Q-Concept-Learning : Généralisation à l'aide de treillis de Galois dans l'Apprentissage par Renforcement

Marc Ricordeau
Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
161 rue Ada 34392 Montpellier Cedex 5 France
ricordeau@lirmm.fr

Résumé

Une propriété intéressante de l'Apprentissage par Renforcement est qu'il permet d'apprendre sans connaissances préalables sur l'environnement, notamment sans modèle. Cependant, quand des agents utilisent de tels algorithmes permettant une généralisation, ils sont incapables de justifier leurs choix. Après avoir rappelé les bases de l'Apprentissage par Renforcement et des treillis de Galois, nous introduirons Q-Concept-Learning, un algorithme d'apprentissage par renforcement permettant de généraliser l'apprentissage, d'utiliser des langages structurés ainsi qu'une explication des choix de l'agent. Une validation expérimentale sera également présentée.

Mots Clef

Apprentissage par Renforcement, Treillis de Galois, Treillis de Concepts, Généralisation.

Abstract

One of the interesting properties of Reinforcement Learning algorithms is that they allow to learn without prior knowledge about the environment. However, when the agents use algorithms that enable a generalization of the learning, they are unable to explain their choices. After a reminder about the basis of Reinforcement Learning the lattice concept will be introduced, Q-Concept-Learning, a Reinforcement Learning algorithm that enables a generalization of the learning, the use of structured languages as well as an explanation of the agent's choices will be presented. An experiment will be also presented.

Keywords

Reinforcement Learning, Galois Lattice, Concept Lattice, Generalization.

1 Introduction

L'Apprentissage par Renforcement peut-être comparé à un apprentissage par essais-erreurs, dans le sens où il permet à l'agent d'apprendre par interaction avec son environnement, sans avoir de connaissances préalables sur celui-ci, ni même de savoir quelle est la tâche à accomplir. Seules des récompenses ou des pénalités lui sont fournies. Cependant, parmi les limitations des algorithmes d'Apprentissage par Renforcement, on peut citer :

Premièrement, ils sont incapables d'utiliser des langages plus structurés que les langages par attributs-valeur pour décrire l'environnement de l'agent.

Deuxièmement, l'agent est souvent incapable de généraliser son comportement. La convergence des algorithmes d'Apprentissage par Renforcement repose sur la possibilité de tester toutes les situations, potentiellement infiniment souvent. Cependant, dans la plupart des cas, essayer toutes les possibilités est irréaliste. Le concepteur biaise donc l'apprentissage en donnant des concepts pertinents pour la tâche à l'agent (voir Stone, Sutton (2001) [8]).

Enfin, les algorithmes permettent à l'agent d'améliorer son comportement, mais celui-ci est incapable de justifier le choix de ses actions. C'est notamment le cas pour les réseaux de neurones.

Nous présenterons ici un algorithme permettant d'utiliser des langages plus structurés que les langages par attributs-valeurs d'une part et permettant une généralisation de l'apprentissage d'autre part.

Après avoir présenté les bases de l'Apprentissage par Renforcement dans la partie 2, nous introduirons dans la partie 3 un exemple qui nous servira d'illustration pour préciser quelques points de l'article et pour l'expérimentation. Dans la partie 4, nous présenterons les bases des treillis de Galois, structure qui sera utilisée pour structurer la mémoire de l'agent. Nous exposerons les modifications algorithmiques conséquentes ainsi que la possibilité d'utiliser des langages plus structurés que les langages par attributs-valeur. Nous présenterons partie 5, **Q-Concept-Learning**, un algorithme d'Apprentissage par Renforcement utilisant les treillis de Galois comme structure pour $Q(acuteetat, action)$, la fonction de base des algorithme d'Apprentissage par Renforcement. Nous montrerons qu'il permet la généralisation ainsi que l'extraction de concepts intéressants. Section 6, nous présenterons des résultats expérimentaux basés sur l'exemple décrit partie 3. Finalement, nous conclurons partie 7.

