INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Simon DELECOURT

SMA - Etude bibliographique

MULTI-AGENTS REINFORCEMENT LEARNING

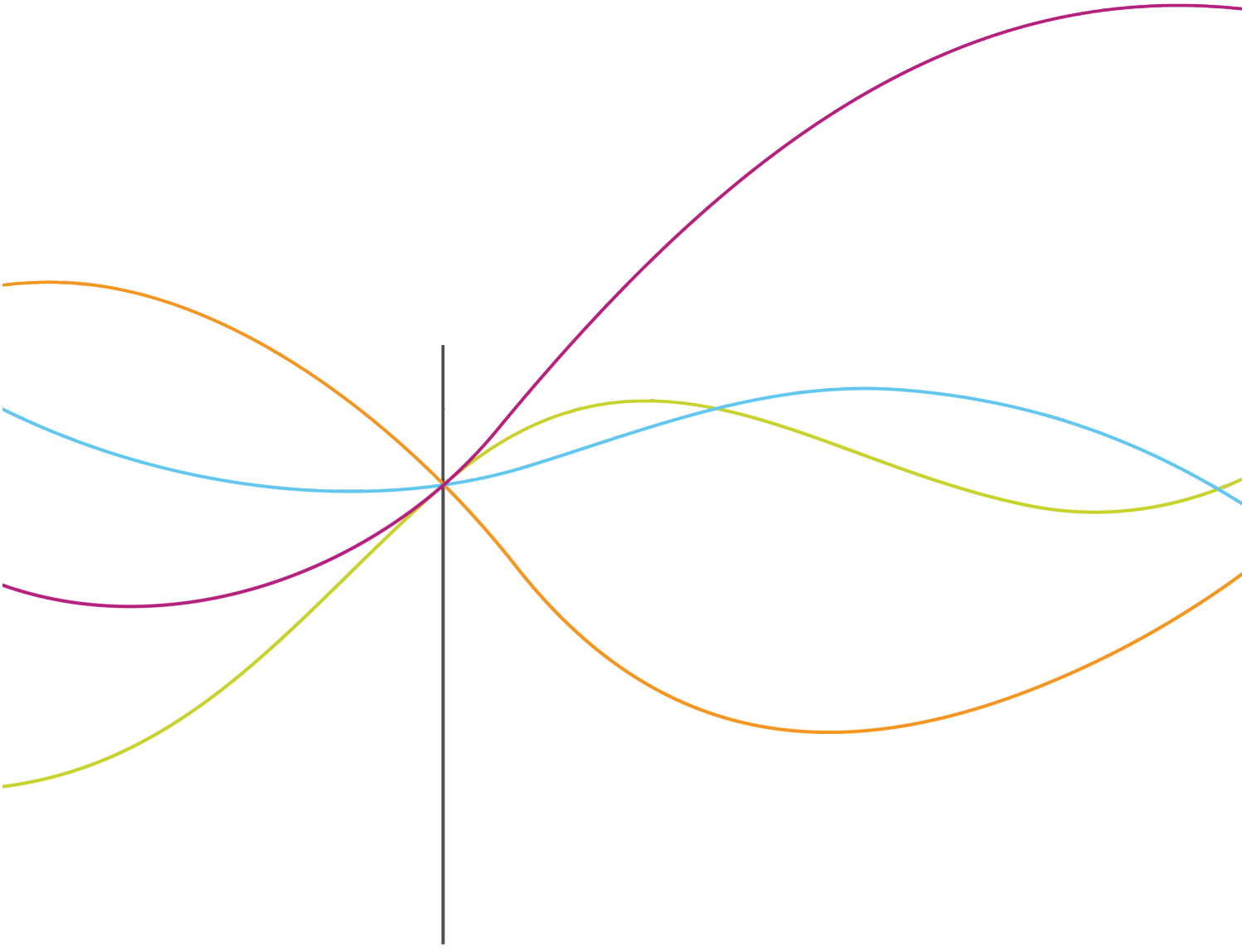


Table des matières

1 Introduction **3**

2 Définitions **4**

3 Cas d'un seul agent **5**

4 Cas de plusieurs agents **8**
4.1 Les différentes classes d'algorithmes : 8
4.2 Fully cooperative 9
4.3 Fully-competitive 12
4.3.1 Algorithme indépendant de l'opposant 12
4.3.2 Algorithmes dépendants de l'opposant 14
4.4 Taches mixtes 15
4.4.1 Algorithmes pour un seul agent 15
4.4.2 Algorithmes agents indépendants 15
4.4.3 Algorithmes agents dépendants 15

5 Les mécanismes de coordinations explicites **16**

6 Applications **17**
6.1 Système de robots 17
6.2 Trading Automatique 17
6.3 Gestion de ressources 17

7 Conclusion **18**

Références **19**

1 Introduction

Cette bibliographie vise à étudier l'usage du reinforcement learning (apprentissage par renforcement en français) comme technique d'apprentissage d'un système multi-agents.

Tout d'abord quelques définitions seront données afin d'avoir les quelques notions nécessaire pour comprendre la suite de la bibliographie.

Le cas particulier d'un système avec un seul agent est présenté dans la section 3. Il sera expliqué comment à partir du formalisme des processus de décision markovien (MDP) il est possible d'aboutir à l'algorithme de Q-learning. Algorithme qui permet d'apprendre une politique de prises d'action. On constatera ensuite les limites de ce type algorithme dans le cadre d'un système avec plusieurs agents. Notamment à cause de l'absence de garantie de convergence, elle même dûe au fait que les agents apprennent en parallèle et du manque de coordination entre eux.

Il sera présenté dans la section 5 quelques mécanismes permettant d'introduire de la coordination entre les agents.

Pour finir, quelques exemples d'applications seront passées en revue.

Cette bibliographie est principalement basée sur le survey "A Comprehensive Survey of MultiAgent Reinforcement Learning" de Busonio et al publié en 2008. Ainsi que de la lecture de quelques papiers décrivant certains algorithmes cités dans cette bibliographie.

