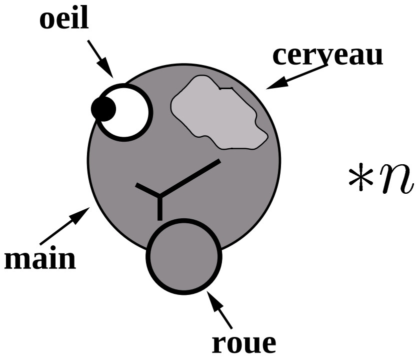
Laboratoire lorrain de recherche en informatique et ses applications

Apprentissage par renforcement dans un système multi-agents



MÉMOIRE

soutenu le 11 juillet 2000

pour l'obtention du

Diplôme d'études approfondies

(Spécialité Informatique)

par

Olivier Buffet

Composition du jury

Jean-Paul Haton
Didier Galmiche
Michaël Rusinowitch
Dominique Méry
François Charpillet
Anne Boyer
Frédéric Alexandre

Univ. Henri Poincaré - Nancy I, Univ. de Nancy II, Univ. de Metz, INPL

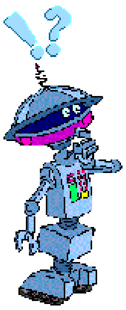
Mis en page avec la classe thloria.

Résumé

Longtemps l'Intelligence Artificielle s'est attachée à faire effectuer par un unique agent des tâches plus ou moins complexes. Les méthodes exactes, cherchant par des méthodes systématiques à résoudre un problème, sont hélas souvent inexploitables quand des contraintes de taille mémoire ou de temps-réel entrent en jeu. On se propose alors de trouver plutôt des solutions optimales ou sous-optimales, par des méthodes approchées de planification ou d'apprentissage qui permettent des convergences beaucoup plus rapides (les processus décisionnels de Markov dont il sera sujet en font partie).

Mais les capacités d'un unique agent peuvent rester insuffisantes, au moins au niveau de ses moyens matériels. L'idée est que certains problèmes sont résolus non par des individus séparément, mais par des groupes d'individus. Même de simples insectes, du fait de leur organisation, peuvent ainsi effectuer des prouesses.

Ce stage de DEA vise à faire une étude des travaux existants dans le domaine de l'apprentissage au sein de systèmes mono- et multi-agents. Le cadre multi-agents pose des problèmes nouveaux aux méthodes d'apprentissage par renforcement connues, les agents devant coordonner leurs actions. Le stage a été en outre l'occasion de mettre en œuvre des algorithmes classiques sur quelques exemples simples.



Remerciements

François, Alain, Gaston
et toute l'équipe Maia.

Table des matières

Chapitre 1

Introduction

1.1 Sujet

Comme nous allons le voir tout à l'heure, les systèmes multi-agents (SMA) permettent de résoudre certains problèmes en utilisant plusieurs agents d'"intelligence" plutôt réduite et donc de faible complexité. L'émergence d'un comportement organisé au sein du groupe d'agents permet d'améliorer les capacités de ces agents.

D'autre part, un agent voué à résoudre un problème ou accomplir une tâche dans un cadre donné connu peut être conçu de façon spécifique pour son usage. Mais concevoir pour chaque usage un nouvel agent s'avère fastidieux. Il peut même suffire de quelques changements dans son environnement pour qu'un robot devienne inopérant. Souhaitant avoir des agents capables de s'adapter à une nouvelle situation, on cherche à leur donner des capacités d'apprentissage du comportement à adopter, ou d'un modèle du monde pour pouvoir ensuite planifier leurs actions.

Le problème posé est donc celui d'un ensemble d'agents ayant un but commun, n'ayant aucune idée de la façon d'atteindre ce but, et devant apprendre le comportement à adopter pour résoudre ce problème. Notre intérêt se porte vers les situations faisant appel à la coopération des agents. Il faudrait trouver comment des agents peuvent apprendre ensemble à coopérer, sans que cet apprentissage soit centralisé bien entendu. On peut donner l'exemple d'une équipe de football ne sachant pas jouer et apprenant à organiser leur jeu, à faire des passes et à fixer les rôles de chaque joueur.

Ce "mélange" de SMAs et d'apprentissage par renforcement doit permettre avec des agents assez simples de résoudre des problèmes complexes en s'adaptant à ceux-ci. On essaye de cumuler les avantages des deux outils pour faire un système souple et efficace.

Si les méthodes multi-agents et l'apprentissage par renforcement sont des outils courants dans divers domaines, leur combinaison est par contre beaucoup moins courante, du fait des problèmes qu'elle soulève. En effet, le comportement à adopter en présence d'autres agents va dépendre des dits agents. La difficulté vient du fait que chacun cherche à s'adapter aux autres, même si c'est aussi la clef de la coopération des agents. On sort de ce fait du cadre de l'apprentissage par renforcement classique, ce qui appelle à chercher de nouvelles méthodes d'apprentissage plus adaptées.

1.2 Organisation du rapport

Ce rapport présente en premier lieu les systèmes multi-agents dans leurs principes et leurs apports à la résolution de certains problèmes. Les SMAs réapparaîtront plus tard dans l'extension de l'apprentissage par renforcement au cadre multi-agents.

La suite expose les modèles de Markov utilisés pour l'apprentissage, ces modèles étant adaptés à la représentation des problèmes posés puisqu'ils font apparaître à la fois les états de l'environnement, les actions possibles pour un agent, et les transitions entre états dues aux actions. L'apprentissage lui-même est ensuite présenté, en abordant plus particulièrement une méthode courante : le Q-learning.

Une dernière partie s'intéresse à l'apprentissage dans le cadre d'un système multi-agents, présentant les problèmes nouveaux qui apparaissent, ainsi qu'un exemple assez simple. Le rapport se conclue sur différents points qui mériteraient d'être approfondis pour améliorer l'apprentissage coopérant.

Chapitre 2

Systèmes Multi-Agents

2.1 Introduction

Les **systèmes multi-agents** sont un outil de résolution de problèmes dans le domaine de l'intelligence artificielle. L'idée de départ est que certains problèmes peuvent être efficacement résolus par un ensemble d'agents plutôt que par un seul, un comportement intelligent **émergeant** de la combinaison de comportements simples. On peut donc utiliser des agents de faible complexité pour des problèmes a priori difficiles. Si différents éléments du problème nécessitent physiquement l'intervention de plusieurs agents, la multiplicité des agents peut s'avérer d'autant plus utile.

Dans un SMA, chaque agent n'a souvent que des informations incomplètes ou des capacités insuffisantes à la résolution du problème posé. De plus il n'y a pas de système de contrôle global : chacun prend ses propres décisions, éventuellement après s'être entendu avec ses congénères. Ces principes permettent d'utiliser des agents assez simples pour ne pas être trop spécialisés, et donc être réutilisables.

Dans la nature, les systèmes multi-agents les plus visibles sont les sociétés animales : fourmis, hommes, moutons, voire les associations animales ou végétales qui, par leur action commune, créent un comportement "de groupe" ou plus simplement coopératif. On s'intéresse donc à voir comment d'un ensemble d'agents plutôt simples émerge un comportement intelligent.

Un système multi-agents est considéré dans son **environnement**. Différents éléments sont examinés dans le comportement d'un SMA : les **interactions** avec l'environnement, la **communication** entre agents, et l'**organisation** de la société.

2.2 Types d'agents

On peut distinguer une multitude de types d'agents, les critères de distinction pouvant être variés. Commençons par donner une définition de ce qu'est un agent en général.

2.2.1 Agent : définition

Il n'y a pas de réel consensus sur la définition d'un SMA, souvent confondu avec les problèmes d'intelligence artificielle distribuée, ou la définition d'un agent. On peut pour référence donner la définition de [Ferber 95] :

On appelle agent une entité physique ou virtuelle :

- a. qui est capable d'agir dans un environnement,
- b. qui peut communiquer directement avec d'autres agents,
- c. qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),
- d. qui possède des ressources propres,
- e. qui est capable de percevoir (mais de manière limitée) son environnement,
- f. qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- g. qui possède des compétences et offre des services,
- h. qui peut éventuellement se reproduire,
- i. dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Cette définition, si elle n'est pas universelle, a l'avantage de présenter une grande partie des caractéristiques dont peut être doté un agent.

2.2.2 Agent réactif ou cognitif ?

On peut classer les agents selon divers critères. La séparation en deux catégories la plus courante est la suivante :

Les agents **cognitifs** sont les plus évolués. Ils implémentent en général des algorithmes de type planification pour décider de leurs actions.

Les agents **réactifs** à l'inverse cherchent la simplicité. Leurs actions sont provoquées (plus que choisies) par des réactions réflexes aux changements de leur environnement.

Il faut noter qu'à force de vouloir utiliser des agents de plus en plus évolués, on risque de se porter plus sur les capacités d'un agent que d'un système multi-agents. Ces deux classes mettent en valeurs de différentes façons les principes d'interaction, communication et organisation (plus ou moins contrôlées ou émergeantes).

2.3 Des exemples

Le terme "système multi-agents" étant encore mal défini car il regroupe diverses idées, quelques exemples peuvent préciser utilement ce qu'est un SMA.

2.3.1 Les fourmis

Le premier exemple de système multi-agents que nous donnerons est assez courant : les fourmis. Ces charmants insectes ont la capacité de trouver le chemin le plus court entre deux points et ce, sans avoir a priori de notion de distance (en tout cas elles n'en ont nul besoin).

Les données du problème de base dans le cas des fourmis est que le chemin cherché doit relier la fourmilière à une source de nourriture, deux chemins de longueurs différentes étant envisageables, mais les fourmis n'ayant aucune mesure ni notion de distance.

Les fourmis partant de la fourmilière vont prendre indifféremment l'un ou l'autre chemin, à l'aller comme au retour. Sauf que, au retour, ces dernières déposeront sur le chemin choisi une trace chimique (des phéromones), qui aura pour effet d'inciter les autres fourmis à suivre la piste marquée. Et les dépôts successifs auront pour effet de renforcer ce marquage. Or, le chemin le plus court étant aussi le plus rapide, les fourmis le choisissant auront plus vite fait de le marquer fortement que les fourmis passant sur l'autre voie. Les fourmis étant d'autant plus attirées par un chemin que celui-ci est marqué de phéromones, le phénomène va en s'amplifiant.

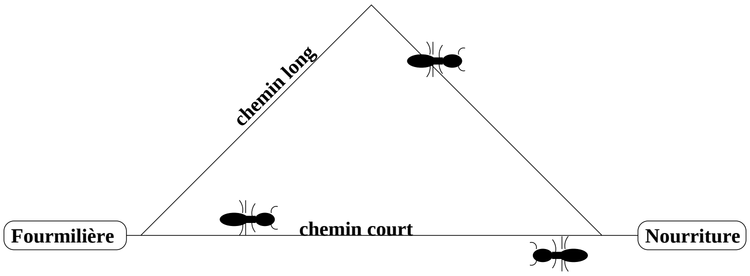


FIG. 2.1 – recherche du chemin le plus court chez les fourmis

2.3.2 Le football

Pour donner rapidement un autre exemple de SMA "naturel", on peut citer l'équipe de football, dans laquelle les joueurs coopèrent pour atteindre un but commun, chacun se spécialisant et l'ensemble formant un tout organisé. La *Robocup*, concours de robotique réelle ou simulée selon les catégories, a donné lieu à un certains nombre de travaux sur le sujet [1].

Cet exemple n'illustre pas seulement un système à agents plus cognitifs que les fourmis. Le football montre aussi l'aspect compétitif entre agents si l'on considère l'opposition entre les équipes. Une version simplifiée du jeu, opposant deux joueurs, est utilisée dans [Littman 94] pour illustrer le propos de Littman.

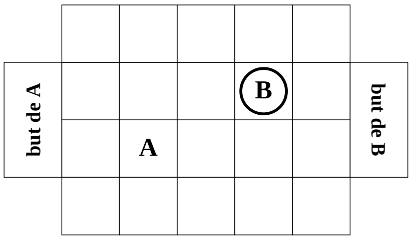


FIG. 2.2 – football à deux joueurs (B a la balle)

2.4 Nos agents

Dans le cadre de l'apprentissage, nous n'allons certes pas utiliser des agents purement réactifs. L'objectif est de leur faire acquérir une certaine "intelligence". Néanmoins, le thème abordé étant celui des systèmes multi-agents, un comportement assez simple doit suffire pour mettre en valeur un travail de groupe émergeant.