2 Bases de l'Apprentissage par Renforcement

Rappelons les bases de l'Apprentissage par Renforcement. Pour une bonne introduction sur le sujet : Mitchel (1997) [7]. Pour une étude plus complète : Sutton, Barto (1998) [11] ou Kaebling, Littman, Moore. (1996) [3].
Considérons un agent situé pouvant percevoir tout ou partie de son environnement. Il peut agir sur celui-ci avec des actions et reçoit une récompense ou une pénalité en retour.

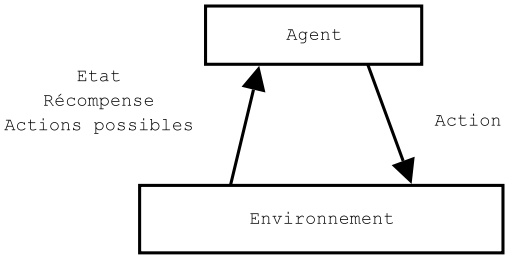

FIG. 1 – Apprentissage par Renforcement : une interaction entre un agent et son environnement

Les algorithmes d'Apprentissage par Renforcement permettent aux agents de produire des comportements (appelés politiques) dans le but de maximiser la récompense cumulée, c'est à dire à long terme. On peut rapprocher cet apprentissage d'un apprentissage par essais-erreurs.
Considérons l'équation (1) qui est l'équation de mise à jour du plus connu des algorithmes d'Apprentissage par Renforcement : le Q-Learning & Dayan (1992) [13].

$$Q(e,a) \leftarrow (1-\alpha)Q(e,a) + \alpha[récompense + \gamma Max_{a'}Q(e',a')] \quad (1)$$

$0 \leq \alpha \leq 1, 0 \leq \gamma \leq 1$

La base de l'apprentissage par renforcement est l'approximation de la fonction $Q(e,a)$ par interactions successives de l'agent avec l'environnement. $Q(e,a)$ associe pour chaque couple (état, action) la valeur de la récompense espérée à terme, c'est à dire la somme des récompenses que l'agent espère en prenant cette action dans cet état, avec un paramètre de dégressivité γ. Plus γ est petit, plus l'agent est myope.
L'apprentissage se fait par une combinaison linéaire entre la valeur de renforcement obtenue (ici $récompense + \gamma Max_{a'}Q(e',a')$) et la récompense espérée ($Q(e,a)$). Le paramètre α sert à pondérer la remise en cause des valeurs précédemment estimées. Plus α est grand, plus la nouvelle valeur de renforcement aura de l'influence. Il est d'usage de diminuer ce paramètre au fur et à mesure de l'apprentissage.

Notation 1 *$Q(e,a)$ représente la vraie valeur de la récompense cumulée à terme en choisissant l'action a dans l'état e.*

Notation 2 *$\hat{Q}(e,a)$ représente la valeur estimée de la récompense cumulée à terme en prenant l'action a dans l'état e.*

L'agent met à jour $\hat{Q}(e,a)$ en combinant ce qu'il connaît déjà avec ce qu'il vient juste d'apprendre (le paramètre α). L'agent peut déterminer une politique gloutonne en utilisant $Q(e,a)$. Il peut prendre dans chaque état, l'action associée à la meilleure récompense cumulée espérée. L'agent doit également choisir ses actions en faisant un compromis entre l'utilisation de la connaissance déjà acquise et l'exploration de nouvelles actions.
La preuve de convergence donnée dans Mitchel (1997) [7] que l'environnement soit stable, c'est à dire qu'il réagit toujours de la même façon aux actions. Cette preuve suppose que chaque couple (état, action) est visité potentiellement infiniment souvent.
Parmi les avantage de tels algorithmes citons :
- Une réponse rapide de l'agent. Celui-ci n'a qu'à comparer les valeurs des actions possibles dans l'état où il se trouve (les $Q(e, action_i)$, si l'agent est dans l'état e), pour choisir une action.
- L'agent peut apprendre et utiliser sa connaissance en même temps.
- L'agent n'a pas à avoir de modèle de son environnement. C'est une propriété forte en apprentissage.