2 Définitions

On peut considérer que la première définition de l' **apprentissage par renforcement** est :

> "Reinforcement is the strengthening of a pattern of behavior as a result of an animal receiving a stimulus in an appropriate temporal relationship with another stimulus or a response." Pavlov's Monograph (1927)

Une définition plus récente serait :

> Le reinforcement learning est un domaine de l'apprentissage automatique qui *"consiste, pour un agent autonome (robot, etc.), à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps"* [1].

Tout l'enjeu de l'apprentissage par renforcement est alors de trouver la meilleure politique Π de façon à maximiser la somme des récompenses.

Markov Decision Process (MDP) est une théorie mathématique qui permet de définir formellement l'apprentissage par renforcement. Une MDP est défini par le quadruplet $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}\}$, où :

— $\mathcal{S}$ est l'ensemble des états que peut prendre l'environnement.

— $\mathcal{A}$ est l'ensemble des actions que peut prendre l'agent.

— $\mathcal{T}$ est une fonction de transition d'un état dans un autre état suite à une action. Dans le cadre de transitions déterministes elle prend la forme : $\mathcal{T} : \mathcal{A}x\mathcal{S} \rightarrow \mathcal{S}$.

— $\mathcal{R}$ est l'ensemble des récompenses que peut obtenir l'agent.

A chaque pas de temps, l'agent a connaissance de l'état dans lequel il se trouve (s_t). Il va alors choisir une action parmi l'ensemble des actions possibles dans cet état : $a_t \in A(s_t)$. Ce choix est guidé par la politique $\Pi : S \rightarrow A$ que l'agent a appris. Finalement, l'agent reçoit une récompense $r_t \in \mathcal{R}$ et un nouvel état s_{t+1}. Ce processus est décrit dans la figure (fig. 1).

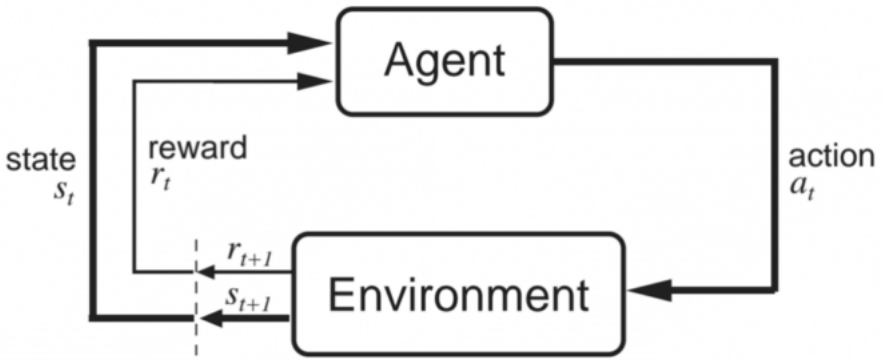


FIGURE 1 – Schéma du fonctionnement d'une MDP

On distingue différents types d'environnement :

— **Environnement déterministe** : dans ce cadre, la fonction de transition T et la fonction de récompense R sont des fonctions déterministes. La fonction T aura la forme suivante : $T : S\,xA \rightarrow S$.

— **Environnement stochastique** : dans un tel environnement, les conséquences de prendre la même action depuis le même état peuvent varier au cours du temps. Ainsi, la fonction prend la forme suivante : $T : S\,xAxS \rightarrow [0;1]$. Avec $\sum_{s' \in S} T(s,a,s') = 1 \quad \forall s \in S, u \in U$ est une fonction de probabilité qui indique la probabilité de passer dans l'état s' sachant que

l'action a a été prise à l'état s.

3 Cas d'un seul agent

La majeure partie des techniques d'apprentissage par renforcement se basent sur la théorie des *Markov Decision Processes* (MDPs).

Dans ce cadre, l'agent a pour objectif de maximiser l'espérance de la somme des discounted rewards (i.e. $E\left\{\sum_{j=0}^{\infty}\gamma^j r_{t+j}\right\}$ où r_{t+j} est la récompense escomptée par l'agent, j tour dans le futur. L'intérêt du coefficient $\gamma \in [0;1]$ est d'avoir un compromis entre récompense immédiate (γ petit) et récompense à long terme (γ grand).

Pour résoudre ce problème d'optimisation formulé par une MDP, il faut définir plusieurs notions :

— ***Value function*** : cette notion représente, pour un état donné, la quantité de reward que l'agent peut espérer obtenir en suivant la politique π. Mathématiquement, cela se traduit par :

$$V^{\pi}(s) = E\left[\sum_{i=1}^{\infty}\gamma^{i-1}r_i\right] \quad \forall s \in \mathcal{S} \tag{1}$$

La *value function* optimale correspond alors à :

$$V^{\pi}(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S} \tag{2}$$

L'objectif final est alors de trouver la politique optimale π^* telle que :

$$\pi^* = \operatorname*{argmax}_{\pi} V^{\pi}(s) \quad \forall s \in S \tag{3}$$

— ***Q value*** : cette notion définie la qualité d'une action depuis un état donnée, $Q : S\,xA \rightarrow R$. On a alors les relations suivantes :

$$V^*(s) = \max_{a} Q^*(s,a) \quad \forall s \in S \tag{4}$$

$$\pi^*(s) = \operatorname*{argmax}_{a} Q^*(s,a) \quad \forall s \in S \tag{5}$$

— **L'équation de Bellman :** cette équation est à la base de plusieurs méthodes pour résoudre une MDP. Elle permet de définir de manière récursive la Q fonction optimale. $Q^*(s,a)$ est égale à la somme de la récompense immédiate plus la somme des récompenses futures *discounted* de tous les états, pondérés par leur probabilité (donné par la fonction de transition), ainsi :

$$Q^*(s,a) = R(s,a) + \gamma E_{s' \in S}[V^*(s')] \tag{6}$$

$$= R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)V^*(s') \tag{7}$$

Parmi les méthodes permettant de résoudre une MPD il existe la méthode ***value iteration***. Celle-ci consiste alors à initialiser Q, V aléatoirement puis à les mettre à jour successivement à chaque itération selon l'algorithme 2, jusqu'à convergence des valeurs Q et V.

