Utiliser des protocoles d'interaction et de la mémoire pour l'apprentissage par renforcement d'actes de communication

S. Hoet* N. Sabouret*
Shirley.Hoet@lip6.fr Nicolas.Sabouret@lip6.fr

*Laboratoire d'Informatique de Paris 6
104, avenue du président Kennedy
75016 Paris – FRANCE

Résumé :

Dans ce papier, nous nous intéressons à l'apprentissage par renforcement pour la communication entre agents dans les SMA. L'étude de la littérature nous a permis de mettre en évidence trois principaux problèmes. Tout d'abord il est nécessaire pour l'agent de construire les actes de communication qu'il utilise pour communiquer avec les autres agents. Ensuite il nous faut adapter les actes de communication afin de les modéliser en tant qu'actions pour leur appliquer des techniques d'apprentissage par renforcement. Enfin nous devons résoudre les problèmes liés à la non markovité des systèmes multi-agents afin de pouvoir utiliser les algorithmes usuels d'apprentissage par renforcement utilisé dans le cadre mono-agent. Nous présentons un algorithme d'apprentissage par renforcement qui permet de traiter ces trois problèmes de manière entièrement distribuée. Nous nous appuyons pour cela sur l'utilisation d'agents introspectifs capables de raisonner sur leurs actions et sur leurs états. Puis nous démontrons que nous pouvons assimiler un acte de communication à une action en nous fondant sur la théorie des actes de langage et les travaux sur les PDSMs. Enfin nous montrons comment l'utilisation d'actes de communication peut résoudre le problème de non markovité en étant toutefois complétée par l'utilisation de la mémoire de l'agent.

Mots-clés : Apprentissage par renforcement, Acte de language, Système multi-agent

Abstract:

In this paper, we propose a reinforcement learning mechanism for multi-agent communication. Existing work underlines three main problems. First, agents need to construct appropriate communicative acts. Second, we have to implement the whole communication process as agent actions, so as to apply classical reinforcement learning algorithms. Last, the non-markovity of MAS raises several issues that have to be solved to guarantee the algorithm convergence. We propose a distributed method to deal with these three problems. This method is based upon introspective agents, that can reason about their own actions and data. We then prove that communicative acts can be seen as actions, according to the speech acts theory, using a Semi-Markovian model of the problem. Last, we show how communicative acts can solve non-markovity issues, using a limited memory of passed actions and observations.

Keywords: Reinforcement Learning, Communication, Multi-Agent System

1 Introduction

Dans les systèmes multi-agents, la communication directe est une forme d'interaction entre les agents qui consiste en l'envoi d'acte de communication et qui repose sur l'intention de communiquer des agents. En pratique dans les SMA l'agent connaît le contenu et le destinataire du message ainsi que le moment où l'envoyer. Mais si l'on se place dans des SMA ouverts c'est-à-dire où les agents sont hétérogènes, un agent doit pouvoir s'adapter en apprenant à communiquer en fonction de ses besoins sans avoir de connaissances préalables sur ce qu'il peut communiquer ni quand le communiquer.

Il existe de nombreux travaux sur l'apprentissage d'actions pour un agent, notamment des méthodes d'apprentissage par renforcement. L'apprentissage par renforcement est une technique permettant à un agent d'apprendre par essai/erreur à choisir la meilleure action selon la situation dans laquelle il se trouve [SB98]. Il se résout facilement par des algorithmes tel que le Q-Learning [Wat89] qui repose sur le modèle mathématique des processus décisionnels de Markov [Put94]. Or si l'on se réfère à la théorie des actes de langages [Aus62, Sea85] sur laquelle reposent les langages de communication des systèmes multi-agents, un acte de communication est assimilable à une action dans le sens où celui ci produit des effets sur son destinataire. Par conséquent il est envisageable d'utiliser l'apprentissage par renforcement pour apprendre à communiquer. C'est pourquoi dans ce papier nous proposons un algorithme d'apprentissage par renforcement pour les actes de communication.

Cependant cette approche pose des difficultés dans le cadre des SMA. Pour commencer, peu de travaux ont étudiés le problème de l'apprentissage d'actes de communication. Par exemple, [Tan93, Mat98] ont utilisé la communication dans leur modèle d'apprentissage pour traiter des états cachés, mais ils utilisent une communication dirigée (l'agent sait ce qu'il doit com-

muniquer et à qui). L'apprentissage des actes de communication eux mêmes (dans le sens de quoi, quand et à qui) n'a pas été encore étudié dans le cadre des PDMs.

Le second problème dans les PDMs est que les actions sont toutes considérées comme atomiques : elles sont exécutées sur un pas de temps d'exécution et leur effets sont immédiatement disponible à l'agent ou à l'environnement. Au contraire la communication est un exemple typique d'action à durée variable, principalement parce que les SMA sont asynchrones. Quand un agent demande à un autre agent d'exécuter une action, il n'a pas de garantie que celui ci va l'exécuter immédiatement, même si celui ci a répondu qu'il acceptait d'effectuer cette tâche. Pour résoudre ce problème une solution est d'utiliser les Processus Décisionnel Semi-Markoviens (PDSMs) qui permettent de modéliser des PDMs avec des actions à durée variable [Put94, Ber95]. Un avantage des PDSMs, comme l'a montré [BD95], est que les algorithmes classiques d'apprentissage par renforcement peuvent être étendus aux PDSMs.

Cependant, l'utilisation des PDSMs n'apparaît pas comme suffisante, parce que la propriété de Markov n'est plus valide dans un SMA asynchrone. Bien que beaucoup de travaux admettent que cette propriété n'est pas obligatoire pour que l'algorithme d'apprentissage converge vers la solution, il existe des heuristiques pour réduire la "non-markovité" quand l'algorithme ne converge pas. Une solution proposée est de modéliser le problème en utilisant les PDMs Partiellement Observables (PDMPOs[CKL94]), c'est-à-dire des PDMs dans lequel l'agent a accès à seulement une partie du problème. Les PDMPOs permettent de prendre en compte le fait que les autres agents agissent sur l'environnement en représentant leur actions comme des états cachés à l'agent apprenant. Par définition les algorithmes d'apprentissage par renforcement fondé sur le modèles des PDMs ne convergent pas dans le cadre des PDMPOs [SJJ94]. Cependant, [Mcc96, DS03, Dut00] ont montré qu'il est possible de convertir un PDMPO en PDM en utilisant la mémoire de l'agent. Comme nous le montrerons plus tard cette solution est adaptée au cadre de l'apprentissage d'actes de communication.