En revanche :
- Dans les algorithmes de base, l'agent doit connaître l'ensemble des états possibles de son environnement au début de son apprentissage.
- C'est un apprentissage par coeur. Tous les états doivent être explorés afin de déterminer une politique. Pour permettre une généralisation de l'apprentissage, on doit utiliser des techniques de réseaux de neurones ou d'approximation de fonction (Sutton, Barto (1998) [11])

3 Exemple

Dans cette partie, nous présenterons un exemple que nous utiliserons pour illustrer des définitions ainsi que dans la partie expérimentation.
L'exemple consiste en un monde-grille, de 3×3, contenant un agent, un case récompense et une case mur. Toutes les autres cases sont vides. Le monde est entouré de murs. Les murs n'ont aucun effet, sinon qu'ils peuvent être déplacés s'il n'y en a aucun autre derrière Si un mur est déplacé sur une case récompense, la récompense est détruite. Si l'agent pousse un mur alors qu'il est impossible à déplacer, rien ne se passe.
La tâche à apprendre est d'aller aussi vite que possible sur la case récompense.
L'agent reçoit une récompense immédiate de 1.0 s'il arrive sur la case récompense. Toutes les autres actions donnent -0.01.
L'agent a une vision complète de son environnement. Il y a 504 $(9 \times 8 \times 7)$ états différents avec 4 actions dans chaque (si l'agent est contre un mur, il peut essayer

de le déplacer. Pousser et se déplacer sont des actions différentes. Chaque épisode dure jusqu'à ce que l'agent atteigne la case récompense ou qu'il la détruise avec un mur. Cet exemple est proche de celui décrit dans Miyamoto & Uehara (1998) [14]. Nous utilisons cet exemple car il contient de nombreuses situations différentes, cependant, beaucoup obéissent à des règles similaires (par exemple, lorsque l'agent est à droite d'une récompense, il doit toujours aller à gauche). Cela permettra de montrer les propriétés de généralisation de notre algorithme.

4 Représentation de $Q(e, a)$ par treillis de Galois

Un façon classique de représenter la fonction $Q(e, a)$ en apprentissage par renforcement est d'utiliser une table qui associe à chaque couple (état, action) une valeur. Les états sont souvent décrits dans un langage par attributs-valeur. Nous allons décrire une structure par treillis de Galois pour $Q(e, a)$ qui permettra d'utiliser des langages plus structurés que les langages par attributs-valeur.

Premièrement, nous rappellerons la base de la construction des treillis de Galois, puis nous introduirons une représentation de $Q(e, a)$ par treillis de Galois, ainsi que les conséquences algorithmiques d'une telle structure. Ensuite, nous présenterons un algorithme incrémental pour la construction de Q. Enfin, nous montrerons que la structure de treillis de Galois permet l'utilisation de langages plus structurés que les langages par attributs-valeur pour décrire l'environnement. Pour cela nous utiliserons les L-Langages (Learning-Languages) décrits dans Liquière (2002) [4].

4.1 Contextes Formels et Treillis de Galois

Dans cette partie, nous rappellerons les bases des treillis de Galois (ou treillis de concepts), du point de vue de l'Apprentissage par Renforcement. Les définitions utilisées ici sont issues de Ganter, & Wille (1999) [1].

Premièrement, considérons une relation binaire entre deux ensembles, le premier étant les objets à décrire, et le second les attributs pour les décrire. Ces ensembles liés par la relation sont appelés contexte formel. Pour notre problème, les objets sont les états de l'environnement vus par l'agent. Ils sont décrits avec un langage par attributs-valeurs dont les valeurs sont booléennes, et qui représentent la présence ou non d'un attribut dans l'état. Par exemple, si l'agent perçoit un mur dans l'état e de son environnement, l'attribut mur aura la valeur **Vrai** pour e, ou e sera en relation avec mur. On pourra suivre sur la figure 2 les différentes étapes de la construction du treillis de Galois.