Pour décrire rapidement un agent tel qu'il apparaîtra par la suite, nous dirons : qu'il agit dans/sur son environnement en fonction de la situation qu'il perçoit (sans avoir de mémoire des événements passés), qu'il ne communique pas de façon explicite, ne peut se reproduire et qu'il cherche à maximiser les récompenses qu'il perçoit.

On peut d'ailleurs rappeler ce qui distingue la notion d'objet de certains langages de la notion d'agent : *"Les objets travaillent gratuitement, les agents travaillent pour de l'argent."* ([Jennings 98] [2]).

[1] `http ://www.robocup.org`

[2] L'article de Jennings, Sycara et Wooldridge [Jennings 98] constitue d'ailleurs une présentation assez avancée du domaine de l'agent et du multi-agents. L'ouvrage [Ferber 95] a, pour sa part, une visée beaucoup plus vaste sur le sujet des SMAs.

Chapitre 3

Markov

3.1 Introduction

Le problème de l'apprentissage, comme de la planification, est de contrôler le comportement d'un agent. On va donc avoir besoin de modéliser cet agent dans son environnement. Le modèle utilisé considère d'une part les **états** possibles du **système (=agent+environnement)**, et d'autres part les **actions** possibles de l'agent dans son environnement.

L'environnement étant incertain, il vient facilement à l'idée de faire appel à une modélisation stochastique du système. L'outil de base qui va ainsi nous servir à l'apprentissage et au fonctionnement général des agents est le **processus de décision Markovien (MDP)** et ses dérivés. Il s'agit là d'une classe de **modèles de Markov**, eux-même appartenant aux **processus stochastiques**.

Processus stochastiques
$\downarrow$
Modèles de Markov
$\downarrow$
Processus de décision de Markov

FIG. 3.1 – évolution des modèles

L'idée de départ d'un processus stochastique est d'utiliser comme représentation un graphe dans lequel les nœuds sont les états possibles du système et les arcs (orientés et annotés) donnent les probabilités de passage d'un état à un autre.

3.1.1 Processus stochastique

Un processus stochastique est une famille $(X(k), k \in K)$ de variables aléatoires définies sur un espace Ω. K est un ensemble d'indices. $X(k)$ pourra représenter l'état d'un système ou une observation à l'instant k. $P(X = x)$ est la probabilité d'une réalisation x de la variable aléatoire X. Pour une observation à l'instant k par exemple, $P(\Omega(k) = \omega)$ est la probabilité d'avoir ω comme observation à l'instant k.

Le processus est dit **discret** si les variables X sont en nombre fini ou dénombrable : $K = \{1, ..., T\}$. Il s'agit d'une discrétisation du temps. Dans la suite, les processus utilisés sont tous discrets sauf précision contraire.

Evolution du processus
Le processus a une évolution chronologique : démarrage à l'instant $t = 0 \to X(0)$, puis passage d'un état au suivant : $X(0) \to X(1) \to ...X(T)$. On veut obtenir $X(i)_{i=1..T}$ $\forall i \leq T$ avec $P(X(0), X(1), ..., X(i))$. P est définie de proche en proche (on note $X(i) = X_i$) :

$$
\begin{aligned}
& P(X_0, X_1..., X_T) \\
& = P(X_0, X_1..., X_{T-1}) * P(X_T|X_0, X_1..., X_{T-1}) \\
& = P(X_0, X_1..., X_{T-2}) * P(X_{T-1}|X_0, X_1..., X_{T-2}) * P(X_T|X_0, X_1..., X_{T-1}) \\
& = P(X_0) * P(X_1|X_0) * P(X_2|X_0, X_1) * P(X_T|X_0, X_1, ..., X_{T-1})
\end{aligned}
$$

3.1.2 Modèle de Markov

Dans le modèle de Markov, à l'état t, $X(t)$ ne dépend que des n états précédents (mémoire du processus dans un modèle ici d'ordre n). A l'ordre 1, comme c'est souvent le cas (et comme ce sera le cas par la suite) :

$$P(X(t) = s_i|X(t-1) = s_j, X(t-2) = s_k, ...) = P(X(t) = s_i|X(t-1) = s_j) \tag{3.1}$$

Le modèle de Markov est dit stationnaire si la probabilité de transition entre deux états ne varie pas avec le temps :

$$\forall t, k\ P(X(t) = s_i|X(t-1) = s_j) = P(X(t+k) = s_i|X(t+k-1) = s_j) \tag{3.2}$$

Ceci conduit à définir une matrice de probabilités de transitions $A = [a_{ij}]$ où les a_{ij} sont définis par :

$$a_{ij} = P(X(t) = s_j|X(t-1) = s_i)\ 1 \leq i \leq N, 1 \leq j \leq N, \tag{3.3}$$

la matrice A étant stochastique, c'est à dire vérifie :

$$\forall i, j\ a_{ij} \geq 0,\ \forall i \sum_{j=1}^{N} a_{ij} = 1 \tag{3.4}$$

où N est le nombre d'états du système.

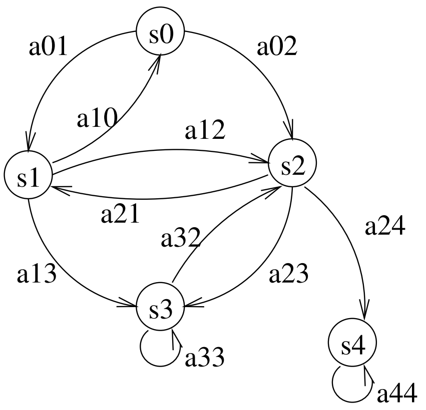


FIG. 3.2 – graphe d'un modèle de Markov
a_{ij} = probabilité de transition de $X(i) \to X(j)$.

3.2 Introduction aux MDPs

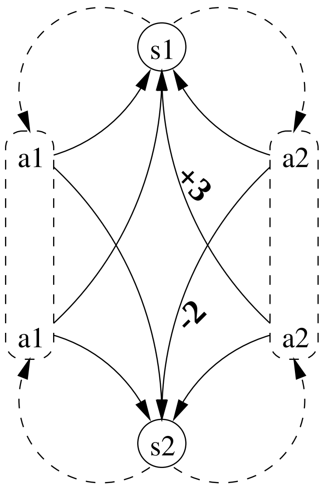
Les agents vont être plongés dans un monde où, partant d'un état donné, une même action n'aboutit pas toujours au même état. Ceci peut venir du fait que l'état n'est qu'incomplètement connu (on ne connait qu'une observation partielle), ou que des actions simultanées d'autres agents peuvent avoir lieu. Les modèles de Markov permettent de modéliser de telles situations.

Les différentes actions effectuées et les différents états rencontrés amèneront à tout agent un gain ou un coût. Les agents doivent donc chercher à agir au mieux. Leurs **politiques** (l'application qui, à un état du système, associe une action à effectuer) peuvent être diverses selon qu'elles considèrent le plus ou moins long-terme, différentes estimations du gain futur escompté étant possibles.

Un dernier point à noter est qu'un agent peut ne pas connaître au départ les lois qui régissent son environnement. Il lui faudra alors les apprendre plus ou moins directement (apprendre un modèle ou une politique adaptée au modèle méconnu) pour améliorer ses chances de gain.

3.3 Processus de Décision Markoviens (MDP)

Un processus de décision markovien est constitué de :
- un ensemble d'**états** $\mathcal{S}$,
- un ensemble d'**actions** $\mathcal{A}$,
- une fonction de **gain/récompense** $r : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Re)$, et
- une fonction de **transition d'état** $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, où un élément de $\Pi(\mathcal{S})$ est une distribution de probabilité sur l'ensemble $\mathcal{S}$. On écrit $T(s, a, s')$ la probabilité de faire la transition de l'état s à l'état s' en utilisant l'action a.



Processus de décision Markovien à 2 états s_1 et s_2, et deux actions possibles a_1 et a_2. Depuis un état s, l'action a choisie est prise parmi a_1 et a_2. Un état quelconque s' a alors une probabilité d'occurence $T(s, a, s')$. Et le gain de l'agent dépend pour sa part de la distribution stochastique $r(s, a)$.

FIG. 3.3 – exemple de MDP

3.4 Usage

3.4.1 Points de départ

On cherche à maximiser la récompense à plus ou moins long terme. Il faut donc mesurer l'**espérance de gain**, en tenant par exemple de moins en moins compte du futur (confiance moins bonne, grâce au coefficient $\gamma \in [0, 1]$) :

$$E(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{3.5}$$

D'autres méthodes de calcul sont possibles, mais seule la formule précédente sera utilisée par la suite.

Dans chaque état, il faudra donc déterminer l'action optimale à effectuer. On définit ainsi une politique p, application ici déterministe :

$$p : \mathcal{S} \to \mathcal{A} \tag{3.6}$$

3.4.2 Grandeurs utilisées

L'espérance de gain va donc dépendre de la politique choisie. On peut ainsi définir, pour une politique p fixée, l'utilité d'un état :

$$V_p(s_t) = E\left(\sum_{k=0}^{\infty} \gamma^k E[r(S_{t+k}, p(S_{t+k})) \mid (S_t = s)]\right) \tag{3.7}$$

Pour une politique optimale : $V^*(s) = \sup_p V_p(s)$, l'utilité d'une politique optimale peut s'écrire de manière récursive sous la forme de l'équation de Bellman :

$$V^*(s) = \max_{a \in \mathcal{A}} \left(E_{r \times \pi(s,a)}[r(s, a) + \gamma V^*(s')]\right) \tag{3.8}$$

On obtient une politique déterministe optimale en cherchant pour quelle action on obtient ce maximum. Encore faut-il connaître la fonction V^*.

En planification, les algorithmes appelés *Value Iteration* et *Policy Iteration* permettent, en partant d'une fonction V quelconque, de la faire converger vers V^* et donc de trouver une politique optimale.

Une autre grandeur, $Q(s, a)$, peut être utilisée, obtenue à partir de $V(s)$. Elle va fournir la “qualité” de l'action a effectuée dans l'état s :

$$Q(s, a) = E_{r \times \pi(s,a)}[r(s, a) + \gamma V^*(s')] \tag{3.9}$$

Q a été introduite par [Watkins 89] dans un algorithme d'apprentissage par renforcement que nous verrons par la suite.

3.5 Un exemple

Prenons un rat dans un labyrinthe. Ce labyrinthe peut être facilement représenté par un graphe où chaque nœud décrit une position possible du rat. Si le but est d'amener le rat à un point donné, la fonction de gain (ou récompense) associera par exemple une valeur positive de 1 à toute action qui mène au but, 0 sinon.

Si l'on considère que le système de décision connaît exactement la position du rat, ce qui n'est pas nécessairement le cas de notre rongeur, ce système peut donner à l'animal des ordres de déplacement sans que ceux-ci soient suivis avec exactitude. Il peut y avoir une imprécision sur la commande du simple fait que notre sujet préfère les couloirs peints en orange à ceux peints en vert. On aura alors peut-être intérêt à lui demander de suivre un chemin un peu plus long mais qu'il suivra plus sûrement.

Le problème se pose aussi en cas d'environnement dynamique. L'apprentissage d'une politique en continu a un avantage certain si le rat prend goût pour les couloirs verts par exemple.

3.6 Autres modèles

A partir du modèle de Markov sont définies des variantes servant aussi bien en reconnaissance des formes qu'en planification (comme les MDPs). Le but général est de deviner l'état présent ou futur d'un système. Voici quelques modèles Markoviens dérivés :

- **HMM** Dans un Modèle de Markov Caché, l'état du système n'est pas connu. On a par contre une observation qui est liée par des lois probabilistes aux états. On ne peut être sûr d'un état avec une perception du monde extérieur, mais une suite de perceptions peut affiner un jugement. C'est le principal outil de reconnaissance de formes issu des modèles de Markov.