Dans ce papier, nous proposons un algorithme d'apprentissage d'actes de communication qui permet de prendre en compte l'observabilité partielle du monde et l'asynchronisme lié aux SMA en utilisant de la mémoire et des PDSMs.

La suite de cet article est organisée de la façon suivante. Dans la section 2 nous rappelons brièvement la forme d'un acte de communication ainsi que l'utilisation de la mémoire pour résoudre un PDMPO. Dans la section 3 nous présentons notre algorithme et enfin dans la section 4 nous résumerons nos travaux avant d'évoquer les perspectives de notre travail.

2 Existant

2.1 Nomenclature

Dans cet article nous notons $\mathcal{A}$ l'ensemble des agents. Chaque agent $Agt \in \mathcal{A}$ est constitué d'un ensemble d'actions et d'un ensemble de données : $Agt = \langle A_{Agt}, D_{Agt} \rangle$. L'ensemble des données D_{Agt} est l'ensemble des couples $\langle var, val \rangle$, où var est le nom de la donnée et val sa valeur. Une variable peut représenter soit une donnée observée par l'agent dans l'environnement, soit une croyance acquise suite à une interaction avec un autre agent.

L'état d'un agent est caractérisé par l'ensemble des données contenu dans D_{Agt}, il sera noté s dans la suite de cet article et l'ensemble des états possibles d'un agent sera noté S_{Agt}. Nous noterons aussi V_{Agt} l'ensemble des symboles de variables associées.

Une action est un tupple $a = \langle name, \mathcal{P}, \mathbf{E} \rangle$ où :
- $name$ est le nom de l'action. Il est unique pour un agent dans le sens où deux actions du même agent ne peuvent avoir le même nom.
- $\mathcal{P}$ est l'ensemble des préconditions nécessaires pour rendre valide l'action. Ces préconditions sont construites à partir de l'ensemble des données de l'agent D_{Agt}. Ces préconditions doivent être vraies pour que l'action puisse être exécutée par l'agent. Nous dirons qu'une action est possible, si l'ensemble de ses préconditions est vérifié, impossible sinon. La syntaxe précise des préconditions n'est pas importante pour la compréhension de cet article. Seul l'ensemble des variables impliquées dans la précondition sera utilisé dans nos algorithmes.
  Nous notons $vars(p)$ l'ensemble des variables impliquées dans la précondition $p \in \mathcal{P}$. Ces variables seront utilisées dans notre algorithme et nos protocoles.
- $\mathbf{E}$ est l'ensemble des effets de l'action, c'est-à-dire les propositions positives ou négatives

construites sur l'ensemble des données de l'agent D_{Agt} qui seront vraies après l'exécution de l'action. Comme pour les préconditions, nous notons $vars(e)$ l'ensemble des variables utilisées pour définir un effet donné $e \in \mathbf{E}$.

Soit s l'état de l'agent, nous notons $A_{Agt}^{(s)}$ l'ensemble de toutes les actions possibles dans cet état. Outre les actions de l'agent A_{Agt}, nous construirons un ensemble d'actes de communication qui sera considéré comme un ensemble d'actions possibles pour l'agent dans notre algorithme.

2.2 Actes de communication

Dans notre modèle un acte de communication est un message au sens de FIPA [FIP02, FIP97]. L'apprentissage du langage et de l'ontologie étant un problème à part entière nous ne le traiterons pas ici et nous considérerons que les agents utilisent le même langage de contenu et la même ontologie. Pour la compréhension de cet article, nous supposerons que le langage de contenu est une logique propositionnelle d'ordre 0 utilisant les opérateurs $\neg, \cup, \cap, =$ sur deux ensembles d'atomes (les variables V_{Agt} et les actions des agents A_{Agt}) et un ensemble de fonctions arithmétiques. Un contenu de message est donc un ensemble (éventuellement vide) de propositions. Dans la plupart des cas, les propositions seront réduites à un atome (c'est-à-dire un symbole appartenant à l'ensemble $A_{Agt} \cup V_{Agt}$). Enfin, pour alléger les notations nous ne représenterons pas les paramètres de contrôle des messages (reply-with, conversation-id).

Un message est donc caractérisé par un tupple $\langle snd, p(rcv, c)\rangle$, avec $snd \in \mathcal{A}$ l'émetteur du message , $rcv \in \mathcal{A}$ son destinataire, p le performatif utilisé et c le contenu du message dont la nature (c'est-à-dire les éléments propositionnels) dépend de p.

Dans notre étude, nous nous limiterons à la communication pour demander de l'information sur des états ou demander à un autre agent de faire quelque chose. Dans ce but nous proposons d'utiliser deux performatifs différents : query et request.

Query. Les messages de la forme

$$m = \langle snd, query(rcv, v)\rangle$$

permettent à l'agent snd de demander à l'agent destinataire rcv la valeur de la variable $v \in D_{snd}$. Il correspond au message suivant dans la norme FIPA :

$$\langle snd, query{-}ref(rcv, Ref(x).value(rcv, v, x))\rangle$$

où la proposition $value(rcv, var, val)$ est vraie si et seulement si $(var, val) \in D_{rcv}$. L'agent rcv peut alors répondre avec soit :

- $\langle rcv, inform(snd, v = v_0)\rangle$ pour affirmer que $(v, v_0) \in D_{rcv}$, comme il est défini dans FIPA.
- $\langle rcv, unknown(snd, v)\rangle$ pour affirmer que la variable v n'appartient pas à l'ensemble D_{rcv}. Cela correspond selon la norme FIPA au message suivant : $\langle rcv, not - understood(snd, \langle snd, m\rangle)\rangle$ avec m le message initialement envoyé.

Request. Les messages de la forme

$$m = \langle snd, request(rcv, a)\rangle$$

permettent à l' agent snd de demander à l'agent rcv d'exécuter l'action a, représentée par son nom[1]. Sa sémantique suit la définition selon la norme FIPA du performatif ***request***. L'agent rcv peut alors répondre à l'aide d'un des trois messages suivants :

- $\langle rcv, agree(snd, a)\rangle$ pour répondre que l'agent rcv accepte d'exécuter l'action a
- $\langle CV, impossible(snd, pe)\rangle$ pour répondre que l'agent rcv ne peut pas exécuter l'action car au moins une des préconditions de cette action est fausse. pe est l'ensemble de ces préconditions échouées. Cela se traduit en norme FIPA par le message suivant : $\langle rcv, refuse(snd, (\langle rcv, a\rangle, pe))\rangle$
- $\langle rcv, not - understood(snd, a)\rangle$ pour répondre que l'agent rcv ne peut pas exécuter l'action a puisqu'il ne connaît pas cette action ($a \notin A_{snd}$). Ce qui correspond au message FIPA suivant : $\langle rcv, not - understood(snd, m)\rangle$ où m est le message initial.