Définition 1 *Un* $\mathbf{contexte\ formel}$ $\mathbb{K} := (G, M, I)$ *consiste en deux ensembles G et M et une relation I entre G et M. Les éléments de G sont appelés les* $\mathbf{objets}$ *et les éléments de M sont appelés les* $\mathbf{attributs}$ *du contexte. Pour dire qu'**un objet g est en relation I avec un attribut m**, nous écrirons* $\mathbf{gIm}$ *et lirons **g a l'attribut m**.*

Pour un ensemble $A \subseteq G$ d'objets,
$A' = \{m \in M \mid gIm$ pour tout $g \in A\}$
respectivement, pour un ensemble $B \subseteq G$ d'attributs,
$B' = \{g \in G \mid gIm$ pour tout $m \in B\}$

A' représente donc l'ensemble des attributs vus sur l'objet A. Respectivement, B' est l'ensemble des objets qui ont l'attribut B.

Deuxièmement, les objets (ici les états de l'environnement) ainsi que les attributs (ici, les propriétés de l'environnement) sont regroupés pour donner des concepts.

Définition 2 *Un* $\mathbf{concept\ formel}$ *du contexte (G, M, I) est un couple (A, B) avec $A \subseteq G$, $B \subseteq M$, $A' = B$ et $B' = A$. Nous appelons B l'*$\mathbf{extension}$ *et A l'*$\mathbf{intention}$ *du concept (A, B).*

Ensuite, le contexte est clarifié, c'est à dire que les attributs tous présents ou absents simultanément sont fusionnés dans un seul concept. Dans notre contexte, il ne peut y avoir deux états dans Q avec les mêmes attributs. Ainsi, la première partie de la définition du contexte clarifié est toujours vraie.

Définition 3 *Un contexte (G, M, I) est* $\mathbf{clarifié}$*, si pour tout objet $g, h \in G$ de g'=h' on a toujours g = h et, réciproquement, m' = n' implique m=n pour tout $m, n \in M$.*

Finalement, des concepts sont ordonnés dans un treillis, en utilisant la relation d'inclusion des attributs comme relation d'ordre partielle.

Définition 4 *Si (A_1, B_1) et (A_2, B_2) sont les concepts d'un contexte, (A_1, B_1) est appelé* $\mathbf{sous - concept}$ *de (A_2, B_2), indiquant que $A_1 \subseteq A_2$. Dans ce cas, (A_2, B_2) est un* $\mathbf{super - concept}$ *de (A_1, B_1) et nous écrivons $(A_1, B_1) \leq (A_2, B_2)$. La relation $\leq$ est appelée* $\mathbf{ordre\ hiérarchique}$ *pour les concepts. L'ensemble de tous les concepts de (G, M, I) ordonnés ainsi est appelé* $\mathbf{treillis\ de\ Galois}$ *(ou treillis de concepts) du contexte (G, M, I).*

4.2 Représentation de Q(e,a) par un treillis de Galois

La fonction $Q(e, a)$ dans les méthodes d'Apprentissage par Renforcement sert à retrouver une valeur, associée à l'action a dans l'état e, correspondant à $\hat{Q}(e, a)$, la récompense à terme évaluée. L'algorithme permettant de trouver $\hat{Q}(e, a)$ dans des représentations classiques de $Q(e, a)$ est une recherche dans une table. Avec une représentation de $Q(e, a)$ par un treillis de Galois (cf. figure 3), la recherche des valeurs des actions associées à un état devient une recherche descendante dans un treillis (cf. algorithme 1).