- **MDP** (voir leur présentation détaillée dans ce rapport)

- Les **MMDP** (MDP Multiples) sont une variante des MDPs adaptée au cas des systèmes multi-agents, de même que les **DEC-MDP** (MDP décentralisés) et les **jeux de Markov**. Ces trois modèles sont présentés en annexe.

- **SMDP** Le modèle, dit Semi-Markovien, a pour but d'améliorer la gestion du temps, considérant que le passage dans un état peut être de durée variable (selon des lois stochastiques).

- **POMDP** Mélange de HMMs et MDPs, ces MDPs Partiellement Observables ajoutent l'idée qu'un agent n'a de son environnement qu'une perception partielle, donc qu'il ne connaît qu'une observation et non un état complet.

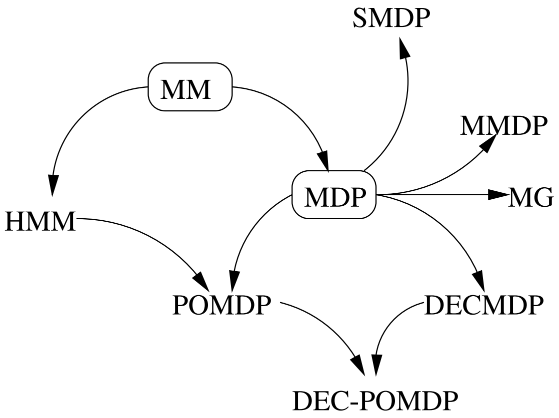


FIG. 3.4 – la famille des MDPs

Chapitre 4

Apprentissage par renforcement et MDPs

4.1 Apprentissage par renforcement

4.1.1 Introduction

Le problème posé est celui d'un agent placé dans un environnement au sein duquel il doit atteindre un but, que ce soit un but final ou un but de "maintien" d'un état. Une première méthode est de planifier son comportement à l'avance, connaissant un modèle suffisamment complet de l'environnement. Comme celui-ci est souvent imprévisible, il pourra y avoir lieu de réviser régulièrement le plan.

Dans le cas où aucun modèle suffisant n'est connu, la planification n'est plus utilisable et il va falloir apprendre par essai-erreur un comportement qui va résoudre le problème posé à l'agent. C'est ainsi que l'on définit l'apprentissage par renforcement. Dans le cadre des processus de décision markoviens, il sera fait usage de techniques statistiques et des méthodes de programmation dynamique.

Différents problèmes se posent dans un tel apprentissage :
- Faut-il apprendre directement un comportement sans connaître de modèle de l'environnement ou à l'inverse apprendre un modèle et en déduire le meilleur comportement possible ?
- Sachant qu'il y a un gain à maximiser, comment doser l'exploitation de l'apprentissage déjà effectué (pour un gain immédiat) et l'exploration des comportements possibles (pour améliorer le gain futur) ? C'est ce que l'on appelle le dilemme exploration/exploitation.
- Selon le problème posé, est-il préférable de considérer un gain à court ou long terme ? (Cette question se pose aussi en planification.)

Note : Pour une présentation plus complète de l'apprentissage par renforcement, on peut se reporter à [Kaelbling 96].

4.1.2 Modèle de l'apprentissage

On retrouve dans les processus de décision markoviens les éléments nécessaires à l'apprentissage par renforcement. En effet, il faut au moins :
- un ensemble discret d'états de l'environnement $\mathcal{S}$,
- un ensemble discret d'actions de l'agent $\mathcal{A}$,
- un ensemble de signaux de renforcement scalaires (dans $[0, 1]$ ou sur $\Re$ par exemple).

La figure 4.1 montre un agent dans son environnement, ainsi que les différentes grandeurs utiles.

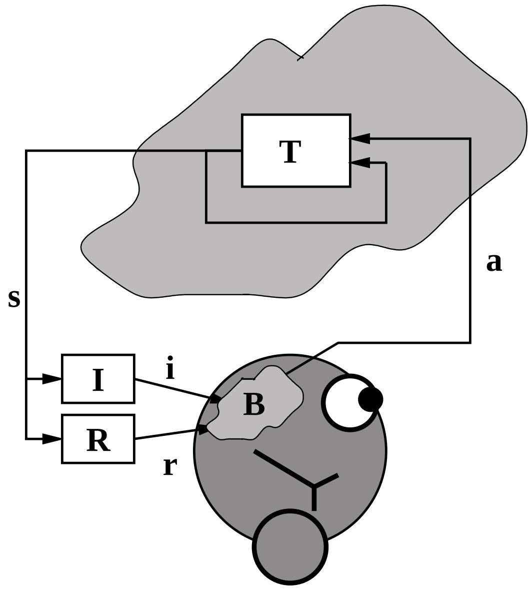


Schéma de principe d'un agent dans son environnement

a : action
s : état
i : entrée (perception)
r : renforcement
T : fonction de transition
(loi régissant l'environnement)
B : comportement

FIG. 4.1 – un agent

Chaque action a va donner lieu à un changement d'état de l'environnement et à un signal de renforcement :

$$(s, a) \rightarrow^T (s', r) \tag{4.1}$$

L'environnement n'est pas a priori déterministe : effectuer une action dans un état peut amener à différents états et produire différents signaux de renforcement comme cela a déjà été évoqué dans la présentation des MDPs.

De plus, l'agent peut ne percevoir qu'une observation incomplète de l'état du système, d'où la transformation $s \rightarrow i$. C'est d'ailleurs en fonction des observations perçues et des récompenses reçues que le comportement B de l'agent va choisir l'action à effectuer.

4.2 Q-learning

4.2.1 Principe

Le Q-learning [Watkins 89] fait partie des méthodes d'apprentissage par renforcement sans modèle. Il s'agit d'apprendre par l'expérience les actions à effectuer en fonction de l'état actuel.

On associe à chaque paire (s, a) (où l'action a est exécutée depuis l'état s), une grandeur Q par laquelle on veut apprendre l'intérêt de l'action a quand on est dans l'état s. $Q(s, a)$ est l'espérance de gain, c'est-à-dire la somme :

– de l'espérance de gain immédiat dû à l'action choisie,
– et des espérances de gain de tous les états, pondérées par la probabilité d'atteindre chacun d'eux depuis l'état s en menant l'action a.

Nous avons vu que, dans le cas d'une politique idéale, Q^* s'écrit :

$$Q^*(s, a) = E_{r \times \pi(s,a)}[r(s, a) + \gamma V^*(s')] \tag{4.2}$$

Mais V^* peut aussi être écrite grâce à Q^* (équation de Bellman) :

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a) \tag{4.3}$$

On obtient ainsi une définition récurrente de Q^* :

$$Q^*(s,a) = E_{r \times \pi(s,a)}[r(s,a) + \gamma \max_{a \in \mathcal{A}} Q^*(s',a)] \tag{4.4}$$

L'algorithme du Q-learning cherche une approximation de Q^* comme point fixe de l'équation précédente, en écrivant :

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t[r_t + \gamma_t * \max_{a' \in \mathcal{A}} Q_t(s_{t+1},a')] \tag{4.5}$$

Cette équation permet de mettre à jour $Q(s_t,a_t)$ en fonction de sa valeur précédente et de l'expérience qui vient d'être vécue.

D'où l'algorithme :

initialiser $Q_0(s,a)$ arbitrairement
choisir un point de départ s_0
tant que la politique n'est pas assez bonne
    choisir a_t en fonction de $Q_t(s_t,.)$
    en retour : s_{t+1} et r_t
    $Q_{t+1}(s_t,a_t) \leftarrow (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t(r_t + \gamma * \max_{a' \in \mathcal{A}} Q_t(s_{t+1},a'))$
    $t \leftarrow t+1$
fin tant que

Il faut alors régler le coefficient α pour fixer progressivement la politique apprise. γ permet pour sa part de moduler l'importance des récompenses escomptées à venir.

Note : Le critère `tant que la politique n'est pas assez bonne` ne sera pas discuter ici, nous avons choisi de laisser un apprentissage en continu indéfiniment.

4.2.2 Implantation

La description faite du Q-learning laisse beaucoup de points indéfinis si l'on veut mettre en application cette méthode. Il faut encore définir comment évolue α, comment est choisie une action, comment faire évoluer le comportement entre exploration et exploitation. Il n'y a pas de méthode idéale, mais les méthodes utilisées dans cet exposé sont présentées ici.

- $\underline{\alpha}$ **α doit être compris entre 0 et 1, et décroître vers 0.** On peut prendre simplement $\alpha = 1/n$ où n est le nombre de fois où l'action a actuelle a été exécutée dans l'état s actuel. Mais la décroissance risque d'être un peu trop rapide. On précède donc cette suite en $1/n$ par une période pendant laquelle α est constant (0.3 par exemple).
- $\underline{\gamma}$ **On doit de même choisir γ constant compris entre 0 et 1.** Pour que les récompenses se diffusent bien, on prend par exemple $\gamma = 0.8$.

4.2.3 Choix des actions

Il reste à choisir les actions à effectuer selon les connaissances acquises. Étant donné un état s, et connaissant $Q(s,a_i)$ pour $i \in [0..n]$, il faut :

– au début favoriser l'exploration en utilisant avec autant de probabilité les n actions possibles,
– au fur et à mesure tendre vers l'utilisation de l'action de plus grande valeur $Q(s,a)$.

Pour cela, il est souvent fait usage du Q-learning Boltzmanien ([Watkins 89] par exemple), où une distribution de Boltzmann est utilisée pour choisir les actions :

$$\omega(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{b \in \mathcal{A}} e^{\frac{Q(s,b)}{T}}} \tag{4.6}$$

La température T doit donc être prise assez élevée au début, pour décroître vers 0 quand l'apprentissage sera terminé.

exploration		exploitation
T élevée	$\rightarrow$	$T \simeq 0$

4.2.4 Essai : Tri de liste (1 agent)

Un premier problème qui a servi pour les premiers essais de Q-learning est un problème de tri de liste. Il s'agit d'apprendre à un agent, pour chaque configuration de la liste à trier, quelle transposition effectuer pour ordonner cette liste. Pour limiter le nombre de coups possibles, on ne permet à l'agent que de transposer deux éléments consécutifs de la liste (ainsi que les deux éléments de bout de liste). Une liste contiendra juste trois ou quatre éléments.

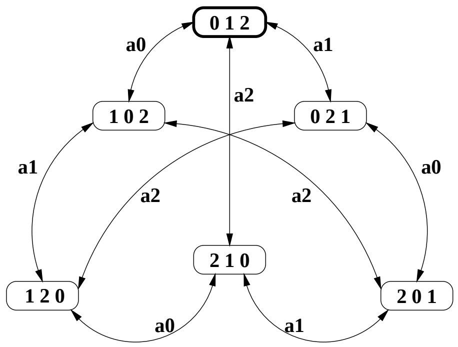


FIG. 4.2 – le graphe d'états de la liste

constantes	valeurs
γ	0.8
$1/\alpha_{max}$	10000
$t_{initial}$	1.0
t_{final}	0.01
t_{k+1}/t_k	0.999

FIG. 4.3 – quelques unes des constantes définies

s	a0	a1	a2	liste
0	33	33	33	0 1 2
1	0	99	0	0 2 1
2	99	0	0	1 0 2
3	37	21	41	1 2 0
4	31	27	40	2 0 1
5	0	0	99	2 1 0

$a_i(s)$: probabilité d'effectuer la transposition (e_i, e_{i+1}) depuis l'état s.

FIG. 4.4 – table apprise après 5000 coups.

La table présentée ci-dessus est obtenue à partir de la table $Q(s,a)$ apprise, et exprime directement la politique obtenue. Il faut tout de suite noter que l'état 0 étant l'état de victoire, on n'a jamais à choisir d'action depuis cet état (d'où $Q(0,a)$ nul pour tout a, donc des probabilités toutes égales à 33%).

Pour les autres coefficients de la table, on peut noter que dans la situation actuelle, si une transposition mène à la liste ordonnée, il est quasiment sûr qu'elle sera exécutée. Dans les états 3 et 4 (qui sont strictement équivalents), les trois transpositions possibles ont autant d'intérêt, d'où une politique qui ne favorise pas particulièrement une action par rapport aux autres.

```
1 2 0 : 3      s=3 -> a0
+-+

2 1 0 : 5      s=5 -> a2
+---+

0 1 2 : 0      s=0 -> fin
```

Chaque ligne présente une configuration de liste et le numéro d'état correspondant.