2.3 Q-Learning

Le Q-Learning [Wat89] est un algorithme d'apprentissage par renforcement qui repose sur l'utilisation d'une fonction de valeur $Q : S_{Agt} * A_{Agt} \longrightarrow \mathbf{R}$ où $Q(s, a)$ correspond à la récompense attendue par l'agent $Agt \in \mathcal{A}$ lorsqu'il se trouve dans l'état s et qu'il exécute l'action a.

[1] Dans ce papier, le nom d'une action comprend à la fois la référence de l'action ainsi que la valeur de ses paramètres. Par conséquent les actions prendre(a) et prendre(b) sont deux actions différentes

La meilleure action pour chaque état s est alors définie par : $a^* = argmax_{a \in A_{Agt}} Q(s,a)$.

Le principe du Q-Learning est de construire itérativement la fonction Q pour chaque couple (s,a) en expérimentant les actions dans l'environnement. A chaque prise de décision (c'est à dire choix d'une action à effectuer), l'agent choisit une action a dans $A_{Agt}^{(s)}$, exécute a, reçoit une récompense $r(s,a)$ et observe son nouvel état s'. Puis il met à jour la valeur Q(s,a) en fonction de s' et de $r(s,a)$ obtenue selon la formule suivante :

$$Q(s,a) = (1-\alpha_t)Q(s,a)+\alpha_t(r+\gamma \max_{a \in A} Q(s',a'))$$

où $alpha_t \in [0,1]$ est le taux d'apprentissage et γ est le facteur d'actualisation (Il permet de modéliser les préférences de l'agent en matière de récompense. Si l'agent préfère une récompense immédiate γ sera proche de 0, si au contraire toutes les récompenses sont importantes pour lui γ tendra vers 1).

La stratégie de sélection utilisée pour l'action à exécuter est la température de Boltzman où la probabilité de sélectionner l'action a dans l'état s est égale à :

$$P(a_t = a | s_t = s) = \frac{e^{Q(s,a)/T_t}}{\sum_{b \in A} e^{Q(s,b)/T_t}}$$

Avec T_t le paramètre de température qui décroît doucement en fonction du temps t. Lorsque la température au départ est très élevée alors la probabilité de choisir une action a ne dépend pas de $Q(s,a)$. Elle est donc uniforme. Au contraire lorsque la température est plus faible et proche de 0, la probabilité de choisir l'action a dépend de la valeur de $Q(s,a)$. Ainsi une action avec un Q élevé aura plus de chance d'être choisie.

Une des limites du Q-Learning par rapport à notre problème est qu'il est fondé sur le modèle des PDMs dans lesquels chaque action est exécutée sur un pas de temps. Au contraire les SMA sont généralement asynchrones. Ainsi les actes de communication prendront plus d'un pas de temps à s'exécuter.

2.4 Utilisation de la mémoire

La mémoire d'un agent consiste pour un agent à se souvenir de ses actions et de ses observations passées. Dans un PDM il n'existe pas de notion de mémoire car la propriété de Markov est vérifiée :

$$\forall s' \in S_{Agt} \qquad P(s_{t+1} = s' | s_t, a_t) = P(s_{t+1} = s' | s_t, a_t, s_{t-1}, .., s_1, a_1)$$

L'état de l'agent ne dépend que de l'état précédent et de la dernière action exécutée. Dans un PDMPO les techniques de reconnaissance d'état doivent utiliser de la mémoire afin de distinguer deux états différents qui correspondent à la même observation selon le point de vue de l'agent [Ber95]. Au lieu de définir son état sur ses seules observations, l'agent utilise aussi sa mémoire qui contient ses actions et ses observations passées. Si l'agent utilise assez de mémoire alors le problème peut redevenir markovien. Cependant l'utilisation de la mémoire peut conduire à une explosion combinatoire due à une augmentation du nombre d'états. La méthode proposée par [Mcc96, DS03, Dut00] est de trouver la taille minimale de la mémoire à utiliser pour que l'algorithme d'apprentissage converge.

L'idée générale est d'utiliser un arbre de décision. Cet arbre a pour feuilles les états possibles de l'agent c'est à dire les informations qu'il doit utiliser pour prendre ses décisions. Chaque branche de l'arbre correspond à une distinction introduite afin de distinguer du noeud père les états cachés. Les branches partant de la racine représentent une distinction sur les observations actuelles de l'agent, puis le second niveau de branche correspond à une distinction sur la dernière action effectuée, le troisième niveau à une distinction sur la dernière observation de l'agent et ainsi de suite comme le montre la figure 1.

L'arbre se construit itérativement :
- Au départ l'agent n'utilise pas de mémoire. Il construit la première série de fils (qui correspondent aux noeuds de profondeur 1) en utilisant les états qu'il observe.
- Après Niter expériences (Niter est fixé expérimentalement), on détermine les N états les plus ambigus. On ajoute alors un cran de mémoire à l'agent qui va lui permettre de mémoriser la dernière action qu'il a effectuée. L'état de l'agent est alors caractérisé par $S + M_1$ et on recommence l'apprentissage, mais on ne prend en compte M_1 que dans les états ambigus (dans les autres états on continue le Q-Learning classique avec S)
- On itère ce processus. Après k étapes , la mémoire M_k (utilisée pour les noeuds ambigus) contient k éléments : S , action, S , action...

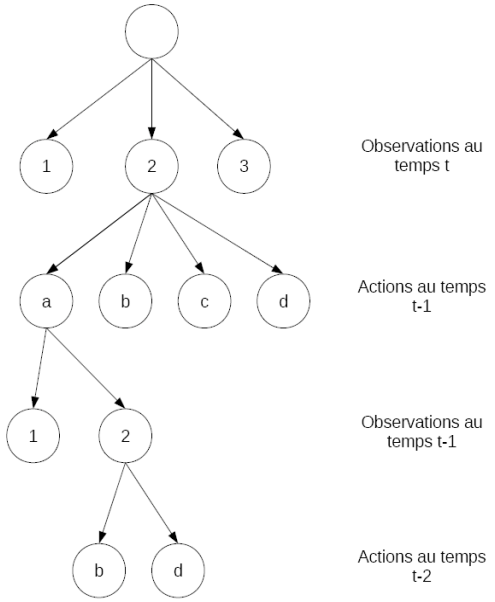

FIG. 1 – Un arbre de décision : Les observations de l'agent sont représentées par des entiers et les actions par des lettres.