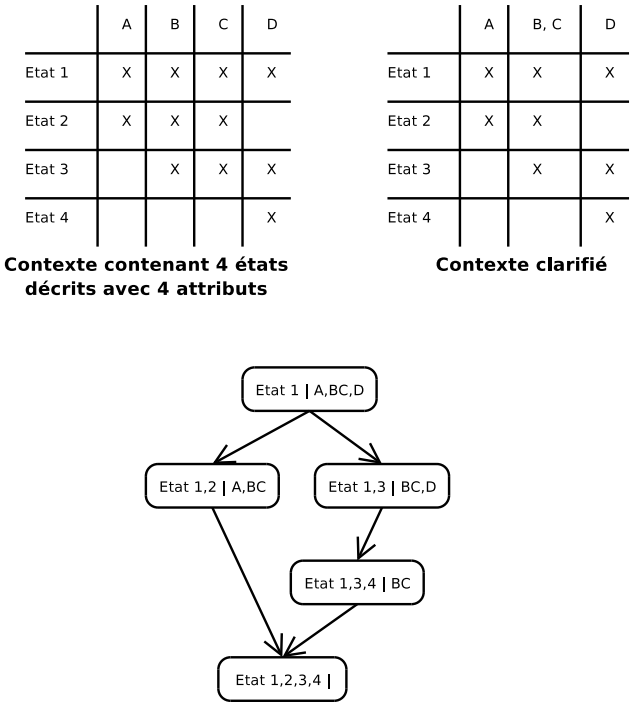

FIG. 2 – Exemple de construction de treillis de Galois

4.3 Construction Incrémentale de $Q(e, a)$

Comme nous l'avons mentionné dans la partie 2, sur l'Apprentissage par Renforcement, l'agent doit connaître l'ensemble des actions possibles dans l'ensemble des état possibles au début de son apprentissage. Nous pouvons supprimer cette contrainte par une construction incrémentale du Treillis de Galois représentant $Q(e, a)$. Un tel algorithme est décrit dans Godin & Missaoui, Alaoui (1995) [2]. Il consiste en une comparaison des attributs du nouvel objet à insérer (le nouvel état) avec une recherche par spécialisation dans le treillis, en construisant des nouveaux noeuds quand nécessaire. Cet algorithme a été implémenté pour notre expérience.

4.4 L-Langage (Learning-Language)

Nous avons introduit une représentation de $Q(e, a)$ par treillis de Galois, mais avec un langage par attributs-valeurs booléen. En effet, les états de l'environnement ont été décrits par la présence où l'absence d'attributs. Il est prouvé dans Liquière (2002) [4] que tout langage muni d'une relation d'ordre partiel et d'un opérateur de généralisation peut être utilisé pour construire un treillis de Galois.

Définition 5 *Un L-Langage (Learning Language) est un quadruplet* $[L, \geq, \otimes, \cong]$ *avec*
– *L est un langage avec une relation d'ordre partiel* $\geq$.
– $\otimes$ *est un opérateur de généralisation avec : si* $D = D_1 \otimes D_2$ *alors* $D \geq D_1$, $D \geq D_2$ *et* $\forall D', D' \geq D_1$ *et* $D' \geq D_2 \Rightarrow D' \geq D$
– $\cong$ *est une relation d'équivalence entre les éléments de L.*

