obtenue est alors une approximation de la fonction valeur recherchée. De plus, la solution de viscosité est assurée de tendre vers cette fonction quand le pas de discrétisation tend vers 0.

Cette technique est utilisable avec un algorithme de programmation dynamique (quand la dynamique de l'environnement est connue), mais aussi avec des algorithmes d'apprentissage à base de différence temporelle. Voir Munos [2000] pour plus de détails.

Une discrétisation à pas constant souffre de défauts : le pas est trop petit dans certaines zones, entraînant des calculs inutiles ; le pas est trop grand dans d'autres, entraînant de grandes imprécisions.

Pour pallier ces problèmes, dans le cas où la dynamique de l'environnement est connue (on connaît donc $\mathcal{P}$ et $\mathcal{R}$), Munos and Moore [2001] proposent d'effectuer une discrétisation adaptative. L'idée est de discrétiser initialement relativement grossièrement. Ensuite, entre deux états voisins à ce pas de discrétisation, si la variation de la fonction valeur estimée est trop importante, on discrétise plus finement. On itère ce processus jusqu'à obtenir une approximation suffisamment précise.

4.1.2 Résolution d'un PDM continu

En représentant la fonction valeur par une fonction paramétrée, l'équation HJB se met sous la forme d'un système d'équations différentielles reliant les paramètres, l'état et les éligibilités. L'intégration de ce système n'est pas simple, notamment d'un point de vue technique, pour que l'algorithme converge vers la bonne fonction valeur.

Malgré cela, Coulom [2002] a résolu plusieurs problèmes de contrôle optimal continus par apprentissage par renforcement en utilisant un réseau de neurones comme approximateur de fonction.

4.2 PDM partiellement observables

Ici, l'environnement suit une dynamique markovienne. Cependant, l'agent n'a pas accès à l'état mais seulement à des observations. On ne donne ci-dessous que quelques idées sur ces problèmes et on conseille la lecture de Aberdeen [2003], Murphy [2000] pour en savoir plus.

Définition 6 *Un PDM partiellement observables (PDM-PO) est défini par un sextuplet :*

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \mathcal{T})$$

où $\mathcal{S}$, $\mathcal{A}$, $\mathcal{P}$ et $\mathcal{R}$ sont les composants habituels d'un PDM et $\mathcal{Z}$ l'ensemble des observations possibles, T la fonction de transition entre les observations :

$$\begin{aligned} \mathcal{T} \quad : \quad & \mathcal{S} \times \mathcal{A} \times \mathcal{Z} \to [0,1] \\ & (s,a,z) \mapsto \mathcal{T}(s,a,z) = Pr[z_{t+1} = z | s_t = s, a_t = a] \end{aligned}$$

Le PDM définit par $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ est qualifié de « sous-jacent » : on suppose qu'il existe mais on ne suppose pas forcément qu'on le connaît.

Définition 7 *La fonction de perception ϕ est définie par :*

$$\begin{aligned} \phi \quad : \quad & \mathcal{S} \times \mathcal{Z} \to [0,1] \\ & s(s,z) \mapsto Pr[z_t = z | s_t = s] \end{aligned}$$

Le problème est toujours de trouver une stratégie optimale π^*. Contrairement au cas des PDM, il existe généralement des optima locaux. En outre, la définition même de politique n'est pas évidente. Plusieurs définitions de ce qu'est une politique viennent à l'esprit :

1. $\pi : \mathcal{Z} \to \mathcal{A}$;
2. $\pi : \mathcal{Z} \to \Pi(\mathcal{A})$ où la notation $\Pi(\mathcal{A})$ signifie « distribution de probabilité sur l'ensemble $\mathcal{A}$ » ;
3. π : histoire de l'agent $\to \mathcal{A}$ ou $\Pi(\mathcal{A})$, où l'histoire de l'agent est l'ensemble de ses triplets (z_t, a_t, r_t) depuis $t = 0$
4. ...

4.2.1 Cas où le PDM-PO est connu

L'histoire de l'agent est une entité qui acquiert rapidement une grande taille et n'est pas facile à manipuler ni à traiter. On préfère utiliser une statistique résumant adéquatement l'histoire de l'agent. Ainsi, on définit :

Définition 8 *L'état de croyance $b_t(s)$ d'un agent est la probabilité d'être dans l'état s à l'instant t.*

b est donc une distribution de probabilités sur $\mathcal{S}$.