Afin de déterminer quand introduire une nouvelle distinction sur la mémoire, [DS03] propose d'utiliser les trois critères suivants :

Ambiguïté sur l'action optimale : Un état s est plus ambigu si l'action optimale a_s^* n'est pas clairement définie, car les valeurs des $Q(s,.)$ des deux meilleures actions sont très proches.

Nombre de mises à jour de la fonction Q : Un état s est plus ambigu qu'un autre état si sa fonction $Q(s,.)$ est mise à jour plus souvent. En effet un noeud mis à jour souvent correspond à un noeud souvent rencontré donc susceptible de correspondre à plusieurs états cachés.

Convergence des Q(e,.) : [DS03] ont noté que la convergence de la fonction Q est plus lente pour un état ambigu que pour un état non ambigu. Par conséquent, les auteurs enregistrent pour chaque action a la dernière variation de la valeur $Q(s,a)$ et l'état s est considéré plus ambigu quand les variations $Q(s,.)$ sont importantes.

Chaque état est classé selon chaque critère et les auteurs considèrent la somme des rangs obtenus pour définir les éléments les plus ambigus. Ensuite les N états les plus ambigus sont alors distingués en ajoutant un cran de mémoire.

L'algorithme se termine soit parce que la mémoire de l'agent a atteint une valeur seuil, soit parce que la différence entre les performances des deux derniers arbres n'est pas significative. La performance d'un arbre correspond à la récompense moyenne obtenue par l'agent lorsqu'il utilise la politique induite par l'arbre. Cette solution semble très intéressante pour traiter des problèmes non-markoviens. Cependant elle nécessite que l'agent connaisse l'ensemble des actions qu'il peut effectuer pour chaque état s. Dans notre problème ce n'est pas le cas, l'agent devant construire ses actes de communication au fur et à mesure de ses besoins.

3 Apprentissage des actes de communication

Nous présentons dans cette section notre algorithme d'apprentissage d'actes de communication. Notre solution prend en compte les hypothèses d'observabilité partielle et d'asynchronisme liées aux SMA. Premièrement nous montrons comment l'agent peut déterminer les contenus des messages, en utilisant des protocoles simples. Ensuite nous présentons notre mécanisme d'apprentissage par renforcement des actions et des actes de communication de l'agent. Nous montrons alors comment les agents peuvent apprendre à attendre les effets de leurs messages directifs. Enfin nous montrons comment notre algorithme peut être adapté pour traiter l'hypothèse de non-markovité en désambiguïsant les observations de l'agent.

3.1 Construction des actes de communication

Dans cet article, nous avons choisi dans un premier temps de nous limiter à la construction de messages d'acte *query* et *request*. La première difficulté de l'apprentissage par renforcement est de déterminer les contenus qui ne sont pas donnés à priori dans un SMA ouvert. Pour cela nous proposons de déterminer deux protocoles d'interaction simples qui reposent sur deux hypothèses. Premièrement les agents sont coopératifs dans le sens où ils effectuent les actions qui leur sont demandées via un message *request* (sous la condition que les préconditions de l'action soient vérifiées) et ils répondent correctement (ils ne mentent pas) lorsqu'ils connaissent la réponse d'un message *query*. Deuxièmement les agents sont capables d'introspection : ils sont capables de raisonner sur leurs actions et leurs données. L'idée générale est alors que l'agent va d'abord " explorer " le SMA à l'aide des

deux protocoles afin de construire un maximum d'actes de communication et tester les messages qu'il en déduit. Le dilemme exploration/exploitation est géré via un facteur δ qui tend à décroître lorsque l'exploration ne conduit plus à un résultat. Ce facteur est différent de la température utilisée pour déterminer le choix de l'action à utiliser dans un état. Le facteur δ lui détermine si l'agent doit continuer à construire des actions ou apprendre à utiliser ses actions pour un état donné.

Construction des messages *Request*. Pour construire les contenus des messages *request*, nous utilisons le protocole *what-order* illustré sur la figure suivante 2. Le performatif

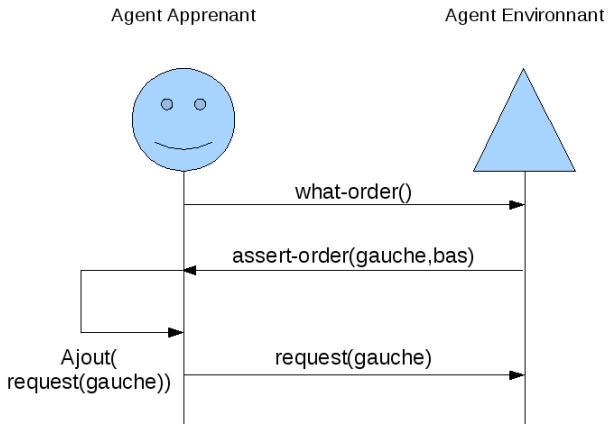

FIG. 2 – Protocole what-order

what-order, qui est utilisé avec un contenu vide, a été proposé dans le cadre de travaux antérieurs [CS07, Sab02]. Le message $\langle Agt, what\text{-}order(d, \emptyset)\rangle$ correspond au message suivant selon la nomenclature FIPA : $\langle snd, query(rvc, Ref(x).PossAct(x, rcv))\rangle$ où $PosAct(x, rcv)$ est vraie si x est une action possible de l'agent rcv au moment de la réception du message. Ce message permet à snd de demander à l'agent rcv l'ensemble de ses actions possibles.

La réponse attendue est un message de la forme :

$$m' = \langle d, assert - order(Agt, \langle a_1, .., a_n\rangle)\rangle$$

où $\forall i \quad a_i \in A_{rcv}$. A partir de cette réponse l'agent snd peut construire un ensemble d'actes de communication. $\forall a_i \in contenu(m\_rep)$, le message $\langle snd, request(rcv, a_i)\rangle$ est un acte de communication envisageable. Du point de vue de l'implémentation l'agent ajoute simplement ces actions à son ensemble d'actions $A^{(s)}_{Agt}$ (s étant l'état de l'agent snd au moment où le protocole *what-order* a été initié).