La table d'apprentissage utilisée est celle de la figure 4.4.

FIG. 4.5 – exemple de tri de liste 1.

4.2.5 Essai : Tri de liste (2 agents)

Le tri de liste va permettre de voir un premier exemple d'apprentissage coopératif, deux agents apprenants à remettre ensemble de l'ordre dans la liste.

La première tentative faite était de mettre deux agents identiques à celui vu précédemment, les deux agents faisant chacun une transposition (éventuellement la même) à chaque étape. Hélas, on n'a qu'une chance sur deux qu'une liste puisse être triée en un nombre pair de transpositions, ce qui a coupé court à cette première méthode.

Le comportement adopté pour nos agents est assez simple : chacun choisit un élément de la liste et l'échange avec l'élément choisi par l'autre agent. Il n'y a qu'une transposition faite à chaque étape, et cette fois-ci toutes les transpositions sont admises.

L'apprentissage effectué est un Q-learning simple, bien que les conditions de convergence de cet algorithme ne soient pas respectées (voir les explications dans le chapitre suivant).

s	aa0	aa1	aa2	ab0	ab1	ab2	liste
0	33	33	33	33	33	33	0 1 2
1	0	99	0	0	0	99	0 2 1
2	99	0	0	0	99	0	1 0 2
3	0	99	0	0	0	99	1 2 0
4	0	0	99	99	0	0	2 0 1
5	99	0	0	0	0	99	2 1 0

$aa_i(s)$: probabilité pour l'agent A d'effectuer la transposition (e_i, e_{i+1}) depuis l'état s.
$ab_i(s)$: idem pour l'agent B.

FIG. 4.6 – table apprise après 5000 coups.

On voit qu'au cours de leur apprentissage les agents se sont spécialisés, chacun ayant, pour une configuration de la liste, le rôle de prendre un élément bien précis pour effectuer la transposition adequat. Evidemment, la spécialisation est tout à fait différente à chaque fois que l'on relance le processus.

```
2 0 1 : 4      s=4 -> aa2 et ab0
b---a

1 0 2 : 2      s=2 -> aa0 et ab1
a-b

0 1 2 : 0      s=0 -> fin
```

Chaque ligne présente une configuration de liste et le numéro d'état correspondant.

La table d'apprentissage utilisée est celle de la figure 4.6.

FIG. 4.7 – exemple de tri de liste 2.

4.3 POMDPs...

4.3.1 ...les cas réels

Les algorithmes et résultats vus jusqu'à présent concernaient des problèmes markoviens, dans lesquels par exemple on connaît à chaque instant l'état exact du système. Cette connaissance étant souvent inaccessible dans les cas réels, on se retrouve avec un problème où sont seulement connues des observations de l'état du système. Il faut alors introduire les Processus de Décision Markoviens Partiellement Observables (POMDP) pour formaliser ces nouveaux problèmes.

4.3.2 Définition

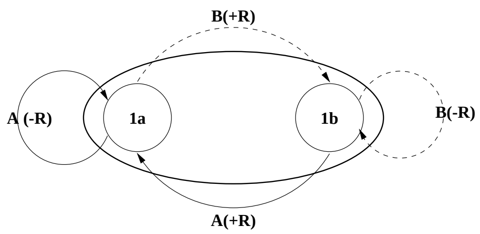
Note : Sont notés en gras les éléments nouveaux par rapport à un MDP.

- un ensemble d'états $\mathcal{S}$,
- un ensemble d'actions $\mathcal{A}$,
- **un ensemble d'observations** Ω ,
- une fonction de gain $r : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Re)$,
- **une application** $\mathcal{O} : \mathcal{S} \rightarrow \Pi(\Omega)$ **qui associe à chaque état** s **une distribution de probabilités sur l'espace des observations** , et
- une fonction de transition d'état $T : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$, où un élément de $\Pi(\mathcal{S})$ est une distribution de probabilité sur l'ensemble $\mathcal{S}$. On écrit $T(s, a, s')$ la probabilité de faire la transition de l'état s à l'état s' en utilisant l'action a.

On n'a plus connaissance, dans un POMDP, de l'état réel du système, mais on a accès à une observation de cet état.

4.3.3 Le problème

[Singh 94] montre que les résultats connus sur la convergence des fonctions V et Q par les algorithmes de planification ou d'apprentissage déjà évoqués (policy iteration et Q-learning par exemple) ne sont plus applicables quand on sort du cadre des MDPs. Un exemple simple de Singh, Jaakkola et Jordan met en évidence le type de problèmes rencontrés :



Ce POMDP a deux états $1a$ et $1b$, pour une seule observation 1 (représentée par l'ellipse encadrant les deux états), et deux actions possibles A et B, dont les récompenses selon l'état peuvent être $+R$ ou $-R$. Sachant que l'on voit l'observation 1, on ne peut déterminer s'il faut choisir l'action A ou B pour obtenir une récompense positive. Il n'existe donc pas de politique déterministe optimale.

FIG. 4.8 – l'exemple problématique

Les POMDPs vont nécessiter l'usage de politiques stochastiques et non plus déterministes comme c'était le cas pour les MDPs. Des méthodes sont rapidement décrites dans le chapitre 4.3.5.

4.3.4 Un exemple

Si l'exemple du rat dans un labyrinthe, rat guidé par un œil extérieur, vous a semblé peu réaliste, nous pouvons y remédier. Il suffit d'imaginer être en présence d'un rat autonome, sans mémoire des endroits où il est passé. Il est possible qu'il puisse se forger une politique stochastique (qui n'associe pas systématiquement la même action à une observation) qui lui permette de trouver son chemin. Mais cela reste assez difficile à concevoir.

Pour un problème partiellement observable comme celui-là, même avoir de la mémoire ne résout pas tout. Mais si l'on connaît le MDP sous-jacent qu'est le véritable labyrinthe, l'historique des observations peut largement améliorer la situation décrite dans le premier paragraphe, mais au prix de lourdes manipulations.

On préfère, quand cela est possible, être face à un problème qui soit "presque" un vrai MDP. C'est-à-dire un POMDP dont une observation correspond à quelques états proches du véritable MDP sous-jacent. L'exemple principal développé au cours de ce rapport se classe dans cette catégorie pour laquelle on va essayer des algorithmes issus des MDPs classiques.

4.3.5 Utilisation des POMDPs

La situation la plus confortable avec des POMDPs est celle où le modèle est connu (processus de transition et d'observation). On peut alors estimer, d'après l'historique des observations successives, dans quels états on peut se trouver. L'ensemble des états réels possibles, pondérés par leurs probabilités respectives, constitue ce que l'on appelle un *"belief state"* du système (ce qui pourrait être traduit par "états supposés"). Parmi les algorithmes faisant de telles estimations, on citera l'algorithme *witness* [Littman 94-b].

Si le POMDP est de modèle inconnu, des méthodes issues du domaine des HMMs (modèles de Markov cachés utilisés en reconnaissance des formes) existent, mais restent difficiles d'utilisation dans des cas réels. D'autres algorithmes spécifiques ont été essayés, tels que l'*algorithme du Lion* de Whitehead et le *Const-Sensitive Q-Learning* de Tan.

Pour une présentation plus complète sur le sujet, voir [Dutech 99].

Dans le cadre de ce DEA, l'approche choisie est plus simple. Il s'agit de considérer les observations obtenues comme étant les états d'un MDP, et donc d'utiliser un algorithme de Q-learning classique. Sous réserve que les observations reflètent bien les états possibles du système, et que la politique utilisée soit stochastique et non plus déterministe, on peut obtenir par apprentissage une politique assez satisfaisante et ne générant pas de blocages (comme cela aurait été le cas avec une politique déterministe).

4.3.6 Essai : Roshambo

Le jeu à deux joueurs “pierre-papier-ciseaux” n'est pas exactement un POMDP. Il pose néanmoins des problèmes de convergence proches de ceux des POMDPs, dans la mesure où la politique obtenue doit être stochastique. Dans ce jeu, chaque agent (joueur) doit choisir l'un des trois objets pierre, papier et ciseaux. Les choix faits sont ensuite confrontés pour déterminer le gagnant sachant que pierre bat ciseaux, ciseaux bat papier, et papier bat pierre. Le MDP associé est un jeu de matrice : il n'a qu'un seul état. Les trois actions possibles sont les choix des trois objets.

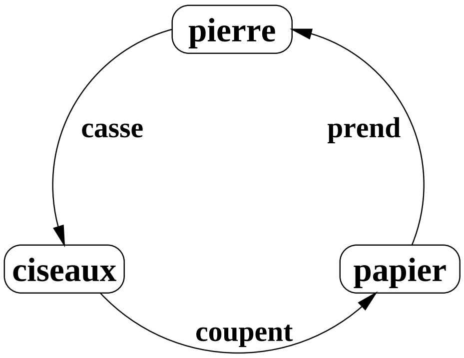


FIG. 4.9 – le jeu le plus bête

Sur ce jeu :
`http ://www.worldrps.com/`
`http ://www.cs.ualberta.ca/~darse/rsbpc.html`

Deux situations principales se présentent :

[2 agents apprenants]

En prenant deux agents apprenant par Q-learning, tous deux en arrivent à des politiques jouant équitablement les trois coups au hasard. Assez logiquement, si l'un des deux joueurs favorisait certains coups, l'autre en profiterait pour jouer en conséquence.

	pierre	papier	ciseaux
$Q_A(a)$	-0.0048	0.0038	-0.0014
Action de A	33218	33561	33221
$Q_B(a)$	-0.0022	-0.0008	0.0047
Action de B	33352	33508	33140

Ci-contre, une table présentant les tables Q pour les joueurs A et B après 100000 parties, et le nombre d'occurences des différents coups.

Les deux joueurs sont à peu près à égalité avec 32092 parties gagnées par A contre 32303 pour B.

[1 seul agent apprenant]

Comme attendu donc, l'apprentissage prend son intérêt si l'on fixe le comportement de jeu de l'un des joueurs (A, qui use plus du papier que de la pierre et très peu des ciseaux), alors que l'autre (B) apprend en continu. Le second adapte sa politique de façon à gagner dans la majorité des parties.

	pierre	papier	ciseaux
$Q_A(a)$	0	1	-1
Action de A	24473	66491	9036
$Q_B(a)$	-0.471	0.262	0.526
Action de B	1127	6295	92578

Ci-contre, une table présentant les tables Q pour les joueurs A et B après 100000 parties, et le nombre d'occurences des différents coups.

B bat ici A par 63290 parties à 24001. Evidemment, contre un joueur ayant un tel défaut, il aurait été préférable d'avoir une politique déterministe...

Chapitre 5

Apprentissage par renforcement dans un système multi-agents

5.1 Introduction

Les systèmes multi-agents ayant été présentés, ainsi que l'apprentissage par renforcement dans le cadre du Q-learning, nous pouvons enfin aborder la problématique présentée en introduction : Comment des agents peuvent-ils apprendre un comportement coopératif pour atteindre un but commun ? Le problème se pose aussi si les buts sont distincts mais peuvent être atteints par un comportement de groupe. Les questions d'apprentissage compétitif où les agents ont des buts plutôt opposés sont un cas qui nous intéressera moins.

5.2 Problème par rapport aux MDPs

Pour l'instant, seuls des environnements stationnaires (dont les transitions d'états sont de probabilités invariantes dans le temps) ont été considérés. Cette condition de stationnarité permet la convergence du Q-learning par exemple vers une politique déterministe optimale dans le cas de MDPs.

Or le fait de se placer dans le cadre d'un système multi-agents où les agents apprennent, comme c'est notre cas, va nous faire sortir du cadre stationnaire courant. Supposons, comme le montre la figure 5.1, un problème à deux agents A et B situés dans un même environnement, les deux agents apprenant. Comme B est en plein apprentissage, son comportement évolue. Or B est un élément de l'environnement si l'on prend du point de vue de A. Donc cette évolution du comportement de B se traduit pour A par une loi de transition de l'environnement non stationnaire.