Plus généralement, on peut définir $b(z, a, s)$, la probabilité d'être dans l'état s si, suite à l'observation z, on a émis l'action a.

Disposant de b_t et ayant émis l'action a_t et observé z_{t+1}, on met à jour l'état de croyance pour obtenir :

$$b_{t+1}(s') = \frac{\phi(s', z_{t+1}) \sum_s \mathcal{P}(s, a_t, s') b_t(s)}{\sum_{s,s''} \phi(s'', z_{t+1}) \mathcal{P}(s, a_t, s'') b_t(s)}$$

Propriété 13 *La connaissance de l'état de croyance courant est suffisante pour contrôler l'agent de manière optimale et markovienne.*

Propriété 14 *La fonction valeur des états de croyance est linéaire par morceaux.*

Plusieurs algorithmes ont été proposés :
- voir Cassandra [1999] ;
- Q-MDP : adaptation du *Q-Learning* à des états de croyance : $Q(b, a)$. Une application de cette approche est GIB, le meilleur programme de Bridge actuel Ginsberg [1999].

4.2.2 Cas où le PDM sous-jacent est inconnu

- si on connaît $\mathcal{Z}$:
  - approche exacte : on construit un PDM sous-jacent à l'aide des observations. Cette approche est très lourde en terme de temps de calcul ;
  - approche non exacte :
    - *Q-Learning* adapté à des observations : $Q(z, a)$;
    - on utilise un algorithme d'optimisation de modèle (Lion, CSQL, ... : *cf.* Whitehead and Lin [1995]) ;
    - on utilise les chaînes de markov cachées pour essayer d'estimer le modèle le plus vraisemblable du PDM sous-jacent. Les techniques dites de « filtres particulaires » semblent les mieux adaptées Doucet et al. [2000], mieux adaptées que l'algorithme de Baum-Welch.
- si on ne connaît pas $\mathcal{Z}$ (donc, on ne connaît pas $\mathcal{T}$), on essaie de construire un PDM au fur et à mesure de l'acquisition d'expériences en-ligne.

4.2.3 Quelques résultats

Les résultats théoriques sur les PDM-PO sont plus difficiles à obtenir. On a néanmoins montré (*cf.* Singh et al. [1994]) que :

Propriété 15
- *d'une manière générale, il n'y a pas de stratégie optimale stationnaire ;*
- *cette stratégie optimale ne peut généralement pas être trouvée par un algorithme TD ;*
- *la stratégie optimale n'est plus forcément un point fixe pour l'algorithme d'itération de la stratégie ;*

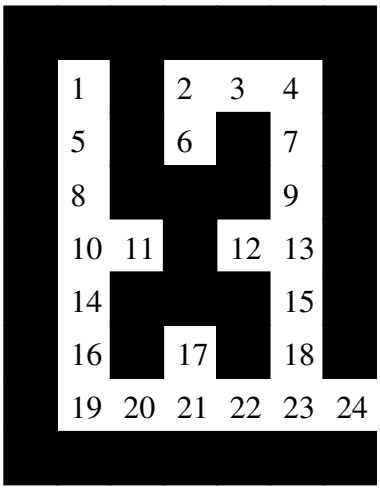

FIG. 1 – Labyrinthe utilisé dans l'exemple. Les numéros indiquent les états.

- *la meilleure stratégie stationnaire non déterministe peut être arbitrairement meilleure que la meilleure stratégie déterministe ;*
- *la meilleure stratégie stationnaire non déterministe peut être arbitrairement pire que la meilleure stratégie du PDM sous-jacent.*

4.3 PDM non stationnaires

On peut distinguer le cas où la dynamique du système n'est pas constante mais reste proche d'une dynamique moyenne, du cas où la dynamique évolue significativement. Dans ce second cas, on peut encore distinguer le cas où la dynamique du système évolue de manière cyclique, du cas où cette évolution est apériodique.

5 Le labyrinthe

5.1 Le problème

On se donne un labyrinthe en 2D que l'on voit « par dessus ». On veut apprendre une politique nous amenant le plus vite possible vers une sortie, quelle que soit la position de départ.