Construction des messages *Query*. Nous construisons le contenu des messages *query* en utilisant le contenu des messages *impossible* lors de l'échec d'un message *request* :

$$m' = \langle rcv, impossible(snd, pe)\rangle$$

Ce message (comme nous l'avons montré dans la section 2) contient pe l'ensemble des préconditions de l'action demandée qui ont échoué. Chaque précondition $p \in pe$ contient un ensemble de variable $vars(p)$ qui peuvent être utilisées pour construire de nouveaux messages *query* : $\forall p \in pe$, $\forall v \in vars(p)$, le message $\langle snd, query(rcv, v)\rangle$ est un nouvel acte de communication envisageable (l'agent snd peut l'ajouter à l'ensemble des actions $A^{(s)}_{Agt}$). L'utilisation des messages de performatif *impossible* est illustrée sur la figure 3.

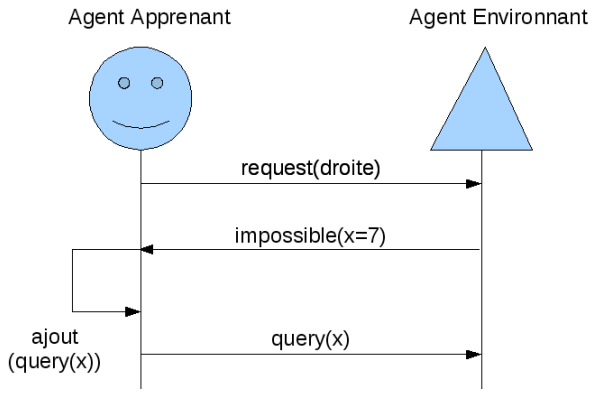

FIG. 3 – Utilisation d'un message de performatif impossible pour construire un nouvel acte de communication

Algorithme d'exploration. A chaque prise de décision de l'agent, une variable $Rand$ est générée aléatoirement :

– Si $Rand < \delta_s$ avec s l'état de l'agent, l'agent utilise un protocole *what-order* pour déterminer de nouveaux actes de communication[2]. La valeur du paramètre δ_s est immédiatement corrigée en fonction de la réponse obtenue. Si l'agent reçoit un message *assert-order* contenant au moins une nouvelle action alors la valeur de δ_s est augmentée : $\delta_s = \delta_s * 2$ (on notera que δ_s étant une probabilité nous limitons sa plus haute valeur à 1). Sinon, δ_s est décrémenté : $\delta_s = \delta_s * 0.8$

[2]Le receveur du message est choisi selon une distribution uniforme

– Si $Rand \geq \delta_s$, l'agent choisit une action à effectuer parmi ses actions (actions propres et actes de communication) en utilisant la température boltzmanienne T.

3.2 Mécanisme d'apprentissage

Afin que l'agent apprenne à utiliser ses actions c'est à dire ses actions propres ainsi que les actes de communication qu'il a construit, nous utilisons l'algorithme de Q-Learning classique. Cependant cette approche pose trois problèmes.

Premièrement dans un SMA les agents sont asynchrones, les temps de réponse aux actes de communication peuvent être variables. Par conséquent les actions de l'agent apprenant ont des temps d'exécution différents. Or les formalismes des PDMs et des PDMPOs considèrent les actions comme s'exécutant sur un pas de temps fixe et identique pour toutes les actions de l'agent. C'est pourquoi nous utilisons le formalisme des PDSMs qui permet de modéliser un PDM où les actions ont des durées variables d'exécution et se résout via un algorithme de Q-Learning modifié prenant en compte le temps d'exécution de l'action.

Deuxièmement il faut anticiper l'absence de réponse des agents (dû soit à la perte des messages, soit à l'autonomie des agents dans les SMA qui peuvent prendre la décision de ne pas répondre). Pour éviter que l'agent n'attende une réponse indéfiniment, nous utilisons un time-out au delà duquel l'agent considère le message comme perdu.

Troisièmement la réception d'un message d'acte *agree* signifie que l'agent émetteur de la réponse accepte d'effectuer l'action demandée et non qu'elle est finie d'être exécutée. Dans le protocole FIPA-Request, ce problème est résolu par l'utilisation d'un message de performatif *confirm* pour indiquer que l'action est terminée [FIP03]. Cependant cette approche possède plusieurs limites. D'une part cette solution requiert l'envoi de nouveaux messages (lesquels peuvent être coûteux). D'autre part, l'utilisation d'un message *confirm* dans notre algorithme impliquerait que l'agent ne pourrait rien faire jusqu'à ce qu'il ait reçu la confirmation que l'action demandée a été effectuée ce qui est en totale contradiction avec le paradigme SMA. De plus ce message pourrait se perdre.

Utilisation des PDSMs. Un processus décisionnel semi markovien (PDSM) est un PDM dans lequel une action peut durer plus d'un pas de temps [Ber95, Put94]. On le définit par un quintuplet $\langle S, A, p, r \rangle$ où :
– S est un ensemble d'états.
– A est un ensemble d'actions.
– $P(\tau, s'|s, a)$ une fonction de transition qui décrit la probabilité de se trouver dans l'état s' au bout de τ pas de temps si l'agent est dans l'état s et qu'il effectue l'action a.
– $r : S*A \Longrightarrow \mathbb{R}$ est la fonction de récompense.
L'algorithme du Q-Learning peut alors s'appliquer en utilisant une formule adaptée. En effet la mise à jour de la fonction Q(s,a) ne se fait pas seulement en fonction du nouvel état s' et de la récompense finale obtenue, mais en tenant compte de chaque récompense obtenue à chaque pas de temps compris entre le début et la fin de l'action [BD95]. Cependant chacune de ces récompenses est modérée en fonction du temps mis pour l'obtenir. Ainsi les récompenses obtenues en début d'action sont plus intéressantes que celles obtenues en fin d'action ceci visant à privilégier entre deux actions obtenant les mêmes récompenses la plus rapide. La mise à jour de la valeur $Q(s, a)$ se fait alors selon la formule suivante :

$Q(s_t, a) = (1 - \alpha_t) * Q(s_t, a) + \alpha_t * [\gamma^k \max_{a' \in A'(s_{t+k})} Q(s_{t+k}, a') + r_{t+1} + \gamma r_{t+2} + .. + \gamma^{k-1} r_{t+k}]$ où :
– k est le nombre de pas de temps nécessaire à l'exécution de l'action
– $A'(s_{t+k})$ est l'ensemble des actions au temps $t + k$
– r_t correspond à la récompense obtenue au temps t
– Les paramètres α_t et γ sont identiques à ceux utilisés dans le Q-Learning classique présenté dans la section 2.3

Gestion du time-out. Afin de gérer le problème de non-réponse aux actes de communication nous utilisons un time-out. Si l'agent ne reçoit pas de réponse au delà de cette limite, il stoppe l'action et il ne modifie pas la Q-Value de ce message, parce qu'il suppose que son message a été perdu.