Algorithm 1 méthode value iteration [3]

/*initialization*/

$Q[s,a] = 1 \quad \forall s \in S \ \forall a \in A$
$V[s] = 1 \quad \forall s \in S$
repeat
  for *all* $s \in S$ **do**
    for *all* $a \in A$ **do**
      $Q(s,a) \leftarrow E[s'|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
    end
    $V(s) \leftarrow \underset{a}{max} Q(s,a)$
  end
until *V(s) and Q(s,a) converged;*

Cette méthode nécessite la connaissance de la fonction de récompense et de la fonction de transition. On parle d'une méthode **model-based**.

Une méthode très connue pour résoudre une MDP est la méthode dite de ***Q-learning***. Contrairement à l'algorithme vu précédemment, celui est un algorithme ***model-free***, cela signifie que l'agent n'a aucune connaissance de la fonction de transition d'un état à un autre et de la fonction de récompense. L'agent va donc découvrir les bonnes et les mauvaises actions selon l'état dans lequel l'environnement se trouve par essai et erreur (**trial and error** en anglais).

Cette méthode consiste tout simplement à mettre à jour la table des Q values selon l'équation suivante :

$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \cdot \underset{a'}{max} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right] \tag{8}$$

Où $\underset{a'}{max} Q_k(s_{k+1}, a')$ permet de déterminer la meilleure action a' à prendre depuis l'état s_{k+1}. L'expression entre crochet détermine la direction vers laquelle tendre. α_k est le *learning rate*.

Il a été prouvé que ces méthodes convergent vers la solution optimale si les conditions suivantes sont vérifiées :

— Les Q valeurs sont stockées pour chaque pair (action, état) et mis à jour séparément.

— La suite des *learning rates* (α_k) est une suite dont la somme des éléments tend vers l'infini mais la somme des carrés des éléments est finie. (i.e. $\sum_k \alpha_k = \infty$, $\sum_k \alpha_k^2 < \infty$)

— L'agent peut essayer toutes les actions possible pour tous les états. Il n'y a pas de probabilités nulles. Cela signifie que l'agent doit explorer différentes actions avec une certaine probabilité plutôt que d'exploiter la politique en cours d'apprentissage (i.e. compromis **Exploration** vs **Exploitation**).

Le point numéro 1 impose de garder en mémoire une table de Q valeurs de taille (| S | x | A |).
Pour satisfaire la condition 2, on peut choisir un decay learning rate ($\gamma_0 = 1; \gamma_{k+1} = decay * \gamma$ avec $decay \in]0;1[$.
Il existe différentes stratégies d'exploration. L'exploration ϵ-greedy force l'agent, avec une probabilité ϵ à choisir une action $a \in \mathcal{A}$ aléatoirement, sinon l'agent choisit la politique optimale en cours d'apprentissage.

Il existe également la stratégie d'exploration de Boltzmann. Celle-ci consiste à choisir l'action a lorsque l'on est dans l'état s selon la probabilité suivante :

$$h(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}} \tag{9}$$

Le facteur τ contrôle degré d'exploration de la stratégie. PLus τ est grand plus l'exploration est importante, plus τ est petit plus l'agent va exploiter la stratégie optimale.

Les principales difficultés de l'apprentissage par renforcement pour un seul agent est le fléau de la dimensionnalité. En effet, une des conditions de convergence est de stocker les Q-values pour chaque couple (état, action). Or il peut y avoir un très grand nombre d'état dans l'environnement, de même pour les actions. De plus, le compromis exploitation vs exploration est une des difficultés à maitriser afin de s'assurer de la convergence vers la solution optimale. La définition de la fonction de récompense peut également être l'objet de difficulté. En effet pour de nombreuses taches, la récompense n'intervient que lorsque le jeux est terminé. Il est difficile de déterminer des récompenses intermédiaire qui pourraient guider l'agent vers la solution.

4 Cas de plusieurs agents

Les enjeux et les intérêts de considérer plusieurs agents sont multiples. Tout d'abord, on peut distribuer l'apprentissage de chaque agent sur des machines différentes. De plus chaque agent peut bénéficier du partage de connaissances des autres agents. Ce partage de connaissance peut se faire par communication ou par imitation.

Néanmoins, l'apprentissage multi-agents ne résout pas les difficultés du cas d'un seul agent. Le fléau de la dimensionnalité est toujours présent. En effet, le nombre de Q-values à stocker/calculer grandit exponentiellement avec le nombre d'agent.

D'autre part, l'apprentissage multi-agents apporte ses propres difficultés. Tout d'abord, pour certains problèmes il est difficile de déterminer un objectif d'apprentissage pour chaque agent. En effet les objectifs de chaque agent sont très souvent corrélés et ne peuvent donc pas être maximisé indépendamment les uns des autres.

Les politiques obtenues sont souvent non-stationnaires, puisque les agents apprennent simultanément. Ainsi, la meilleure politique d'un agent peut changer quand un autre agent change sa politique.

Enfin, la coordination entre les agents peut s'avérer compliquée et doit être consistente entre les agents afin de réussir leur tache efficacement.

Le cadre d'un système multi-agents nécessite de définir un nouveau concept celui des *MarkovGame*. En effet les MDPs ne permettent pas de résoudre les problématiques soulevées ci-dessous.

Un *Markov Game* (aussi appelé *Stochastic Game* quand la fonction de transition est stochastique) se définit par :

— $\mathcal{S}$ est l'ensemble des états que peut prendre l'environnement.

— $\mathcal{A}$ est une collection d'ensemble d'actions $\mathcal{A}_1$, ..., $\mathcal{A}_k$ un pour chaque agent.

— $\mathcal{T}$ est une fonction de transition d'un état dans un autre état suite aux actions des agents ($\mathcal{T} : \mathcal{A}_1 x ... x \mathcal{A}_2 x \mathcal{S} \rightarrow \mathcal{S}$).