On utilisera le labyrinthe de la figure 1.

5.2 Formulation comme un PDM

On doit mettre ce problème sous forme d'un PDM, soit exprimer le quadruplet $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$:

- $\mathcal{S}$: l'ensemble des états possibles est constitué de l'ensemble des positions possibles dans le labyrinthe, soit les cases de couloir. On numérote les cases du labyrinthe qui ne sont pas des murs ; chaque état est ainsi identifié par un entier naturel. Ici, ces états seront donc numérotés de 1 à 24. 24 est l'état final ;
- $\mathcal{A}$: l'ensemble des actions est : rester sur place, aller d'une case vers la gauche, une case vers la droite, une case vers le haut, une case vers le bas. On identifie chaque action par un entier naturel, respectivement 0, 1 ($\leftarrow$), 2 ($\rightarrow$), 3 ($\uparrow$) et 4 ($\downarrow$) ;
- $\mathcal{P}$: pour chaque triplet (s, a, s'), on calcule la probabilité de passer de l'état s à l'état s' en effectuant l'action a. On suppose ici que les déplacements sont certains : si l'on effectue l'action aller à gauche

et que la case à gauche n'est pas un mur, l'état suivant est cette case. Par exemple, on aura pour l'état 10 :

$$\begin{aligned}
\mathcal{P}(10,0,10) &= 1 \\
\mathcal{P}(10,0,s \neq 10) &= 1 \\
\mathcal{P}(10,1,10) &= 1 \\
\mathcal{P}(10,1,s \neq 10) &= 1 \\
\mathcal{P}(10,2,11) &= 1 \\
\mathcal{P}(10,2,s \neq 11) &= 0 \\
\mathcal{P}(10,3,8) &= 1 \\
\mathcal{P}(10,3,s \neq 8) &= 0 \\
\mathcal{P}(10,4,14) &= 1 \\
\mathcal{P}(10,4,s \neq 14) &= 1
\end{aligned}$$

– $\mathcal{R}$: on suppose que l'arrivée dans un état de sortie du labyrinthe provoque un retour de 1, alors que toutes les autres transitions entraînent un retour nul :

$$\mathcal{R}(s,a,s') = \begin{cases} 1 \text{ si } s' = 24 \\ 0 \text{ si } s' \neq 24 \end{cases}$$

5.3 La fonction valeur

La fonction valeur associée à la politique optimale vérifie l'ensemble d'équations suivantes, lesquelles proviennent de l'application de l'équation d'optimalité de Bellman :