A chaque prise de décision, l'agent choisit une action a à effectuer et le compteur k qui stocke le nombre de pas de temps depuis lequel l'agent a exécuté son action est initialisé à 0. A chaque pas de temps, k est incrémenté de 1 jusqu'à ce que :
– k>TLIM. Dans ce cas l'agent arrête son action.

– L'action a est terminée (une réponse a été obtenue). L'agent modifie alors la valeur $Q(s, a)$ (où s est l'état de l'agent lorsqu'il a lancé l'action a).

Apprendre à attendre. Lorsque l'agent reçoit un message de performatif *agree*, cela n'implique pas que l'action demandée est terminée mais seulement que l'agent questionné accepte d'exécuter l'action demandée. Cependant notre agent apprenant ne peut pas utiliser la solution proposée par FIPA [FIP03] basée sur un message de performatif *confirm* pour plusieurs raisons. Premièrement, les messages *confirm* peuvent être coûteux. Deuxièmement, il faudrait utiliser un deuxième time-out pour gérer le problème de non réponse de la part de l'agent. Troisièmement, dans beaucoup de situations, si les effets intéressants de l'action se produisent avant que le message *confirm* soit réceptionné (soit parce que l'action est complexe et nécessite plusieurs sous effets dont seulement certains intéressent l'agent apprenant, soit parce qu'un autre agent change l'état de l'environnement entre temps), notre agent perdra un temps précieux à attendre un message *confirm* inutile. Il devrait au contraire utiliser ses observations et lancer une autre action.

Pour ces raisons, nous proposons d'utiliser une autre solution : une action *wait* = $\langle wait, \emptyset, \emptyset \rangle$ qui est ajoutée à l'ensemble des actions possibles de l'agent et qui lui permet d'attendre les effets des actions *request*. Notre agent devra donc non seulement apprendre à agir et à communiquer mais aussi à attendre, en utilisant le Q-Learning adapté aux PDSMs (voir la section 3.2.1). Dans notre algorithme, nous donnons un coût nul à l'action *wait* et un coût négatif à l'envoi d'un message[3] afin de privilégier l'attente à l'envoi d'un message. Enfin l'utilisation de l'action *wait* contrairement à l'utilisation du message *confirm* permet à l'agent de lancer d'autres actions (au lieu d'attendre la fin de celle qui est lancée) et ainsi permettre que le système exécute plusieurs actions en parallèle.

3.3 Gestion de la non markovité

De façon générale dans un système multi-agents la propriété de Markov n'est plus respectée. D'une part parce que les agents ne peuvent toujours observer totalement l'environnement. D'autre part parce que les autres agents agissent aussi sur l'environnement en effectuant leurs propres actions. Comme nous envisageons le problème d'un seul agent apprenant alors l'ensemble des actions effectuées par les autres agents sont assimilables à des états de l'environnement que l'agent ne peut voir. C'est pourquoi nous pouvons modéliser notre problème par un PDMPO.

Pour faciliter la convergence de l'algorithme du Q-Learning dans notre PDMPO, notre idée est d'utiliser les messages de performatif *query* pour obtenir des informations sur l'environnement et les données des autres agents. Cette information est stockée sous forme de croyance dans notre modèle. Cependant, chaque croyance peut devenir fausse durant le processus d'apprentissage, notre agent doit donc apprendre combien de temps il doit garder une croyance donnée.

Pour faire cela, nous avons choisi d'utiliser l'algorithme d'apprentissage que nous avons introduit à la section 2.4. Le principe général est le suivant : chaque croyance reçue après l'envoi d'un message *query* est ajoutée à l'état de l'agent pour la prochaine prise de décision seulement puis elle est effacée immédiatement après. Cependant, puisque l'état de l'agent inclus ses observations et ses croyances elle n'est pas perdue. Par conséquent en utilisant l'algorithme présenté à la section 2.4, l'agent apprend le nombre de pas de temps depuis lequel il a reçu une croyance au lieu de combien de temps il doit s'en souvenir.

Une autre motivation pour choisir d'utiliser la mémoire de l'agent est qu'elle permet à l'agent de se souvenir de ses actions passées. En utilisant cette information, l'agent peut apprendre, par exemple, qu'un acte de communication *query* est plus pertinent après un acte de communication *request*.

De plus l'utilisation de la mémoire permettra d'éviter des situations de blocage dues à l'utilisation de l'action *wait*. En effet si dans un état s il est pertinent d'attendre car l'agent vient de lancer une action a, cela l'est beaucoup moins si l'agent vient d'attendre au coup précédent. Sans mémoire l'agent apprendra à attendre dans l'état s et pourra donc rester bloqué dans cet état si celui n'est pas modifié par une aide extérieure (action lancée auparavant ou un autre agent modifiant l'environnement). Au contraire en utili-

[3] Ce coût sera fonction de la réponse obtenue, un message obtenant une réponse "positive" (c'est-à-dire la réception d'un message *agree* pour l'envoi d'un message *request* ou la réception d'un message *inform* pour l'envoi d'un message *query*) sera moins pénalisé qu'un message obtenant une réponse "négative"

sant la mémoire l'agent pourra apprendre par exemple à attendre après un acte *request* et pas après une action *wait*. Comme nous venons de le voir un noeud où la meilleure action est *wait* est un noeud très ambigu par conséquent nous prendrons en compte cette information dans le choix des noeuds les plus ambigus.

Algorithme d'utilisation de la mémoire. L'agent apprenant $A_l = \langle Agt, \mathcal{M} \rangle$ est un agent $Agt \in \mathcal{A}$ tel que présenté dans la section 2.1 avec une mémoire $\mathcal{M} = \{a_t, s_t, a_{t-1}, s_{t-1}, ..\}$ où $a_t \in A_{Agt}^{(s_t)}, s_t \in S_{Agt}$.