Soient deux agents A et B dans un même environnement (que les agents observent entièrement).
Si B apprend :
$\rightarrow$ Le comportement de B évolue.
$\rightarrow$ A n'est pas dans un MDP stationnaire.

```
autres modèles
      +
autres algorithmes
```

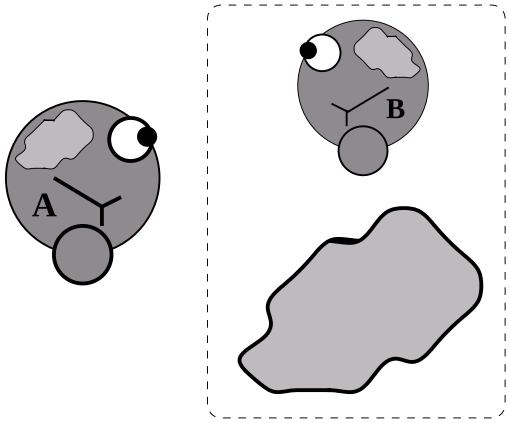


FIG. 5.1 – le premier problème de l'apprentissage dans les SMAs

Le jeu Roshambo vu précédemment est un bon exemple du problème, d'autant que dans son cas, les agents vont à l'encontre l'un de l'autre, d'où une politique obtenue non déterministe pour chacun. Néanmoins, des exemples de coopération avec environnement complètement observé comme le tri à deux agents peuvent converger vers une paire de politiques déterministes qui soit optimale.

5.3 Autres problèmes

L'utilisation de systèmes multi-agents a pour intérêt de distribuer la tâche entre les agents, entre autres pour que chacun perçoive une partie du problème à travers son observation locale de l'environnement. Il est même rare que l'ensemble des perceptions des différents agents reconstitue une connaissance complète de l'état du système.

On pourrait imaginer que tous les agents partagent leurs informations pour agir au mieux, ce qui amènerait facilement à un système centralisé si tous "pensent" de la même façon ou si l'un d'entre eux pense pour les autres. Mais on perd alors l'avantage de la distribution du problème. Le fait de distribuer le problème et sa résolution donne la possibilité d'avoir des mécanismes de raisonnement plus simples dans chaque agent. Chacun va essayer de trouver une solution au problème commun (dans le cas d'une coopération totale) en prenant en compte seulement ses données locales, qu'il peut influencer par son comportement.

Néanmoins, une résolution ainsi distribuée n'est pas évidente. Si une coordination est nécessaire, il va falloir une mise en accord des différents agents concernés pour qu'elle aboutisse. C'est là que des mécanismes de communication, transfert de données ou prise de décision concertée, peuvent être utilisés. De tels mécanismes vont devenir une complication s'il faut les apprendre.

5.4 Modèles et approches multi-agents existants

5.4.1 Introduction

Par définition, un MDP modélise le comportement d'un agent dans un environnement donné. On se sert de cet outil pour trouver le meilleur comportement possible pour l'agent en utilisant des algorithmes de planification ou d'apprentissage. Mais l'utilisation faite des MDPs doit être bien distinguée du modèle lui-même : les informations connues par l'agent dépendent de l'algorithme utilisé plus que du modèle choisi.

Trois modèles de systèmes multi-agents ont été rencontrés, lesquels sont présentés en annexe. Le même problème va se poser avec ces modèles : ils sont issus des MDPs dans le but de les adapter aux SMAs. Or ces modèles diffèrent non seulement sur quelques points de définition, mais aussi et surtout sur l'utilisation que leurs auteurs en font. Il ne faut donc pas les juger trop hâtivement pour éviter de juger plus l'utilisation qui en est faite que les modèles eux-mêmes.

Les présentations qui se trouvent en annexe sont assez formelles. Les parties suivantes donnent une rapide comparaison et différentes remarques à leur propos.

5.4.2 Divers

5.4.2.1 Principales différences entre modèles

- MMDPs et DEC-(PO)MDPs ne sont adaptés qu'à des agents ayant un même but (fonction de gain commune), alors que les jeux de Markov permettent de représenter des agents ayant chacun un but propre.
- Seuls les DEC-POMDPs considèrent des systèmes partiellement observables.
- La fonction de gain d'un MMDP ne prend en compte que l'état actuel et pas les actions effectuées (ce qui peut facilement être modifié).

5.4.2.2 Questions, remarques...

A la lecture de ces présentations on s'aperçoit que, dans le cadre d'agents à but commun, les modèles ne présentent pas de différences très marquées. Aucun ne sort du lot. Certains permettent de modéliser des systèmes plus complexes que les autres. Mais c'est plus dans l'utilisation faite de ces modèles que résident leurs différences.

Si l'on regarde les algorithmes liés aux différents modèles, on peut noter que seuls les MMDPs ont réellement des algorithmes propres de planification et d'apprentissage. Les DEC-(PO)MDPs sont, eux, présentés sans algorithmes (l'article [Bernstein 00] s'intéresse à des problèmes de complexité). Et les jeux de Markov n'ont d'algorithmes spécifiques que dans le cas de jeux à deux joueurs de buts opposés (*minimax-Q*).

5.4.2.3 Ce que sait un agent

(selon les algorithmes utilisés)

Parmi les questions qui peuvent se poser, celle de savoir ce que connaît un agent est primordiale. En effet, les modèles permettent certes de formaliser le problème, mais l'agent ne connaît pas nécessairement toutes les données représentées dans le modèle.

Dans le tableau suivant est donc noté à titre indicatif pour chaque algorithme :
- s'il s'agit d'un algorithme d'apprentissage par renforcement (RL ?),
- si l'agent connaît sa fonction de gain (R_{propre}),
- si l'agent connaît la fonction de gain de ses adversaires ($R_{autresagents}$),
- si la loi de transition du modèle est connue (T),
- et une éventuelle note.

Remarque : Dans certains cas, l'agent peut ne pas connaître les buts des autres agents, mais savoir s'ils ont le même but que lui ou un but opposé, d'où les (R_{propre}) et $(-R_{propre})$ dans la colonne $R_{autresagents}$.

	RL ?	R_{propre}	$R_{autresagents}$	T	Autres agents	Note
MDPs						
Value/Policy Iteration		X	*N.S.*	X	*N.S.*	
Q-learning	X		*N.S.*		*N.S.*	
Fictitious Play	X	?			quelconques	adversaires étudiés
Jeux de Markov						
Q-learning	X				quelconques	
minimax-Q	X		$(-R_{propre})$		1 opposé	
MMDPs						
Value Iteration with State Expansion		X	X (R_{propre})	X	même gain	algo centralisé
Randomization with Learning	X		(R_{propre})		même gain	
DEC-POMDPs						
[Pas d'algo]			(R_{propre})		même gain	(seul POMDP ici)

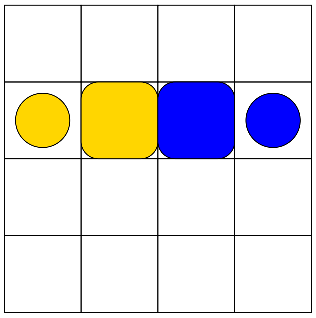
(*N.S.* : Non Significatif)

5.5 L'application choisie

5.5.1 Idée de départ

Les premiers essais effectués étaient concluants, mais un peu simplistes. Un problème plus complexe doit être posé pour regarder le comportement qui va effectivement apparaître dans une situation proche de problèmes réels pour un système multi-agents. Une véritable coopération doit être mise en jeu dans laquelle chaque agent a un rôle à jouer.

L'environnement choisi est un quadrillage à deux dimensions dont la taille peut être choisie, sur lequel sont placés des blocs (aussi appelés cubes) de couleur jaune ou bleu. Deux types d'agents sont présents, distingués eux aussi par leur couleur jaune ou bleu. Le but est, pour les agents, de pousser un bloc bleu contre un bloc jaune, ce qui requiert le travail de deux agents.



Ici les agents n'ont plus qu'à pousser les blocs pour effectuer la tâche demandée.
Note : les agents ne doivent pas nécessairement pousser les blocs de leur couleur.

FIG. 5.2 – exemple de situation

Les parties suivantes présentent les actions et perceptions d'un quelconque agent de notre système.

5.5.2 Actions

Pour se replacer dans le cadre des MDPs, il faut définir ce que seront les actions des agents. Le choix aurait pu être fait d'agents orientés (mouvements : avant, arrière, tourner à droite, tourner à gauche). Mais il s'avère plus simple dans notre situation d'utiliser des agents non orientés (il n'y a pas à considérer la direction de l'agent, ce qui divise par quatre le nombre de situations possibles). On obtient donc comme actions les déplacements vers les quatre points cardinaux indiqués sur la figure 5.3.

- NORD (N)
- OUEST (O)
- SUD (S)
- EST (E)

Note : il n'est pas possible de vouloir rester sur place, même si l'on peut y être contraint.

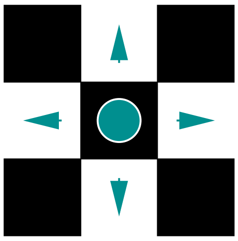


FIG. 5.3 – les actions possibles

5.5.3 Perceptions

Pour pouvoir décider de ses actions, l'agent doit avoir une idée de la situation dans laquelle il se trouve. On aurait pu envisager avoir des agents connaissant l'état exact du système, mais le nombre de situations possibles s'avère beaucoup trop grand pour utiliser un tel MDP "normalement". Il faudrait pour cela avoir en mémoire des tables ayant autant d'états que de dispositions possibles des agents et cubes, ce qui est d'autant plus problématique que la politique apprise serait alors fortement liée au nombre d'agents et de cubes.

Au lieu de cela, on va limiter les perceptions à des informations sur quelques éléments de l'environnement :

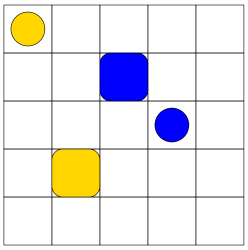
– la direction de l'agent de couleur opposée le plus proche (N-O-S-E)	$*4$
– la direction du bloc jaune le plus proche (N-NO-O-SO-S-SE-E-NE)	$*8$
– la direction du bloc bleu le plus proche (N-NO-O-SO-S-SE-E-NE)	$*8$
– le fait qu'un bloc jaune se trouve sur l'une des huit cases les plus proches	$*2$
– le fait qu'un bloc bleu se trouve sur l'une des huit cases les plus proches	$*2$
Total :	$= 1024(/4)$

Cette limitation des perceptions, hormis le fait que nous sommes dans un système multi-agents, nous fait passer du cadre des MDPs à celui des POMDPs, du fait qu'un agent a une vision bonne à courte distance, est beaucoup moins précise quand les objets s'éloignent. On peut noter par ailleurs qu'un agent n'a pas réellement de notion de la présence d'un autre agent. Il ne voit son congénère que comme un élément de l'environnement, un simple objet.

Pour en revenir au choix des perceptions, dans un premier temps, seules les informations de direction étaient utilisées, ce qui donnait $4 * 8 * 8 = 256$ états possibles. Mais ces perceptions s'avéraient trop insuffisantes pour obtenir une bonne politique. Il a donc fallu ajouter les données sur la proximité de cubes de couleurs, ce qui multiplie par 4 le nombre d'états.

En notant que le problème est invariant par rotation de 90°, on peut diviser par 4 le nombre obtenu (ce qui accélère aussi l'apprentissage). Il suffit de toujours se ramener au cas où l'agent de couleur opposée est au nord.

Certaines situations étant impossibles, il y a en fait un peu moins de 256 cas possibles.



agent	dir(bj)	dir(bb)	dir(autre agent)	près(bj)	près(bb)
jaune	S	E	SE	non	non
bleu	O	NO	NO	non	oui

bj : bloc jaune - bb : bloc bleu

FIG. 5.4 – exemple de perceptions

Note : L'utilisation d'observations partielles ayant entre autres intérêts de diminuer la taille de l'espace exploré, on n'utilisera pas le MDP sous-jacent dans l'apprentissage, même s'il peut être connu : cela irait dans le sens opposé à la simplification qui vient d'être faite.