— $\mathcal{R}$ est la collection de l'ensemble des récompenses que peuvent obtenir chaque agent $\mathcal{R}_1$, ..., $\mathcal{R}_k$.

Si $\mathcal{R}_1 = ... = \mathcal{R}_k$ alors tous les agents ont les mêmes récompenses/objectifs, on est donc dans un *Markov Game* collaboratif.
Si k=2, $\mathcal{R}_1 = -\mathcal{R}_2$ alors il y a 2 agents avec des récompenses/objectifs opposées, on est donc dans un *Markov Game* en compétition.
Sinon on parle de jeux mixtes.

4.1 Les différentes classes d'algorithmes :

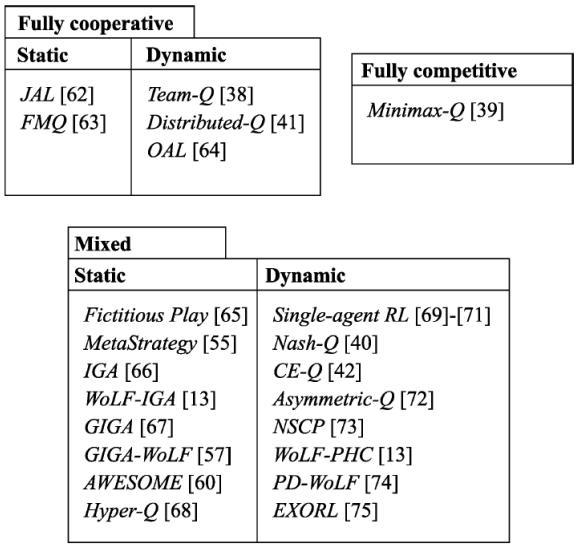
On classe les différents algorithmes selon les taches qu'ils peuvent résoudre. Cette classification prend ainsi en compte le rapport entre les différents agents : entièrement coopératif, entièrement compétitif ou mixte. Puis on distingue les jeux dit statiques et les jeux dynamiques :

— Un jeu **Statique** est un jeu stochastique où l'espace des états S est égal à $\emptyset$. La fonction de récompense ne dépend ainsi que des actions prises. On peut citer le jeu "Pierre feuille

ciseaux" comme exemple d'un jeu statique répété.

— Un jeu **Dynamique** est alors un jeu stochastique où $S \neq \emptyset$

Voici la classification (issu de [5]) des différents algorithmes d'apprentissage par renforcement pour un système multi-agents :



Fully cooperative

Static	Dynamic
JAL [62] *FMQ* [63]	*Team-Q* [38] *Distributed-Q* [41] *OAL* [64]

Fully competitive

Minimax-Q [39]

Mixed

Static	Dynamic
Fictitious Play [65] *MetaStrategy* [55] *IGA* [66] *WoLF-IGA* [13] *GIGA* [67] *GIGA-WoLF* [57] *AWESOME* [60] *Hyper-Q* [68]	*Single-agent RL* [69]-[71] *Nash-Q* [40] *CE-Q* [42] *Asymmetric-Q* [72] *NSCP* [73] *WoLF-PHC* [13] *PD-WoLF* [74] *EXORL* [75]

Fig. 1. Breakdown of MARL algorithms by the type of task they address.

FIGURE 2 – Figure extraite de [5]

4.2 Fully cooperative

Dans un Jeu Stochastique, les agents ont la même fonction de récompense ($\mathcal{R}_\infty = ... = \mathcal{R}_\backslash$). L'objectif est alors de maximiser la somme des discounted rewards de chaque agent. S'il est possible d'utiliser une entité comme controlleur, on parle alors de *joint action learners*. Dans ce cas le problème peut se réduire à la résolution d'une MDP classique via l'algorithme de Q-learning par exemple :

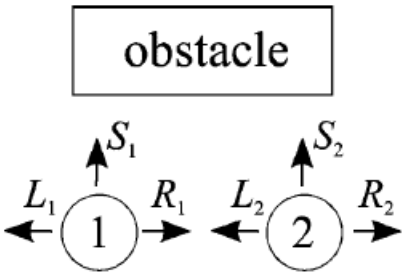
$$Q_{k+1}(s_k, a_k) = Q_k(s_k, a_k) + \alpha_k \left[r_{k+1} + \gamma \cdot \max_{s'} Q_k(s_{k+1}, a') - Q_k(s_k, a_k) \right] \tag{10}$$

Beaucoup d'applications appliquent l'algorithme de Q-learning directement avec succès. Cependant il n'y a aucune garantie théorique de convergence vers une solution optimale. Il apparait plutôt que les agents vont converger vers une politique sous-optimale.

Cependant dans la majorité des cas, les agents sont autonomes. On parle alors de *independant learners* L'utilisation du Q-learning ne permet pas de prendre en compte les actions des autres agents. La solution obtenue par Q-learning serait alors sous-optimale.

L'article [5] illustre ce problème par exemple où 2 agents doivent se suivre pour atteindre une cible. Chaque agent peut effectuer 3 actions : aller tout droit S, tourner à droite (R) et tourner à gauche (L). A gauche de la figure nous avons un aperçu de l'environnement. A droite nous avons la Q-table projetée sur l'état de l'environnement présenté à gauche. Cette table est identique pour les 2 agents. Elle indique que si les agents vont tous les 2 à droite ou tous les 2 à gauche la formation est maintenue et ils ont avancé $Q(L_1, L_2) = Q(R_1, R_2) = 10$. A l'inverse s'ils contournent l'obstacle dans des directions opposées ils se séparent et on donc rempli à moitié l'objectif $Q(L_1, R_2) = 0$ dans les autres cas il y a une collision Q(_,_) est donc négative). Le problème est que les 2 agents, ne communiquant pas, peuvent entrer en collision facilement alors même que leur Q-table semble

leur en empêcher. En effet l'agent 1 peut penser que l'agent 2 va choisir l'action L_2 et choisir en conséquence l'action L_1. Cependant rien ne nous assure que l'agent 2 ne va pas prendre l'action R_2.