Comme dans [Mcc96] nous utilisons un arbre représentant les états sur lesquels l'agent va faire son apprentissage. Chaque noeud de l'arbre est représenté par un tupple :

$$Nd = \langle Di, A_{Agt}^{(Nd)}, Q_{Nd}, \Delta Q_{Nd}, UpD_{Nd}, \delta_{Nd} \rangle$$

- Di est une distinction. Elle est égale à la valeur de la branche qui arrive dans le noeud Nd. $Di \in \{S_{Agt}, A_{Agt}\}$.
- $A_{Agt}^{(Nd)}$ est l'ensemble des actions que l'agent peut exécuter quand il est dans l'état représenté par le noeud Nd. Cet ensemble contient des actes de communication et l'action *wait*.
- Q_{Nd} est l'ensemble des valeurs $Q(Nd, .)$. $Q_{Nd} = \{q_a\}$ avec $a \in A_{Agt}^{(Nd)}$
- $\Delta Q_{Nd} = \{\Delta q_a | \Delta q_a(t+1) = |q_a^{t+1} - q_a^t|$ avec $a \in A_{Agt}^{(Nd)}$ et $q_a^t, q_a^{t+1} \in Q_{Nd}\}$.
- UpD_{Nd} est le nombre de mises à jour du noeud Nd, c'est-à-dire le nombre de fois où l'agent a été dans létat représenté par Nd et qu'il l'a mis à jour durant le processus d'apprentissage.
- δ_{Nd} est le paramètre d'exploration. Il définit la probabilité que l'agent apprenant choisisse d'exécuter un protocole *what-order* dans l'état Nd.

L'ensemble des arbres est noté Arb. Un arbre arb est représenté par un tupple $\langle Nd, \mathcal{F}_{arb} \rangle$ où $\mathcal{F}_{arb} \subset Arb$ est l'ensemble des arbres fils de arb.

A la fin de l'étape d'apprentissage, on détecte les N noeuds les plus ambigus. Un noeud sera plus ambigu qu'un autre si sa meilleure action est *wait*. En cas d'ex-aequo, notre algorithme se basera sur l'heuristique de Dutech et al. [DS03] pour départager les noeuds.

- Le taux d'ambiguïté d'un noeud s est déterminé par la fonction suivante :

$$\begin{aligned} ambigu(s) &= wait(s) \\ &\quad + \frac{1}{3}(rang_s(UpD_s) + \frac{1}{|\Delta Q_s|} \\ &\quad + rang_s(\frac{1}{|\Delta Q_s|} \sum_{a \in A_{Agt}^s} \Delta q_a) \\ &\quad + rangD_s(q^{A_s^1} - q^{A_s^2})) \end{aligned}$$

  Avec A_s^1, A_s^2 les deux meilleures actions pour l'état s (c'est-à-dire les actions dont la valeur q est la plus elevée), la fonction $rang_s(x)$ qui renvoie la position du noeud s si on ordonne tous les noeuds dans l'ordre croissant en fonction de x, $rangD_s(x)$ renvoie la position du noeud s si on ordonne tous les noeuds dans l'ordre décroissant en fonction de x et $wait(s)$ qui renvoie 0 si l'action *wait* est la meilleure action pour le noeud s et qui renvoie M (M étant très grand) sinon.
- Les N noeuds les plus ambigus sont ceux qui possèdent le plus petit taux d'ambiguïté :

$$\begin{aligned} Nd\_ambigu &= \{n | feuille(n, arb), \\ &\quad rang_n(ambigu(n)) < N\} \end{aligned}$$

De même que l'algorithme de Dutech et al., nous arrêtons notre algorithme soit parce que la taille maximale de la mémoire de l'agent est atteinte, soit parce que les performances de notre agent ne peuvent pas être améliorées par un nouveau cran de mémoire. Les performances de notre agent sont déterminées par la récompense moyenne obtenue à l'aide de la politique définie par l'apprentissage par renforcement sur les états courants pour un nombre N d'expériences. L'algorithme s'arrête donc lorsque :
$arret() =$
1 Si $t > Mlim$ Ou
$|\frac{1}{N}(\sum_1^N R(arb_t) - \sum_1^N R(arb_{t-1}))| < \epsilon$
0 Sinon.

Avec $R(arb_t)$, la récompense obtenue par l'agent lorsqu'il utilise l' arbre arb_t et $Mlim$ la taille limite de la mémoire de l'agent.

3.4 Implémentation et évaluation en cours

Notre algorithme d'apprentissage a été implémenté sur la plateforme multi-agent VDL[4], laquelle offre une capacité d'introspection adéquate pour notre modèle [Sab02]. Nous avons

[4] http ://www-poleia.lip6.fr :8180/ sabouret/demos/index.html

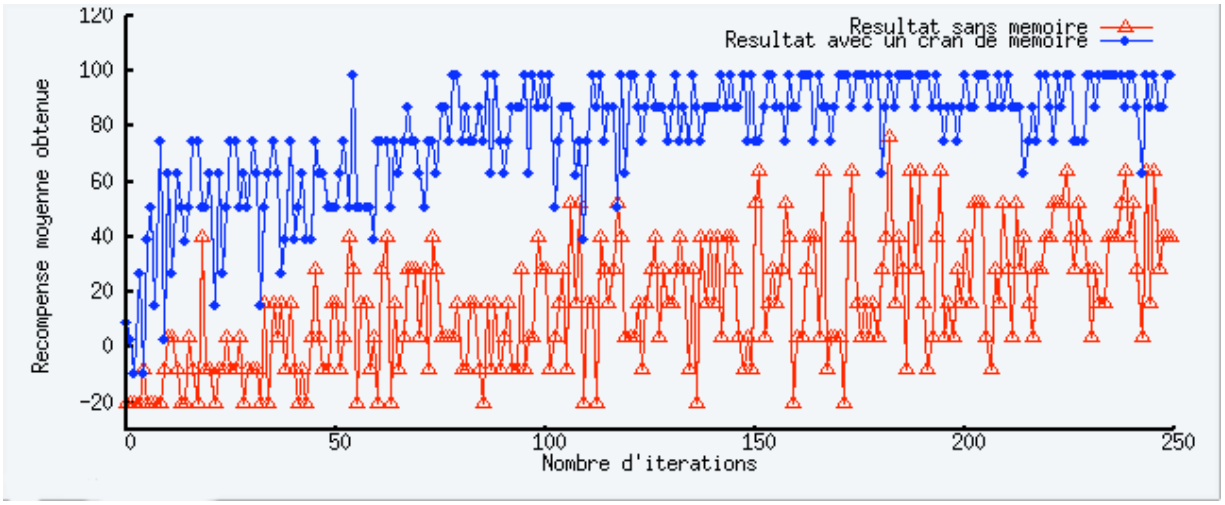

FIG. 4 – Récompense moyenne sur 10 expériences en fonction du nombre d'itérations de l'algorithme pour un agent sans mémoire (triangle) et avec un cran de mémoire (point)

programmé un exemple simple de SMA asynchrone fondé sur le problème de recherche d'un trésor dans un labyrinthe. Un robot se déplace dans un labyrinthe (5*5) afin de trouver un trésor tout en évitant un certain nombre de cases trous (25 % des cases du labyrinthe) pour cela il dispose de 4 actions (haut, droite, gauche, bas). A chaque expérience le labyrinthe n'est pas modifié et le robot est remis sur la même case de départ. L'agent apprenant doit apprendre à diriger le robot afin d'obtenir la récompense. Les deux agents sont asynchrones dans le sens où leurs vitesses d'exécution sont différentes ce qui ne permet pas de garantir que les commandes de l'agent apprenant soient prises en compte immédiatement. Cependant leur vitesse est constante et ne varie pas entre deux expériences. Nous avons choisi cet exemple car il est facile de le complexifier en fonction des modalités de notre algorithme que nous voulons tester (protocole *what-order*, apprentissage de l'action *wait*, utilisation de la mémoire...). Une première expérimentation a permis de valider notre algorithme dans le cas particulier où :