5.5.4 Récompense

Le premier cas dans lequel récompenser un agent est simple à trouver : quand l'agent et l'un de ses confrères poussent deux blocs l'un contre l'autre. On peut ensuite imaginer donner des récompenses plus régulièrement dans le jeu, si les agents se rapprochent des cubes par exemple. Mais il n'est pas toujours bon de chercher un tel rapprochement, car certaines situations peuvent nécessiter au contraire un éloignement. De plus, il est souhaitable d'intervenir le moins possible, donc d'avoir une fonction de gain des plus simples.

Conditions	Gain
fusionner deux blocs (bleu+jaune)	bonus $(+3)$

FIG. 5.5 – fonction de gain

En revanche, si l'on veut aider les agents à se spécialiser dans la poussée de blocs d'une certaine couleur, une récompense dépendant de la couleur du bloc manipulé peut orienter le comportement de l'agent. On peut aussi espérer que les agents d'eux mêmes vont se répartir les rôles, mais la convergence risque d'être beaucoup plus lente dans ce cas.

Conditions	Gain
fusionner deux blocs en poussant sa couleur	bonus $(+3)$
fusionner deux blocs en poussant l'autre couleur	malus (-2)

FIG. 5.6 – fonction de gain (pour spécialiser)

5.5.5 Méthode d'apprentissage

Différentes idées peuvent être exploitées pour réduire le temps d'apprentissage :
- réduire la taille du monde,

– placer les agents dans des situations précises pour leur faire apprendre une partie de leur comportement,
– donner un exemple (en menant un agent).

Dans le cas présent, les agents n'ont pas reçu d'exemples. Par contre, un système de scripts nous a permis d'entraîner les agents dans des situations simples au début (agents n'ayant qu'un mouvement à effectuer pour obtenir une récompense, le tout répété 1500 fois), puis dans des situations de plus en plus complexes (le monde étant de taille croissante), avant d'être laissés indéfiniment dans un monde $6 * 6$.

Un fichier script est une suite de blocs de données de la forme :

nombre de répétitions de la situation	nombre de coups avant répétition
largeur du quadrillage	hauteur du quadrillage
abscisse de l'agent jaune	ordonnée de l'agent jaune
abscisse de l'agent bleu	ordonnée de l'agent bleu
abscisse du bloc jaune	ordonnée du bloc jaune
abscisse du bloc bleu	ordonnée du bloc bleu

FIG. 5.7 – un bloc de données type

Les différentes méthodes présentées ne posent pas de problème dans l'apprentissage : on peut changer la taille du monde sans soucis dans la mesure où les perceptions et actions des agents ne dépendent pas des dimensions de l'environnement. On continue donc toujours à travailler sur la même table Q, même quand on change la situation courante.

Note : Il reste évidemment préférable d'intervenir le moins possible dans l'apprentissage des agents.

5.5.6 Remarques

Le monde choisi étant rectangulaire, un bloc ne doit pas pouvoir être placé dans un coin, sous peine de ne plus pouvoir être déplacé. En fait, on va même interdire de placer un bloc sur une case en bord du monde. Évidemment, il peut rester des situations bloquantes si le nombre d'agents et de blocs est trop important par rapport à la taille du monde, mais ces contraintes ne sont pas gérées par le programme.
Une autre précision à donner est qu'un agent peut pousser un nombre variable de blocs, et même d'agents.

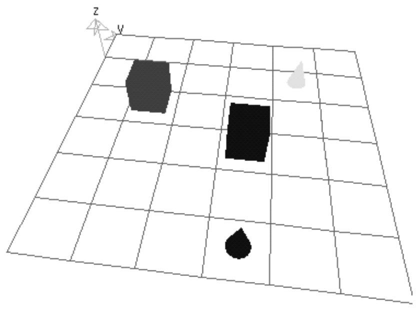


FIG. 5.8 – un exemple de l'affichage réel du programme

5.6 Essai de Q-learning

5.6.1 Introduction

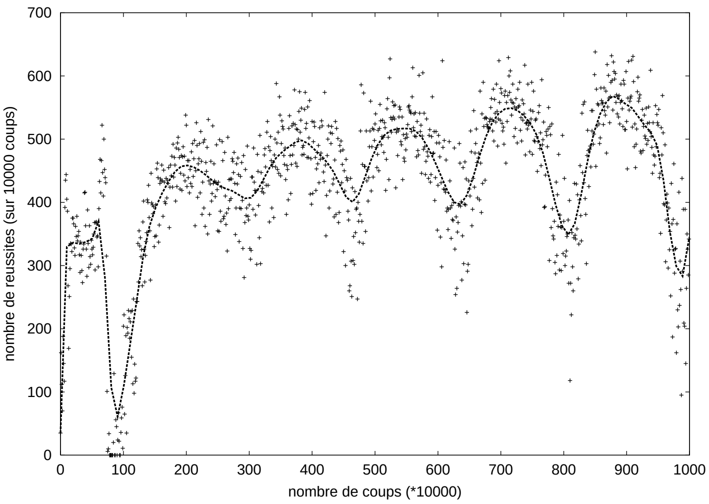
Il faudrait, pour faire du Q-learning, être dans le cadre d'un MDP stationnaire, ce qui n'est pas le cas ici. Cet essai va permettre de voir si, avec des agents se croyant seuls dans un environnement stationnaire, peut se faire un bon apprentissage de comportement coopératif.

Cette première méthode d'apprentissage, même si elle n'est pas utilisée dans les meilleures conditions, permet déjà de faire un certain nombre d'essais suivant des paramètres et choix divers. Il n'y a de toute façon pas de méthode adaptée au cas qui nous intéresse, donc il est aussi simple de partir d'un outil assez connu qui servira de référence. Pour l'instant, nous nous plaçons dans un cadre à deux agents et deux blocs, les "mesures" étant effectuées dans un monde de 6x6 cases.

Le Q-learning nécessite d'identifier des **états** s et des **actions** a. Les actions sont les quatre déplacements (N-O-S-E) indiqués dans la description du problème. Les états vont correspondre aux différentes observations possibles, combinaisons de perceptions présentées aussi dans la description du problème (1024 combinaisons, réduites à 256 par rotation). Ce sont ici bien évidemment les "états" qui n'en sont pas et font que l'on ne respecte pas le cadre normal du Q-learning.

5.6.2 Mesures

Sur toutes les courbes présentées par la suite on a : en abscisse le nombre de buts atteints lors des 10000 derniers coups, et en ordonnée le nombre de coups déjà joués (divisé par 10000). Les courbes présentées ont déjà été lissée (sinon on verrait des amas de points trop mélangés).



Le nuage de points correspond aux données récupérées au cours de l'exécution du programme, et la courbe est une courbe de Bézier obtenue à partir de ces points.

FIG. 5.9 – obtention des courbes à partir de nuages

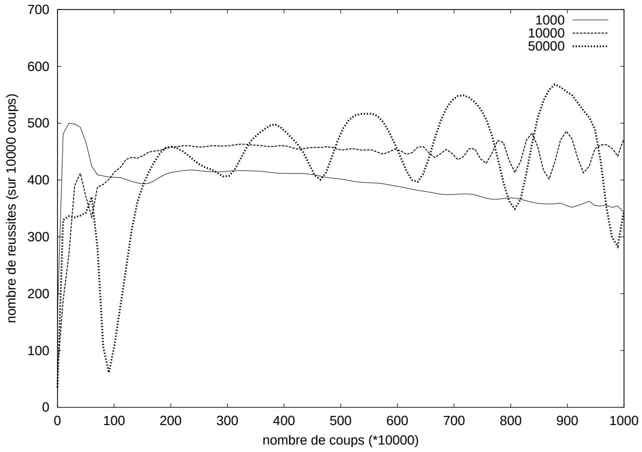
Note : Les images en coin de page montrent une suite de mouvements des deux agents une fois leur apprentissage effectué (animation commençant par la fin du rapport), de même que quelques images présentées en annexe.

5.6.3 Influence de α

Pour rappel, le coefficient α permet de contrôler l'apprentissage, en le faisant passer progressivement d'un apprentissage tenant fortement compte de toute nouvelle expérience à aucun apprentissage une fois que le comportement est considéré comme stable.

La courbe d'évolution du coefficient α n'est pas définie dans l'algorithme du Q-learning. On sait simplement qu'il doit être compris entre 0 et 1, et tendre vers 0.

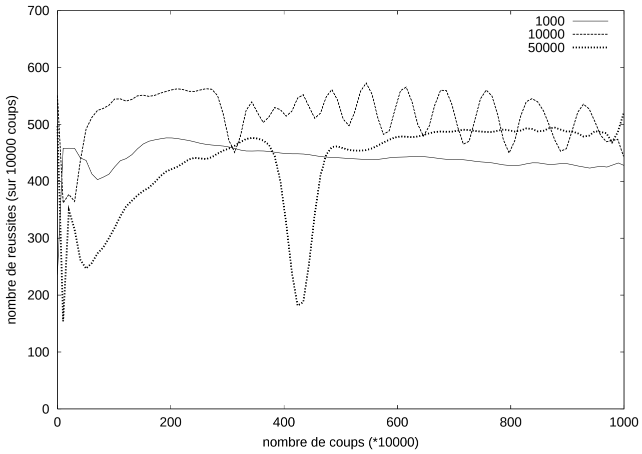
Dans un premier temps, α est fixé pendant un certain nombre d'étapes, de façon à ne pas commencer à figer la mémoire d'un agent avant d'avoir fait un certain nombre d'essais. α a été pris arbitrairement égal à 0.35 pendant les 250 premiers essais de chaque action depuis chaque état. Dès qu'une action a été utilisée plus de 250 fois, α commence à décroître en $1/n$ (où n est le nombre de passages).



Comportements obtenus pour différents α_{min} (on a indiqué ici $1/\alpha_{min}$). Ici on force une spécialisation.

FIG. 5.10 – différents α_{min} avec spécialisation

Ayant une décroissance en $1/n$, l'apprentissage risque de s'arrêter trop tôt. On a donc fixé à ce coefficient α une limite inférieure (il y a d'abord une limite aux compteurs de passages). On obtient de meilleurs résultats si le minorant est de $1 * 10^{-4}$ que pour $1 * 10^{-3}$ quand on force la spécialisation (la courbe fortement oscillante est ici difficilement interprétable).



Comportements obtenus pour différents α_{min} (on a indiqué ici $1/\alpha_{min}$). Pas de spécialisation forcée.

FIG. 5.11 – différents α_{min} sans spécialisation

Si l'on ne cherche pas à spécialiser les agents, on obtient de meilleurs résultats avec un minorant est de $1 * 10^{-4}$ que pour $5 * 10^{-4}$ (pour lequel il y a un "accident" dans l'apprentissage). Dans le cas présent, cette limite permet à l'agent d'apprendre un peu plus longtemps sans trop figer sa politique.

L'évolution montrée ici suit en fait une première phase d'apprentissage où les agents ont été placés dans des situations particulières, ce qui explique probablement sur la courbe claire une rechute due à une modification de l'apprentissage (adaptation à un monde plus grand). Mais il semble difficile d'interpréter ces résultats de façon précise. On peut noter tout de même une oscillation régulière et croissante de la courbe foncée, et peut-être une plus lente sur la claire.

Conclusions : La fonction α utilisée est simple au départ, mais commence à se compliquer pour améliorer les performances. Il serait probablement intéressant d'utiliser une fonction différente pour avoir une meilleure convergence. D'autres algorithmes d'apprentissage permettent pour leur part de modifier la valeur de α quand un essai fructueux motive une relance de l'apprentissage (*prioritized sweeping* et *Queue-Dyna* sont de telles méthodes, mais avec apprentissage d'un modèle du monde).