Q	L_2	S_2	R_2
L_1	10	-5	0
S_1	-5	-10	-5
R_1	-10	-5	10

Fig. 3. (Left) Two mobile agents approaching an obstacle need to coordinate their action selection. (Right) The common Q-values of the agents for the state depicted to the left.

FIGURE 3 – Exemple issu de [5] illustrant l'importance de la coordination entre les agents

Différents types de méthodes permettent en théorie de résoudre ce problème, dans le cadre d'agents coopératifs :

— **Les méthodes *Coordination-Free*** : ces méthodes ne mettent pas en place un système de coordination mais font quelques hypothèses afin de résoudre le problème mentionné ci-dessus.

Il existe l'algorithme *Team Q-learning* [8]. Cet algorithme fait l'hypothèse que $R_1 = \cdots = R_n$. Ainsi si l'agent 1 reçoit une récompense r alors tous les autres agents vont recevoir exactement la même récompense. Naturellement on a alors $Q_1 = \cdots = Q_n$. On peut alors appliquer l'algorithme de Q-learning en parallèle pour tous les agents.

Les auteurs introduisent également le théorème suivant :

> "***Theorem 6***. *In a multiagent environment where Q-learner and a Q-learner, the Q-learner will converge to optimal behavior if the equilibrium is unique. Furthermore, if the agent follows a GLIE policy of team Q-learners will converge to optimal play if and the limit equilibrium is unique, it will converge in behavior with probability one.* "

Où GLIE est une stratégie d'exploration qui explore l'espace des actions de manière pseudo-aléatoire.

La convergence en comportement d'un agent signifie que si on fait tendre le temps vers l'infini, la distribution de ses actions par rapport aux états devient stable.
L'hypothèse que l'équilibre est unique réfère à l'équilibre de Nash.

L'algorithme ***Distributed Q-learning*** [7] est comme son nom l'indique un algorithme distribué. Cela signifie que chaque agent est bien indépendant des autres (*independant learners*). Dans ce cadre il est impossible pour un agent de calculer une Q-table de la forme

	$a =$		
	a'	a''	a'''
$q^{(1)}(s_0, a) =$	11	7	6
$q^{(2)}(s_0, a) =$	11	7	5

$Q : SxA^m \rightarrow \mathcal{R}$. Cette table permettrait de savoir quel est la meilleure action à prendre pour un agent selon l'état de l'environnement et connaissant toutes les actions possibles des autres agents ainsi que la Q-value associée. L'idée des auteurs est plutôt de calculer des petites Q-tables qui sont en fait des projections de cette grande table centrale.

Les projections sont calculées en effectuant une somme pondéré de la table centrale de la manière suivante :

$$q^j(s,a) = \sum_u (Pr(u|a^j) \cdot (r(s,u) + \beta \cdot \underset{a' \in A}{max} q^j(s', a'))) \tag{11}$$

Où $u = (a^1, ..., a^m)$ ainsi Pr(u | a) est la probabilité du vecteur des actions de tous les agents apparait à l'étape d'après sachant que l'agent j à choisi l'action a^j. C'est en fait l'hypothèse sur les actions des autres agents à l'étape d'après considéré par l'agent j.

Cette projection est dite optimiste en effet chaque agent ne considère que les valeurs maximales selon certains plans dans le calcul des petites q-tables. Cela revient donc à faire l'hypothèse que les agents choisiront la meilleure action.

D'autres types de projections sont possibles. Notamment des projections un peu plus pessimistes où l'on fait l'hypothèse que les autres agents ne prennent pas toujours les meilleures actions.

Voici un exemple issu du papier illustrant le processus de sélection d'une action par l'agent 1 depuis l'état s_0 dans un environnement avec 2 agents au total. L'ensemble des actions possibles est $A = \{a', a'', a'''\}$. L'action de l'agent 1 est notée $a^{(1)}$ et l'action de l'agent 2 est notée $a^{(2)}$. Ci-dessous se trouve la fonction de récompense $R(s_0, (a^{(1)}, a^{(2)}))$.

		$a^{(1)} =$		
		a'	a''	a'''
$a^{(2)} =$	a'	11	−30	0
	a''	−30	7	6
	a'''	0	0	5

Pour simplifier l'exemple les auteurs ont fixé le discount factor β à 0. Ainsi la table ci-dessus décrivant la fonction de récompense est également la Q-table centrale.

La meilleure action pour les 2 agents est alors a' car elle permet d'obtenir la meilleure récompense. Voyons ce que donne l'algorithme décrit ci-dessus avec cet exemple.

L'algorithme distributed q-learning permet d'obtenir, grace à (11), la table projetée pour l'agent 1 (notée $q^{(1)}$) et pour l'agent 2 (notée $q^{(1)}$) :
Ainsi la projection permet bien de retrouver la meilleure action pour les 2 agents depuis

l'état s_0 soit l'action a'.

Les auteurs avancent également que l'application directe de l'algorithme de Q-learning pour un seul agent de manière parallèle sur les 2 agents entrainent la selection des actions a'' pour les 2 agents. Cela entrainent une récompense de seulement 7 ce qui est sous-optimale.

— **Les méthodes *Coordination-based*** : ces méthodes ont recours à un mécanisme de coordination.

Par exemple la méthode *Coordination graphs* [**?**] décompose la Q-value globale en Q-value locale qui ne prend en compte qu'un sous-ensemble d'agents. Ainsi on peut imaginer décomposer $Q(s, \mathbf{u})$ comme suit :

$$Q(s, \mathbf{u}) = Q_1(x, u_1, u_2) + Q_2(x, u_1, u_3) + Q_3(x, u_3, u_4) \tag{12}$$

La décomposition peut varier selon les états. Typiquement certains états peuvent nécessiter la coordination de certains agents alors que d'autres états non. L'idée de cette décomposition est de simplifier le calcul du maximum de la Q-value globale en résolvant un problème plus simple, de plus petites dimensions.