$$
\begin{cases}
V^*(1) = \max_{a\in\{0,4\}}\{\gamma V^*(1), \gamma V^*(5)\} \\
V^*(2) = \max_{a\in\{0,2,4\}}\{\gamma V^*(2), \gamma V^*(3), \gamma V^*(6)\} \\
V^*(3) = \max_{a\in\{0,1,2\}}\{\gamma V^*(2), \gamma V^*(3), \gamma V^*(4)\} \\
V^*(4) = \max_{a\in\{0,1,4\}}\{\gamma V^*(3), \gamma V^*(4), \gamma V^*(7)\} \\
V^*(5) = \max_{a\in\{0,3,4\}}\{\gamma V^*(1), \gamma V^*(5), \gamma V^*(8)\} \\
V^*(6) = \max_{a\in\{0,3\}}\{\gamma V^*(2), \gamma V^*(6)\} \\
V^*(7) = \max_{a\in\{0,3,4\}}\{\gamma V^*(4), \gamma V^*(7), \gamma V^*(9)\} \\
V^*(8) = \max_{a\in\{0,3,4\}}\{\gamma V^*(5), \gamma V^*(8), \gamma V^*(10)\} \\
V^*(9) = \max_{a\in\{0,3,4\}}\{\gamma V^*(7), \gamma V^*(9), \gamma V^*(13)\} \\
V^*(10) = \max_{a\in\{0,2,3,4\}}\{\gamma V^*(8), \gamma V^*(10), \gamma V^*(11), \gamma V^*(14)\} \\
V^*(11) = \max_{a\in\{0,1\}}\{\gamma V^*(10), \gamma V^*(11)\} \\
V^*(12) = \max_{a\in\{0,2\}}\{\gamma V^*(11), \gamma V^*(13)\} \\
V^*(13) = \max_{a\in\{0,1,3,4\}}\{\gamma V^*(9), \gamma V^*(12), \gamma V^*(13), \gamma V^*(15)\} \\
V^*(14) = \max_{a\in\{0,3,4\}}\{\gamma V^*(10), \gamma V^*(14), \gamma V^*(16)\} \\
V^*(15) = \max_{a\in\{0,3,4\}}\{\gamma V^*(13), \gamma V^*(15), \gamma V^*(18)\} \\
V^*(16) = \max_{a\in\{0,3,4\}}\{\gamma V^*(14), \gamma V^*(16), \gamma V^*(19)\} \\
V^*(17) = \max_{a\in\{0,4\}}\{\gamma V^*(17), \gamma V^*(21)\} \\
V^*(18) = \max_{a\in\{0,3,4\}}\{\gamma V^*(15), \gamma V^*(18), \gamma V^*(23)\} \\
V^*(19) = \max_{a\in\{0,2,3\}}\{\gamma V^*(16), \gamma V^*(19), \gamma V^*(20)\} \\
V^*(20) = \max_{a\in\{0,1,2\}}\{\gamma V^*(19), \gamma V^*(20), \gamma V^*(21)\} \\
V^*(21) = \max_{a\in\{0,1,2,3\}}\{\gamma V^*(17), \gamma V^*(20), \gamma V^*(21), \gamma V^*(22)\} \\
V^*(22) = \max_{a\in\{0,1,2\}}\{\gamma V^*(21), \gamma V^*(22), \gamma V^*(23)\} \\
V^*(23) = \max_{a\in\{0,1,2,3\}}\{\gamma V^*(18), \gamma V^*(22), \gamma V^*(23), 1 + \gamma V^*(24)\} \\
V^*(24) = \max_{a\in\{0,1\}}\{\gamma V^*(23), 1 + \gamma V^*(24)\}
\end{cases}
$$

La résolution « à la main » peut se faire en se disant que dans l'état 24, le mieux que l'on puisse faire est d'y rester puisque cela rapporte +1 à chaque itération et que l'on ne gagne rien si l'on atteint un autre état que l'état 24. Donc, la dernière équation devient :

$$
\begin{aligned}
V^*(24) &= 1 + \gamma V^*(24) \\
&\text{soit} \\
V^*(24) &= \frac{1}{1-\gamma}
\end{aligned}
$$

Par le même raisonnement, on établit que dans l'état 23, le mieux est d'aller dans l'état 24. Donc :

$$
\begin{aligned}
V^*(23) &= 1 + \gamma V^*(24) \\
V^*(23) &= 1 + \frac{1}{1-\gamma} \\
V^*(23) &= \frac{1}{1-\gamma}
\end{aligned}
$$

Pour les états 18 et 22, le mieux est d'aller vers 23. Donc :

$$
\begin{aligned}
V^*(18) = V^*(22) &= \gamma V^*(23) \\
V^*(18) = V^*(22) &= \frac{\gamma}{1-\gamma}
\end{aligned}
$$

et ainsi de suite, on obtient :

$$
\begin{aligned}
V^*(15) = V^*(21) &= \frac{\gamma^2}{1-\gamma} \\
V^*(13) = V^*(17) = V^*(20) &= \frac{\gamma^3}{1-\gamma} \\
V^*(9) = V^*(12) = V^*(19) &= \frac{\gamma^4}{1-\gamma} \\
V^*(7) = V^*(16) &= \frac{\gamma^5}{1-\gamma} \\
V^*(4) = V^*(14) &= \frac{\gamma^6}{1-\gamma} \\
V^*(3) = V^*(10) &= \frac{\gamma^7}{1-\gamma} \\
V^*(2) = V^*(8) = V^*(11) &= \frac{\gamma^8}{1-\gamma} \\
V^*(5) = V^*(6) &= \frac{\gamma^9}{1-\gamma} \\
V^*(1) &= \frac{\gamma^{10}}{1-\gamma}
\end{aligned}
$$