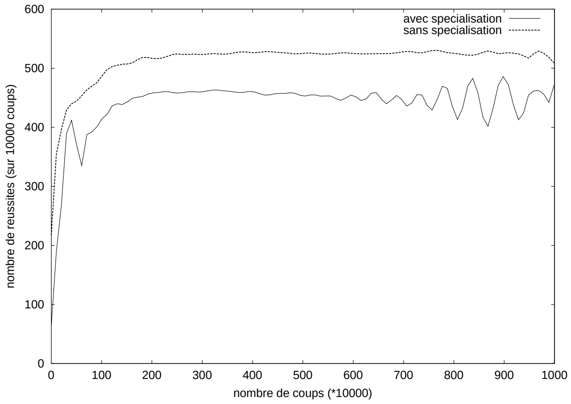
De façon générale, le problème est de savoir comment faire évoluer la valeur de α pour que l'apprentissage se fasse au mieux. Faire varier α en fonction du temps est une première solution assez empirique et qui va beaucoup dépendre du problème posé (de sa taille entre autre). La deuxième solution évoquée est d'observer l'évolution de la table apprise, ce qui implique un travail un peu lalorieux et coûteux en temps machine, mais qui peut fortement améliorer l'efficacité de l'apprentissage (d'autant que dans des situations réelles telles qu'avec de vraies machines, c'est le temps des actions qui va être prépondérant).

5.6.4 Influence de la spécialisation

Sachant qu'il peut y avoir un intérêt à spécialiser les agents (voir l'exemple du tri de liste), des essais ont été menés pour voir ce que l'on obtient si l'on force les agents à agir plutôt sur les cubes de sa propre couleur. En fait, on va :

– soit toujours récompenser quand les deux agents poussent les blocs l'un contre l'autre (apprentissage sans spécialisation),
– soit transformer cette récompense en malus si les agents poussent un bloc d'une couleur différente de sa propre couleur (apprentissage avec spécialisation).

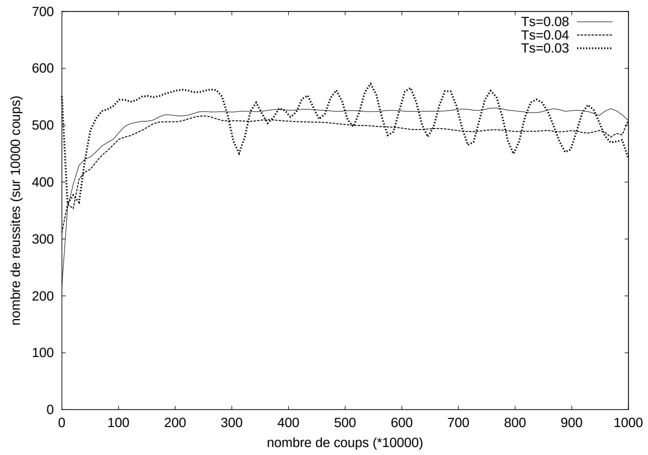
Dans le cas présent, le fait de vouloir spécialiser les agents est plus un frein à l'obtention des politiques jointes les plus efficaces.



Conclusions : Le but de la spécialisation est d'aider les agents à résoudre certains conflits en les incitant à une certaine assignation de rôles. Sans cette aide, ils peuvent ne pas réussir à apprendre des politiques se coordonnant. Il n'y a pas lieu de forcer une telle spécialisation si aucun conflit n'apparaît qui ne soit rapidement résolu. D'autre part, la communication entre agents peut aussi aider à dénouer certaines situations critiques en définissant des rôles pendant une courte période.

5.6.5 Influence de la température

Dans la situation courante, une politique déterministe risque d'être catastrophique, différents états du systèmes pouvant nécessiter des actions différentes alors qu'ils correspondent à une même observation pour un agent. Pour obtenir une politique toujours stochastique en utilisant le Q-learning, la solution communément employée est de fixer une température seuil T_{seuil}. Il n'y a hélas pas a priori de méthode simple pour fixer au mieux ce seuil. Des approches cherchant explicitement les distributions sur l'espace des actions qui constituent la politique fourniraient probablement de meilleurs résultats.



Différents essais d'apprentissage montrant l'effet du choix de différentes températures seuil.

FIG. 5.12 – choix de T_{seuil}

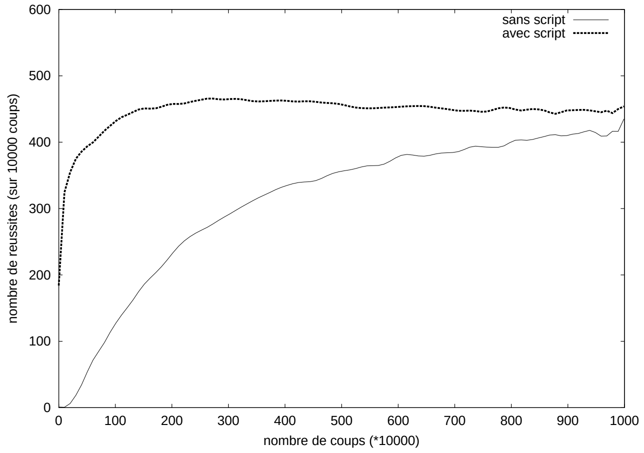
Si l'on voit ici qu'effectivement le niveau de qualité de la politique obtenue dépend de la température, il apparaît aussi que les oscillations malencontreuses qui avaient déjà été rencontrées disparaissent au dessus d'un certain seuil. Rien ne le prouve réellement, mais il est probable qu'essayer d'atteindre une politique "trop déterministe" soit à l'origine de cette instabilité.

Conclusions : Il n'y a pas lieu de discuter plus ici de la température T_{seuil}. Ce coefficient n'est utilisé que pour palier le fait que le Q-learning n'est adapté qu'aux MDPs stationnaires. Il serait préférable d'utiliser un algorithme cherchant directement la politique stochastique la meilleure possible. La variable T par contre permet de gérer le dilemme exploration/exploitation mais, comme pour α, il serait meilleur de régler son évolution en fonction des expériences vécues que d'utiliser une fonction du temps comme c'est le cas ici.

5.6.6 Influence de l'aide des scripts

Un essai a été effectué pour comparer l'apprentissage avec ou sans script d'aide. Dans la plupart des apprentissages effectués, une première phase est dirigée par un script, lequel permet de placer un certain nombre d'essais en partant de situations données.

Un seul script a été utilisé, celui-ci durant un peu plus de 100000 coups. Comme on peut le voir sur la figure 5.13, l'apprentissage est notablement accéléré.



L'apprentissage sans aide d'un script étant plus long, sa courbe est présentée ici en deux parties (1/2 : de 0 à 1.10^6 coups, et 2/2 : de 1.10^6 à 2.10^6 coups).

FIG. 5.13 – intérêt de l'usage de scripts

Conclusions : Il y a peu à dire ici, car ces scripts vont dépendre du problème posé. On peut simplement se demander dans quelle mesure on n'en fait pas trop en encadrant ainsi l'apprentissage (surtout Si l'on considère que l'apprentissage a pour principal intérêt de permettre aux agents de s'adapter à de nouvelles situations).

5.6.7 Conclusion

Nous avons pu voir à travers un exemple les différents paramètres d'un apprentissage par renforcement. Néanmoins, si le Q-learning permet d'adapter facilement un algorithme classique et assez efficace au cas des POMDPs, et si les résultats obtenus sont assez satisfaisant dans notre exemple multi-agents, les réglages qui peuvent être nécessaires et le fait de façon générale qu'il ne s'agisse pas d'un algorithme conçu dès le départ pour le problème posé (SMA+POMDP), font apparaître le besoin d'une méthode réellement adaptée de recherche de politique.

Chapitre 6

Conclusion et perspectives

On a bien retrouvé dans notre problématique les mêmes points délicats que pour l'apprentissage dans le cas d'un agent seul. Mais différents aspects liés à la présence d'autres agents poussent à utiliser d'autres approches que l'usuel Q-learning. Quelques unes de ces approches sont décrites ici.

6.1 Communication

Un aspect des systèmes multi-agents qui n'a pas été évoqué ici est la communication entre agent. Celle-ci donne les possibilités de marquer les intentions des agents, montrer leur état de satisfaction, transmettre des informations sur l'environnement...

Dans l'exemple développé précédemment, où deux agents doivent fusionner des blocs de couleurs différentes, si l'on se place dans un monde contenant plus de blocs, les agents auraient intérêt à se préciser mutuellement de quel bloc ils veulent s'occuper (en le marquant par exemple).

On peut considérer la communication comme construite à partir de signaux dirigés vers un ou plusieurs interlocuteurs, ces signaux pouvant être organisés selon un protocole. Des recherches sont d'ailleurs effectuées pour voir comment un langage se crée et évolue dans un système multi-agents, afin de vérifier les hypothèses soulevées par l'étude du langage humain [De Jong 99].

La communication ne passe pas nécessairement par l'usage d'un langage oral ou écrit, mais peut utiliser tous types de signes : le faciès de l'animal, ou des traces laissées (marquage de territoire ou phéromones déposées par des fourmis pour indiquer un chemin à leurs consœurs).

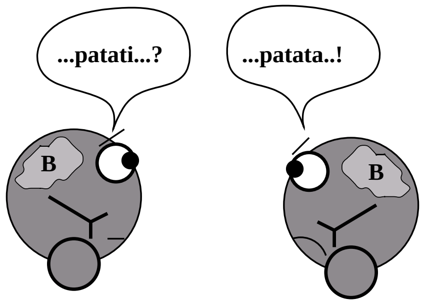


FIG. 6.1 – communication par signes oraux et visuels

Lors du stage, en cherchant un problème à mettre en œuvre, des sujets ont été proposés dans lesquels les agents auraient dû marquer un objet par un drapeau, ou changer leur propre couleur pour exprimer un changement d'état interne. On peut d'ailleurs déjà considérer, dans l'exemple des agents devant fusionner deux blocs, que le fait de se placer à côté d'un des blocs signifie une volonté de s'occuper de celui-ci, et que l'autre agent s'occupe de l'autre bloc. Il s'agit d'une communication indirecte (et non intentionnelle) passant par des interactions avec l'environnement.

6.2 Preuves de convergence, sûreté

S'il existe des résultats prouvant la convergence d'algorithmes tels que *value iteration* ou *Q-learning* dans le cadre de MDPs mono-agents, il reste beaucoup à faire pour avoir des algorithmes convergeants avec des POMDPs ou dans le cas d'un système multi-agents. Il est possible que cela passe aussi par d'autres méthodes d'apprentissage par renforcement, essayant par exemple de reconstruire un modèle du monde comme on pourrait sans doute le faire à partir des U-tree de [McCallum 95]. Dans un POMDP, l'environnement est modélisé par le MDP partiellement observé. Comme il reste souvent utopique d'utiliser ce MDP sous-jacent, on peut imaginer qu'un agent se construise son propre modèle simplifié du monde, en considérant éventuellement à part les modèles de comportement des autres agents.

6.3 Comportement

Un autre aspect qui n'a pas été rencontré jusqu'ici est l'utilisation de quelques comportements de base pré-éxistants plutôt que de simples actions. Le problème est de définir à quel niveau de complexité on veut se placer. On peut déjà considérer que l'action "avancer" est une tâche complexe qui pourrait être décomposée en des commandes motrices vers des roues ou des articulations (avec asservissements...).

Parmi les travaux utilisant des comportements de haut-niveau (rentrer à la maison, suivre un mur, se reposer) dans un cadre multi-agents, on peut citer [Mataric 97].

Dans le domaine de l'apprentissage, certains modèles de robots ont été définis par couche de plus ou moins haut niveau, un système d'apprentissage s'occupant d'asservissement moteur, l'autre de comportements simples (aller à la maison, manger...), et un dernier par exemple gérant les choix de ses derniers comportements dits simples.

6.4 Modélisation de l'environnement

On va de manière générale rencontrer dans le cadre des systèmes multi-agents les mêmes problèmes que dans le cas d'un agent seul :
- passage d'un temps continu à un temps discret,
- passage d'un espace continu à un espace discret,
- généralisation,
- choix de perceptions suffisantes parmi les disponibles,
- ...

Annexe A

Modèles de MDPs multi-agents

A.1 Jeux de Markov

[Littman 94] [Owen 82]

A.1.1 Définition

Un jeu de Markov à k joueurs est défini par :

- un ensemble d'états S,
- une collection d'ensembles d'actions $A_1, ..., A_k$, un pour chaque agent,
- une fonction de transition d'état $T : S \times A_1 \times ... \times A_k \rightarrow PD(S)$,
- $\forall i \in [1..k]$ une fonction de gain $R_i : S \times A_1 \times ... \times A_k \rightarrow \Re$.