— ***Indirect Coordination Methods***, l'idée de ce type de méthodes est que chaque agent doit estimer pour chaque action l'action que vont prendre les autres agents. A partir de cette estimation l'agent peut alors choisir en connaissance de cause, la meilleure action pour le groupe.

Parmi ces méthodes, on distingue celles qui s'appliquent pour une tache statique (Joint Action Learners [9] , *Frequency Maximum Q-value* [10]) et celles qui s'appliquent à une tache dynamique (*Optimale Adaptative Learning* [11]).

4.3 Fully-competitive

Pour des taches avec des agents en compétition, les récompenses s'annulent. Par exemple lorsqu'il y a 2 agents on a : $R_1 = -R_2$. Cela signifie que quand l'agent 1 reçoit une récompense r, l'agent 2 reçoit une récompense -r.

Il existe 2 types de méthodes pour un jeu entièrement compétitif :

— les algorithmes indépendants de l'opposant, c'est à dire qu'aucune connaissance de la stratégie de l'adversaire n'est nécessaire.

— A l'inverse les algorithmes **dépendants de l'opposant** nécessite une connaissance de la stratégie de l'opposant. Une modélisation de son comportement pourra par exemple être effectuée.

4.3.1 Algorithme indépendant de l'opposant

Dans les conditions où l'agent n'a aucune connaissance de la stratégie de l'adversaire, un algorithme de minimax peut-être appliqué. Un tel algorithme suppose que les agents en opposition vont toujours chercher à minimiser notre récompense. En partant de cette hypothèse on peut anticiper les coups des adversaires et choisir en conséquence nos actions. On cherche alors à maximiser notre récompense dans le pire des cas. L'algorithme Minimax est donc bien un algorithme indépendant de la stratégie adverse.

Le papier [6] se restreint à l'étude de Jeux avec 2 agents en compétition.

Les auteurs proposent un algorithme qui s'apparente au Q-learning qu'ils appellent minimax-Q. Cet algorithme permet de considérer les actions de l'opposant et ainsi calculer la meilleure action afin de maximiser la récompense dans le pire des cas. Cela suppose donc que l'opposant choisira toujours la meilleure action pour lui.

Les méthodes de Q-learning sont destinées à résoudre une MDP. Afin d'adapter cette méthode au cadre des *Markov Games*, il faut redéfinir V(s) et définir $Q(s, a, o)$ comme étant la qualité d'une action a quand l'opposant effectue l'action o depuis l'état s :

$$Q(s,a) = R(s,a,o) + \gamma \sum_{s' \in S} \mathcal{T}(s,a,o,s')V(s') \tag{13}$$

$$V(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s,a,o)\pi_a \tag{14}$$

où $PD(A)$ est une distribution de probabilité sur l'ensemble des actions possibles.

L'algorithme complet s'écrit alors :

Algorithm 2 méthode minimax [6]

/*initialisation*/

$Q[s,a,o] = 1 \quad \forall s \in S \quad \forall a \in A \quad \forall o \in O$
V[s] = 1 $\quad \forall s \in S$
$pi[s,a] = 1/|A| \quad \forall s \in S \quad \forall a \in A$
alpha = 1
repeat
  /*Choix d'une action*/
  if *random.uniform()* $< \epsilon$ **then**
    choix d'une action aléatoirement
  else
    choix d'une action suivant la probabilité pi[s,a]
  end
  /*MAJ de l'environnment*/
  reward, s' = environnement.update()

  /*Apprentissage*/
  $Q[s,a,o] = (1-\alpha) * Q[s,a,o] + \alpha * (reward + \gamma * V[s'])$
  pi[s,.] = argmax(pi[s',.] , min(o', sum(a',pi[s,a'] * Q[s,a',o'])))
  V[s] = min(o', sum(a', pi[s,a'] * Q(s,a',o')))
  alpha = alpha * decay
until *V(s) and Q(s,a, o) converged*;

Un algorithme indépendant de la stratégie adverse, de type minimax, ne donne pas forcément la meilleure solution. En effet dans le cas où l'opposant adopte une stratégie particulière (pas nécessairement la meilleure stratégie) la solution donnée par un algorithme qui modélise la politique adverse peut-être meilleure.

Les auteurs ont appliqués leur algorithme sur un exemple numérique. Ils ont créé une simulation une partie de football sur une grille de taille 4*5. Les agents peuvent prendre les actions suivantes : droite, gauche, haut, bas ou rester sur place. Si le joueur avec la balle rentre dans la

case où se situe l'autre joueur il perd la possession de la balle. L'objectif est alors de contourner le joueur adverse afin de rentre dans le but avec le ballon.

Les auteurs ont comparés 4 politiques apprises de différentes manières. 2 grâce à l'algorithme minimax-Q et 2 par l'algorithme de Q-learning. Pour chaque algorithme ils ont mis en opposition durant la phase d'apprentissage soit un agent qui prend des actions aléatoirement soit un agent qui utilise une politique apprise de la même manière. C'est politique sont notées respectivement MR (pour minimax/random), MM (pour minimax/minimax), QR (Q-learning/random) et QQ (pour Q-learning/Q-learning).

Enfin durant la phase de test, ils ont utilisé des "Challengers", ce sont des opposants entrainés avec l'algorithme de Q-learning alors que le joueur en test garde une politique fixe. Ces "Challengers" sont alors considérés comme le pire opposant possible. Voici les résultats qu'ils ont obtenus :

	MR		MM		QR		QQ	
	% won	games	% won	games	% won	games	% won	games
vs. random	99.3	6500	99.3	7200	99.4	11300	99.5	8600
vs. hand-built	48.1	4300	53.7	5300	26.1	14300	76.3	3300
vs. MR-challenger	35.0	4300						
vs. MM-challenger			37.5	4400				
vs. QR-challenger					0.0	5500		
vs. QQ-challenger							0.0	1200

Table 3: Results for policies trained by minimax-Q (MR and MM) and Q-learning (QR and QQ).