- L'apprenant apprend à utiliser des actes de communication *request* (qu'il a préalablement construit à l'aide du protocole what-order) et l'action *wait*.
- L'apprenant observe totalement son environnement.
- L'agent apprenant s'exécute deux fois plus vite que le robot.
- Les récompenses associées aux états sont les suivantes : le trou apporte une récompense finale de -20 et le trésor rapporte une récompense finale de 100. Les autres états ne rapportent pas de récompense.

Les résultats obtenus sont encourageants comme le montre le graphe ci dessous représentant les récompenses obtenues par l'agent apprenant au cours de son apprentissage en n'utilisant soit aucune mémoire (courbe avec des triangles) ou un cran de mémoire (courbe avec des points). On y voit clairement que l'utilisation d'un cran de mémoire permet à l'algorithme de converger vers une solution où la récompense moyenne finale est de 100, alors que sans mémoire l'agent stagne vers une récompense moyenne finale de 40.

Actuellement nous complétons notre modèle expérimental afin d'introduire la construction des actes *query* et leur utilisation dans un contexte d'observation partielle de l'environnement. Afin de valider nos résultats, nous comparerons notre solution à un algorithme d'apprentissage sans communication tel que celui de [DS03]. En étudiant la convergence de l'algorithme, le coût de la communication et le gain résultant de la politique trouvée, nous voulons démontrer que l'apprentissage d'actes de communication améliore vraiment la qualité du processus d'apprentissage. Cette partie est encore en cours de réalisation.

4 Conclusion

Nous nous sommes intéressés dans cette étude à l'apprentissage par renforcement d'acte de communication dans un contexte multi-agents. Dans cette optique nous avons défini trois types de problèmes liés à cette approche : la construction des actes de communication, l'assimilation

des actes de communication à des actions afin de pouvoir leur appliquer des techniques d'apprentissage par renforcement et enfin la résolution des problèmes posés par la non markovité des systèmes multi-agents. Afin de construire les actes de communication nous avons utilisé une solution qui utilise le pouvoir d'introspection des agents et leur capacité à raisonner sur leur actions afin que l'agent apprenant puisse construire les actes de communication dont il a besoin. Ces actes de communication ainsi créés sont alors considérés comme des actions prenant un temps variable afin de leur appliquer un algorithme d'apprentissage par renforcement. Notre algorithme gère le problème de non markovité en employant les informations récupérées par l'agent à l'aide des actes de communication qu'il a créé ainsi qu'en utilisant sa mémoire des actions et des observations passées.

Si les travaux présentés ici permettent d'améliorer l'apprentissage par renforcement dans un contexte multi-agents en proposant un mécanisme d'apprentissage d'acte de communication, ils soulèvent encore de nombreuses perspectives de recherche. Premièrement afin de pallier des problèmes d'explosion combinatoire en mémoires et en calculs, nous avons dû considérer que l'ensemble des actions apprises par notre agent lui étaient utiles comme cela se fait dans la plupart des travaux en apprentissage par renforcement. Dans un cadre plus réaliste, cette hypothèse ne tient plus et par conséquent il serait intéressant de poursuivre les travaux et de définir un mécanisme permettant à l'agent de ne garder en mémoire que les actions utiles à son problème.

Une autre direction de recherche serait de définir une heuristique pour l'utilisation de la mémoire. L'heuristique que nous utilisons est largement inspirée des travaux de [DS03] lesquels ont eu de bons résultats dans un SMA sans communication. En utilisant l'information fournie par la communication (c'est à dire le nombre d'actes de communication ayant échoué pour chaque état de l'agent), nous pensons que l'heuristique pourrait être améliorée. D'autre part il serait intéressant de définir un mécanisme permettant de déterminer de quels types de mémoires (actions ou observations), l'agent a besoin selon l'état où il se trouve.

Enfin nous aimerions par la suite étendre notre recherche à l'ensemble des actes de communication de l'agent afin de bénéficier de la richesse des langages de communication entre agents (ACL).

Références

[Aus62] J. Austin. How to do things with words. *Oxford University Press, Oxford England*, 1962.

[BD95] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 393–400. MIT Press, 1995.

[Ber95] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont , Massachusetts, 1995.

[CKL94] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1023–1028, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[CS07] Yasmine Charif and Nicolas Sabouret. An agent interaction protocol for ambient intelligence. In *2nd International Conference on Intelligent Environnement (IE06)*, 2007.

[DS03] Alain Dutech and Manuel Samuelides. Un algorithme d'apprentissage par renforcement pour les processus décisionnels de Markov partiellement observés : apprendre une extension sélective du passé. 2003.

[Dut00] Alain Dutech. Solving pomdps using selected past events. In *ECAI*, pages 281–285, 2000.

[FIP97] FIPA.org. Fipa communicative act library specification. 1997.

[FIP02] FIPA.org. Fipa acl message structure specification. 2002.

[FIP03] FIPA.org. Fipa request interaction protocol specification. 2003.

[Mat98] Maja J Matari'c. Using communication to reduce locality in distributed multi-agent learning. *Journal of Experimental and Theoretical Artificial Intelligence*, 10 :357–369, 1998.

[Mcc96] Andrew Kachites Mccallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, 1996. Supervisor-Dana Ballard.

[Put94] M.L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc, New York, Etats-Unis, 1994.

[Sab02] Nicolas Sabouret. A model of requests about actions for active components in the semantic web. 2002.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. The MIT Press, 1998.

[Sea85] J. R. Searle. Speech acts : An essay in the philosophy of language. *Cambridge University Press : Cambridge, England*, 1985.

[SJJ94] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *International Conference on Machine Learning*, pages 284–292, 1994.

[Tan93] Ming Tan. Multi-agent reinforcement learning : Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[Wat89] Christopher J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, Cambridge, United Kingdom, 1989.