5.4 Apprentissage par itération de la police

L'algorithme d'itération valeur va tout d'abord calculer la fonction valeur V^* et donc obtenir ce que l'on a indiqué au paragraphe précédent. De là, il déduit la politique optimale par la règle :

$$\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V(s')]$$

soit :

$$
\begin{cases}
\pi^*(24) = 0 \\
\pi^*(11) = 1 \\
\pi^*(2) = \pi^*(3) = \pi^*(12) = \pi^*(19) = \pi^*(20) = \pi^*(21) = \pi^*(22) = \pi^*(23) = 2 \\
\pi^*(6) = 3 \\
\pi^*(1) = \pi^*(5) = \pi^*(8) = \pi^*(10) = \pi^*(14) = \pi^*(16) = 4 \\
\pi^*(4) = \pi^*(7) = \pi^*(9) = \pi^*(13) = \pi^*(15) = \pi^*(17) = \pi^*(18) = 4
\end{cases}
$$

5.5 Apprentissage par Q-Learning

Références

D. Aberdeen. A (revised) survey of approximate methods for solving partially observable markov decision processes, 2003. `http://users.rsise.anu.edu.au/~daa/files/pomdpreview.pdf`. (Cité à la page 19.)

A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72 :81–138, 1995. http://citeseer.ist.psu.edu/barto95learning.html. (Cité à la page 9.)

D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, 1996. (Cité à la page 3.)

A.R. Cassandra. Background on solving pomdps, 1999. http://www.cs.brown.edu/research/ai/pomdp/tutorial. (Cité à la page 20.)

R. Coulom. *Apprentissage par renforcement utilisant des réseaux de neurones, avec des applications au contrôle moteur.* PhD thesis, INPG, 2002. http://remi.coulom.free.fr/Publications/Thesis.pdf. (Cité aux pages 17 et 19.)

Y. Le Cun. Efficient backprop. In G.B. Orr and K.R. Müller, editors, *Neural networks : tricks of the trade.* Springer-Verlag, 1998. http://citeseer.ist.psu.edu/lecun98efficient.html. (Cité à la page 17.)

A. Doucet, N. de Freitas, and N.J. Gordon. *Sequential Monte Carlo Methods in Practice.* Springer, 2000. (Cité à la page 20.)

M. Ginsberg. GIB : Steps toward an expert-level bridge-playing program. In *Proc. IJCAU*, 1999. http://citeseer.ist.psu.edu/ginsberg99gib.html. (Cité à la page 20.)

R. Munos. A study of reinforcement learning in the continuous case by the means of viscosity solutions. *Machine Learning*, 2000. http://www.cmap.polytechnique.fr/~munos/papers/mlj.ps. (Cité à la page 18.)

R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 2001. http://www.cmap.polytechnique.fr/~munos/papers/MLJ2001.ps.gz. (Cité à la page 19.)

K.P. Murphy. A survey of POMDP solution techniques, 2000. http://citeseer.nj.nec.com/murphy00survey.html. (Cité à la page 19.)

R. Neuneier and H.G. Zimmermann. How to train neural networks. In G.B. Orr and K.R. Müller, editors, *Neural networks : tricks of the trade*, pages 373–423. Springer-Verlag, 1998. http://citeseer.ist.psu.edu/lecun98efficient.html. (Cité à la page 17.)

N.N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proc. of the 9^{th} International Conf. on Artificial Neural Networks*, 1999. http://www.inf.ethz.ch/~schraudo/pubs/smd.pdf. (Cité à la page 17.)

Singh, Jaakola, and Jordan. Learning without state-estimation in POMDP. In *Proc. ICML*, 1994. http://www.eecs.umich.edu/~baveja/Papers/ML94.pdf. (Cité à la page 20.)

R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs : a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112 :181–211, 1999. http://citeseer.ist.psu.edu/sutton98between.html. (Cité à la page 17.)

G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38 :58–68, 1995. http://www.research.ibm.com/massive/tdl.html. (Cité à la page 17.)

S.D. Whitehead and L.J. Lin. Reinforcement learning in non-markov environments. *Artificial Intelligence*, 73, 1995. http://citeseer.ist.psu.edu/whitehead92reinforcement.html. (Cité à la page 20.)

Table des matières