FIGURE 4 – Résultat des expériences menés par [6]

On remarque que dans le cadre d'un opposant effectuant une stratégie aléatoire les 4 politiques ont à un peu près le même taux de réussite. Par contre on remarque que la politique QQ est bien meilleure que les autres politiques dans le cas d'un opposant avec une stratégie définie manuellement. Les auteurs ont été surpris de ce résultat puisque aucune preuve théorique ne permet de justifier cela. Cependant, d'autres applications ont montré le même genre de résultats. Par contre il est intéressant de voir que seules les 2 politiques apprises par l'algorithme minimax-Q sont en mesure de rivaliser avec leur challenger. Cela prouve que dans l'algorithme de minimax-Q permet effectivement de trouver la meilleure politique de le pire des cas.

4.3.2 Algorithmes dépendants de l'opposant

Il a été montré dans la section précédente que les algorithmes indépendants de l'opposant, tel que minimax-Q ne permettent pas toujours de trouver la solution optimale si l'opposant adopte une stratégie bien particulière. Il existe alors des algorithmes qui permettent de s'adapter à la stratégie adverse.

Il existe notamment les algorithmes M^* [13] et WoLF-PHC [12] qui permettent de modéliser le comportement de l'adversaire. Par exemple il pourrait suffire de compter le nombre de fois où l'opposant prend l'action a depuis l'état s. Ainsi à partir de cette expérience acquise, on peut savoir si l'action a est probable ou non, et ainsi décider de considérer (ou non) cette action afin de choisir notre propre action.

4.4 Taches mixtes

4.4.1 Algorithmes pour un seul agent

Les algorithmes de *reinforcement learning* pour un système avec un seul agent peuvent être appliqués dans le cadre d'un système multi-agent. Néanmoins les preuves théoriques de convergence sont invalidées dans ce cadre, bien qu'en pratique il se peut que l'algorithme convergence vers une solution acceptable. Dans la littérature ce types d'algorithmes sont souvent utilisés du fait de leur simplicité.

4.4.2 Algorithmes agents indépendants

Les algorithmes indépendants des autres agents pour des taches mixtes sont une généralisation de l'algorithme de minimax décrit à la section 4.3. En effet, l'algorithme de minimax peut être remplacé par un solveur de la théorie des jeux. L'algorithme se reformule alors de la manière suivante :

$$h_{i,k}(x,\cdot) = \mathbf{solve}_i\{Q_{\cdot,k}(x_k,\cdot)\} \tag{15}$$

$$Q_{i,k+1}(x_k,u_k) = Q_{i,k}(x_k,u_k) + \alpha\left[r_{i,k+1} + \gamma \cdot eval_i\left\{Q_{\cdot,k}(x_{k+1},\cdot) - Q_{i,k}(x_k,u_k)\right\}\right] \tag{16}$$

Le calcul de la Q-value d'un agent, pour un état et une action, nécessite la connaissance des Q-tables de tous les autres agents. Ainsi chaque agent doit nécessairement dupliquer les Q-tables de tous les agents.

4.4.3 Algorithmes agents dépendants

L'enjeu de ces méthodes est de converger vers la solution optimale mais également de savoir s'adapter à d'autres agents. Certains algorithmes agents dépendants ont des preuves théoriques de convergence principalement pour des environnements statiques (par exemple AWESOME [21]). Les autres algorithmes utilisent des heuristiques mais leur convergence n'est pas garantie (par exemple WoLF-PHC [12]).

5 Les mécanismes de coordinations explicites

Dans les sections précédentes, certains aspects de la coordination entre les enjeux ont été discutés. Cependant, c'est types de coordination ne relevaient pas d'un consensus entre les agents, ils étaient non-explicites. Nous allons passer en revue ici quelques exemples de mécanismes de coordination explicite.

Il est possible de mettre en place des conventions sociales, des rôles ou de la communication entre agents afin qu'ils se coordonnent.

Les conventions sociales permettent d'introduire des préférences parmi l'ensemble des actions possibles [22]. Par exemple, il est possible d'avoir un agent leader qui va imposer une direction et des agents suiveurs qui vont le suivre. De manière générale, il faut effectuer un ordre entre les agents et entre les actions. Cet ordre est connu de tous les agents. Ainsi, chaque agent peut avoir une approximation de la probabilité d'effectuer telle action pour tous les agents.

La communication entre agents peut également entre un moyen qu'ils se coordonnent. Elle peut être utilisée afin de négocier le choix de certaines actions. Puis ce choix peut-être communiquer aux autres agents afin qu'ils aient l'information des intentions de cet agent sur le long terme. [23]

6 Applications

Les domaines d'applications de l'apprentissage par renforcement pour des systèmes multi-agents sont décrit dans cette section.

6.1 Système de robots

— Système de navigation : chaque robot doit rejoindre une cible tout en évitant des obstacles.

— **Observation de plusieurs cibles :** les robots doivent suivre plusieurs cibles afin de les garder dans la portée de leurs capteurs.

— **Poursuite** : un robot ou une équipe de robot doivent capturer des cibles en mouvement.

— **Transport d'objets** : un ensemble d'objets doivent être déplacés d'un lieu à un autre.

— **Compétition de Football** : ce type de compétition est assez populaire dans le domaine de la robotique. Ces compétitions nécessitent d'apprendre un ensemble de taches variées. Ces taches sont individuelles, réception de la balle, déplacement, tir,... Mais également collective afin d'avoir une stratégie d'équipe : défense et attaque groupées. [20]

6.2 Trading Automatique

Ce domaine d'application consiste à trouver la stratégie optimale afin de maximiser le profit lors d'échange de biens. Cela nécessite que les agents apprennent les mécanismes d'enchère et de négociation. [17] [18], [19]

6.3 Gestion de ressources

Dans la gestion de ressources, les agents doivent apprendre la meilleure stratégie afin de maximiser un objectif. Cet objectif doit prendre en compte : le profit, la satisfaction des clients, un stock maximal limité, temp d'attente, ... [14], [15], [16].

7 Conclusion

Ce travail bibliographique a permis de conceptualiser et de connaître les notions de l'apprentissage par renforcement. Notamment comment à partir du formalisme des MDPs il est possible d'aboutir à l'algorithme de Q-learning.