Précisions : Les algorithmes d'apprentissage utilisés (Q-learning et minimax-Q) ne nécessitent pas la connaissance des fonctions de gain ou de transition des différents joueurs (Il faut noter qu'il ne s'agit là que de jeux à deux joueurs et à somme nulle.).

A.1.2 Utilisation

Pour pouvoir réellement maximiser son gain à plus ou moins long terme, un joueur (un agent) doit prendre en compte le fait que les autres joueurs cherchent eux aussi à maximiser leur gain.Or les fonctions de gain des autres joueurs ne sont a priori pas connues. Il faudrait se contenter de considérer les autres agents comme faisant partie de l'environnement, et connaître leurs comportements par l'expérience.

On retrouve souvent dans les jeux de Markov le cas particulier des jeux à deux joueurs dans lesquels la somme des gains des joueurs est nulle. Le gain de l'un est alors la perte de l'autre (donc la fonction de gain de l'adversaire est connue). On recherche alors la politique optimale par l'algorithme du minimax-Q. L'idée est de trouver l'action, dans un état donné, qui va maximiser le gain, sachant que l'adversaire cherche simultanément à minimiser ce même gain.

Conclusion : L'usage des jeux de Markov semble se restreindre souvent au dernier cas présenté (deux joueurs de fonctions de gain opposées), un algorithme d'apprentissage adapté étant alors connu.

A.2 MMDP (MDP Multi-agents)

[Boutilier 96] [Boutilier 99]

A.2.1 Définition

Un processus de décision Markovien multi-agents est un quadruplet $\langle S, \alpha, \{A_i\}_{i\in\alpha}, Pr, R\rangle$ où :
- S est un ensemble fini d'états,
- α est un ensemble fini d'agents,
- $\forall i$, A_i est l'ensemble fini des actions individuelles de l'agent i,
- $Pr : S \times A_1 \times ... \times A_n \times S \to [0, 1]$ est une fonction de transition,
- et $R : S \to \Re$ est une fonction de gain à valeurs réelles.

Précisions : Dans les MMDPs, tous les agents ont accès aux mêmes informations. Ils peuvent donc facilement se coordonner en agissant suivant les mêmes conventions (tous les agents suivent le même raisonnement). Une autre méthode possible est d'agir au hasard jusqu'à tomber sur une coordination (voir plus loin).

A.2.2 Notations

A.2.2.1 MMDP en général

On note $A = \times_{i\in\alpha} A_i$ l'ensemble des *actions jointes*.

Une *politique individuelle* stationnaire pour un agent i est l'application $\pi : S \to \Delta(A_i)$ qui donne, pour tout état s, la *distribution de probabilités* des actions de l'agent.

La politique est *déterministe* s'il existe toujours une action de probabilité 1, les autres étant de probabilité 0. Elle est *randomisée* sinon.

A.2.2.2 Jeu à un niveau

Un jeu à un niveau G est un MDP pour lequel il n'y a qu'un état, et une seule étape dans le jeu (exemple : jeu pierre-papier-ciseaux).

Une action jointe $a \in \mathcal{A}$ est *optimale* dans un jeu à un niveau G si $\forall a' \in \mathcal{A}\ R(a) \geq R(a')$. L'action $a_i \in A_i$ est *potentiellement individuellement optimale* (PIO) pour l'agent i si une action jointe optimale contient a_i. On note PIO_i l'ensemble des telles actions pour l'agent i.

Le jeu à un niveau $G = \langle \alpha, \{A_i\}_{i\in\alpha}, R\rangle$ induit un *problème de coordination* (CP) ssi il existe des actions $a_i \in PIO_i, 1 \leq i \leq n$, telles que $\langle a_1, ..., a_n\rangle$ n'est pas optimale. Les différentes actions $a_1, ..., a_n$ font partie de différentes actions jointes optimales, mais ne forment pas une telle action jointe.

A.2.3 Description

Les MMDPs peuvent être considérés comme des MDPs dans lesquels les actions sont distribuées entre différents agents. Ayant une fonction de gain commune, les agents ne se trouvent pas en concurrence. De plus, on considère ici des agents ayant une observation complète du système.

Parmi les actions menées par un agent, seules certaines nécessitent une coordination. Il faut donc détecter ces problèmes de coordination (CP) et les résoudre.

Résolution des CPs (dans un jeu à un niveau) :

- *Apprentissage avec hasard :* les agents choisissent les actions PIO aléatoirement jusqu'à ce qu'une coordination soit effectuée. La politique est alors fixée définitivement.
- *Conventions lexicographiques :* nécessitent d'ordonner les agents et leurs actions pour convenir de la politique à suivre.
- *Communication...*

Dans le cas d'un MMDP en général, on peut aussi faire un apprentissage avec hasard, en gérant les effets de la découverte d'une nouvelle coordination et en détectant d'éventuels nouveaux CPs.

Boutilier propose aussi un algorithme dérivé de *Value Iteration* qui prend en compte les CPs. Mais il s'agit d'un algorithme centralisé.

A.3 Dec-(PO)MDP (Decentralized-(PO)MDP)

[Bernstein 00]

A.3.1 Définition

Pour m agents, un DEC-POMDP est défini par :

- un ensemble fini d'états S, (dont $s_o \in S$ comme état de départ)
- des probabilités de transition $P(s'|s, a^1, ..., a^m)$,
- des récompenses escomptées $R(s, a^1, ..., a^m)$ qui dépendent des actions de tous les agents,
- $\forall i \in [1..m]$, l'agent i a :
  - un ensemble fini d'observations : Ω^i,
  - une table de probabilités d'observations O^i, où $O^i(o^i|a^1, ..., a^m, s')$ est la probabilité qu'o^i soit observé étant donné le m-uplet d'actions $\langle a^1, ..., a^m\rangle$ effectuées et ayant mené à l'état s',
  - un ensemble d'actions A^i_o pour toute observation $o^i \in \Omega^i$.

A.3.2 Notations

$\forall\langle a^1, ..., a^m, s'\rangle\forall i$ (i agent), on note $\omega^i(a^1, ..., a^m, s')$ l'ensemble des observations qui ont une chance non-nulle de se produire étant donné le m-uplet d'actions $\langle a^1, ..., a^m\rangle$ effectuées et ayant mené à l'état s'.

On parle de *DEC-MDP* si $\forall\langle a^1, ..., a^m, s'\rangle$, $\forall\langle o^1, ..., o^m\rangle \in \big(\omega^1(...) \times ... \times \omega^m(...)\big)$, l'état peut être déterminé à partir du m-uplet $\langle o^1, ..., o^m\rangle$.

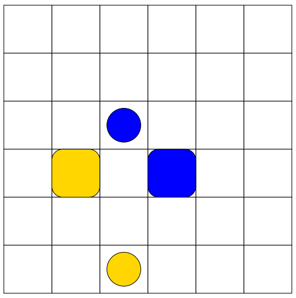
On définit une *politique locale*, δ_i, comme étant l'association d'une action $a^i \in A^i_{o_t}$ à une *histoire locale* $o^i_1, ..., o^i_t$.

Une *politique jointe*, $\delta = \langle\delta^1, ..., \delta^m\rangle$ est définie comme le m-uplet des politiques locales.

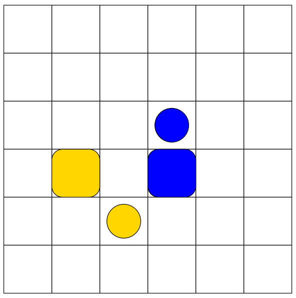
Utilisation : Rien n'est précisé sur ce point. L'article s'intéresse à des problèmes de complexité.

Annexe B

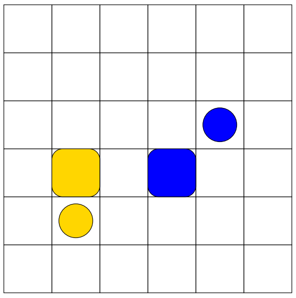
Exemple de mouvement de deux agents



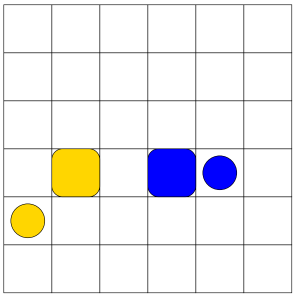
1



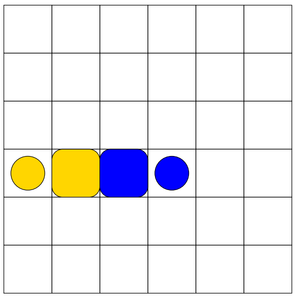
2



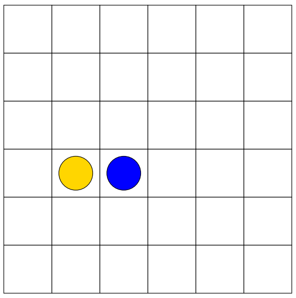
3



4



5



6

Bibliographie

[Bernstein 00] Daniel S. Bernstein et Shlomo Zilberstein, *The Complexity of Decision-Theoretic Planning for Distributed Agents*, University of Massachusetts

[Boutilier 96] C. Boutilier, *Planning, Learning and Coordination in Multiagent Decision Processes*. Dept. of Computer Sciences, University of British Columbia, Vancouver.

[Boutilier 99] C. Boutilier, *Sequential Optimality and Coordination in Multiagent Systems*. Dept. of Computer Sciences, University of British Columbia, Vancouver.

[Crites 98] Robert H. Crites, Andrew G. Barto, *Elevator Group Control Using Multiple Reinforcement Learning Agents*, Machine Learning, 33, 235-262 (1998)

[De Jong 99] De Jong, E.D., *Analyzing the Evolution of Communication from a Dynamical Systems Perspective*, Proceedings of the European Conference on Artificial Life ECAL'99, 689-693.

[Dutech 99] Alain Dutech, *Apprentissage d'environnement : approches cognitives et comportementales*. PhD Thesis, ENSAE, Toulouse, 1999.

[Ferber 95] Ferber J. *Les systèmes multi-agents*. InterEditions.

[Jennings 98] Nicholas R. Jennings, Katia Sycara et Michael Wooldridge, *A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems*, 1, 7-38, 1998

[Kaelbling 96] Leslie Pack Kaelbling et Michael L. Littman, *Reinforcement Learning : A Survey*, Journal of Artificial Intelligence Research 4, 1996

[Littman 94] Michael L. Littman, *Markov games as a framework for multi-agent reinforcement learning*, Brown University / Bellcore

[Littman 94-b] Michael L. Littman, *The Witness Algorithm : Solving Partially Observable Markov Decision Processes*, Technical Report CS-94-40, Brown University, Department of Computer Science, Providence, Rhode Island, 1994

[Littman 95] Michael L. Littman, Anthony R. Cassandra, Leslie Pack Kaelbling, *Learning policies for partially observable environments : Scaling up*. Dept. of Computer Science, Brown University, Providence.

[Littman 96] Littman M. *Algorithms for Sequential Decision Making, Chap. 5*. Ph.D. dissertation and Technical Report CS-96-09, Brown University, Department of Computer Science, Providence, RI, March 1996.

[Mataric 97] Maja J Mataric, *Reinforcement Learning in the Multi-Robot Domain*, Autonomous Robots, 4(1), Mar 1997, 73-83

[McCallum 95] R. Andrew McCallum, *Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State*, University of Rochester

[Owen 82] Owen G., 1982. *Game Theory : Second Edition.* Academic Press, Orlando, Florida.

[Sen 99] Sandip Sen et Gerhard Weiss, *MULTIAGENT SYSTEMS A Modern Approach to Distributed Artificial Intelligence*, Chapter 6 : Learning in Multiagent Systems

[Sheppard 97] J.W. Sheppard, *Multi-Agent Reinforcement Learning in Markov Games*. PhD Thesis, The John Hopkins University, 1997

[Singh 94] Satinder P. Singh, Tommi Jaakkola et Michael I. Jordan, *Learning Without State-Estimation in Partially Observable Markovian Decision Processes*, Massachusetts Institute of Technology

[Watkins 89] Watkins C. *Learning from delayed reward*. PhD Thesis, King's College of Cambridge, 1989.