Nous avons vu également que cet algorithme de Q-learning (ou autres dérivées) peut être appliqué tel quel dans le cas d'un système multi-agents si l'on considère que les autres agents ont un comportement statique et font partis de l'environnement. Néanmoins aucune preuve théorique ne prouve pas la convergence de cet algorithme de le cadre d'un système à plusieurs agents. De plus certains exemples on montré la nécessité d'un nouveau formalisme et de mécanisme de coordination entre les agents.

Le formalisme des Markov Games a été défini afin de répondre aux besoins de l'apprentissage de polique d'action pour un système multi-agents. Ce formalisme a permis alors de dériver l'algorithme de Q-learning en une multitude d'algorithmes. Chacun de ces algorithmes permettant de s'appliquer à un type de *Markov Games* : soit entièrement compétitif, soit entièrement coopératif, soit mixte.

Différents processus de coordination explicite ont été brièvement présenté. Ces processus font intervenir soit une hiérarchie entre les agents en définissant un rôle à chaque agent, soit en établissant un procédé de communication entre les agents.

Bien que la multitude d'algorithmes dérivée de l'algorithme de Q-learning permettent de résoudre une majeure partie des problèmes que l'on peut rencontrer. Il m'apparaît que définir les éléments constituants l'environnement, la relation entre les agents, le processus de communication sont des taches qui peuvent s'avérer être fastidieuses et basées sur l'expérience. Peut-être qu'avec le récent engouement pour les méthodes d'intelligence artificielle nous parviendrons à définir un cadre plus général pour l'apprentissage dans des systèmes variés.

Références

[1] https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement

[2] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning", Machine Learning Proceedings, pages 157-163, 1994

[3] Bertsekas, 1987

[4] https ://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa

[5] A Comprehensive Survey of Multiagent Reinforcement Learning, Lucian Bus¸oniu, Robert Babu˘ska, and Bart De Schutter, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C : APPLICATIONS AND REVIEWS, VOL. 38, NO. 2, MARCH 2008

[6] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in Proc. 11th Int. Conf. Mach. Learn. (ICML-94), New Brunswick, NJ, Jul. 10–13, pp. 157–163

[7] An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems, Martin Lauer and Martin Riedmiller, In Proceedings of the Seventeenth International Conference on Machine Learning, 2000, 535-542

[8] "Value-function reinforcement learning in Markov games", Michael L.Littman, Cognitive Systems Research, Volume 2, Issue 1, April 2001, Pages 55-66

[9] The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems, Caroline Claus and Craig Boutilier, AAAI-98

[10] M. Bowling, "Multiagent learning in the presence of agents with limitations," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, May 2003

[11] X. Wang and T. Sandholm, "Reinforcement learning to play an optimal Nash equilibrium in team Markov games," in Proc. Adv. Neural Inf. Process. Syst. (NIPS-02), Vancouver, BC, Canada, Dec. 9–14, vol. 15, pp. 1571–1578.

[12] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," Artif. Intell., vol. 136, no. 2, pp. 215–250, 2002.

[13] D. Carmel and S. Markovitch, "Opponent modeling in multi-agent systems," in Adaptation and Learning in Multi-Agent Systems, G. Weiss and S. Sen, Eds. New York : Springer-Verlag, 1996, ch. 3, pp. 40–52.

[14] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks : A reinforcement learning approach," in Proc. Adv. Neural Inf. Process. Syst. (NIPS-93), Denver, CO, Nov. 29–Dec. 2, vol. 6, pp. 671–678.

[15] S. P. M. Choi and D.-Y. Yeung, "Predictive Q-routing : A memory-based reinforcement learning approach to adaptive traffic control," in Proc. Adv. Neural Inf. Process. Syst. (NIPS-95), Denver, CO, Nov. 27–30, vol.8, pp. 945–951.

[16] P. Tillotson, Q. Wu, and P. Hughes, "Multi-agent learning for routing control within an Internet environment," Eng. Appl. Artif. Intell., vol. 17, no. 2, pp. 179–185, 2004.

[17] W.-T. Hsu and V.-W. Soo, "Market performance of adaptive trading agents in synchronous double auctions," in Proc. 4th Pacific Rim Int. Workshop Multi-Agents. Intell. Agents : Specification Model. Appl. (PRIMA-01). Lecture Notes in Computer Science Series, vol. 2132, Taipei, Taiwan, R.O.C., Jul. 28–29, pp. 108–121.

[18] J.W. Lee and J. Oo, "Amulti-agentQ-learning framework for optimizing stock trading systems," in Proc. 13th Int. Conf. Database Expert Syst.Appl. (DEXA-02). Lecture Notes in Computer Science, vol. 2453, Aixen-Provence, France, Sep. 2–6, pp. 153–162.

[19] J. Oo, J.W. Lee, and B.-T. Zhang, "Stock trading system using reinforcement learning with cooperative agents," in Proc. 19th Int. Conf

[20] A. Merke and M. A. Riedmiller, "Karlsruhe brainstormers—A reinforcement learning approach to robotic soccer," in Robot Soccer World Cup V (RoboCup 2001). Lecture Notes in Computer Science, vol. 2377, Washington, DC, Aug. 2–10, pp. 435–440.

[21] V. Conitzer and T. Sandholm, "AWESOME : A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents," in Proc. 20th Int. Conf. Mach. Learn. (ICML-03), Washington, DC, Aug. 21–24, pp. 83–90.

[22] M. T. J. Spaan, N. Vlassis, and F. C. A. Groen, "High level coordination of agents based on multiagent Markov decision processes with roles," in Proc. Workshop Coop. Robot., 2002 IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS-02), Lausanne, Switzerland, Oct. 1, pp. 66–73.

[23] F. Fischer, M. Rovatsos, and G.Weiss, "Hierarchical reinforcement learning in communication-mediated multiagent coordination," in Proc. 3rd Int. Joint Conf. Auton. Agents Multiagent Syst. (AAMAS-04), New York, Aug. 19–23, pp. 1334–1335.