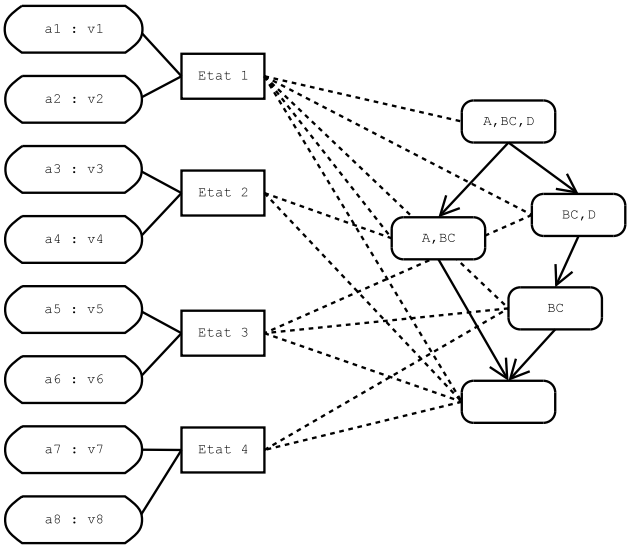

FIG. 3 – Exemple de Q(e,a) représentée par un Treillis de Galois

Nous pouvons donc utiliser cette propriété pour décrire les états de l'environnement à l'aide de langages plus structurés que les langages par attributs-valeur.
Pour l'expérience, la description fournie par l'environnement à l'agent est un graphe étiqueté décomposé en chemins. Ce formalisme et cette méthode ont été choisis pour montrer que l'utilisation d'une représentation par treillis pour $Q(e, a)$ permet d'utiliser des formalismes plus structurés que les descriptions classiques par attributs-valeurs. Le propos n'est donc pas ici de développer la méthode de décomposition des graphes, mais de montrer qu'elle a été effectivement implémentée. Par ailleurs, pour les lecteurs intéressés par cette méthode, elle est décrite dans Liquière, & Sallantin (1989)[6].
– Le L-Langage L est le résultat de la décomposition du graphe en chemins.
– La relation d'ordre partiel $\geq$ est l'inclusion pour l'ensemble des chemins.
– L'opérateur de généralisation $\otimes$ est l'intersection des ensembles de chemins.
– La relation d'équivalence $\cong$ est l'égalité pour les ensembles de chemins.

La figure 4 donne un exemple d'états décomposés en chemins de longueur 3 avec un agent qui peut voir les 4 cases autour de lui. L'expérience utilise la même technique, mais avec une vue plus lointaine pour l'agent et des chemins plus long pour la décomposition.

5 Q-Concept-Learning

Dans cette partie, nous présenterons **Q-Concept-Learning**, un algorithme dérivé du Q-Learning (Watkins

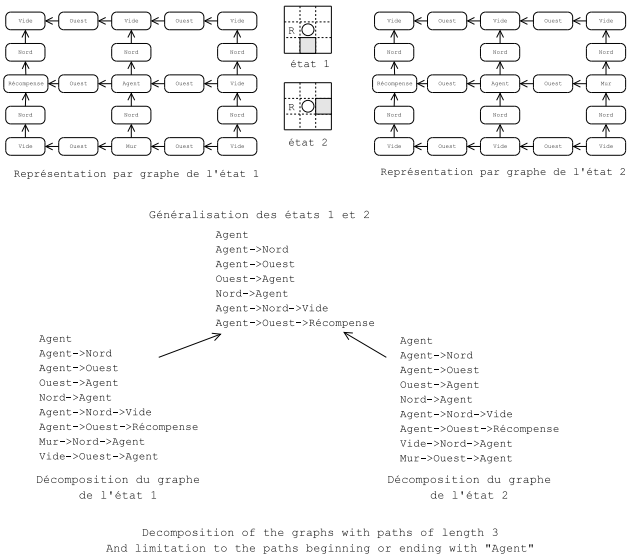

FIG. 4 – Exemples d'états décrits à l'aide de graphes étiquetés et les décompositions correspondantes

& Dayan (1992) [13]) utilisant une structure de treillis de Galois pour $Q(e, a)$. Nous verrons que cet algorithme value les concepts en plus des états de l'environnement. Cette méthode nous permettra de généraliser l'apprentissage. De plus, nous montrerons que l'agent sera capable de trouver et d'utiliser des concepts fiables et significatifs dans le contexte d'une tâche.

5.1 Valuer les Concepts

Le principe des algorithmes par renforcement comme le **Q-Learning**, est d'approximer par interactions successives avec l'environnement, la valeur des récompenses espérées à terme, en prenant l'action a dans l'état e. Ces valeurs sont retrouvées à l'aide de la fonction Q(état, action). L'idée principale de **Q-Concept-Learning**, est d'utiliser une représentation de la fonction Q(état,action) par treillis de Galois et de valuer non seulement les couples (états, actions), mais également les couples (concepts, actions). Ainsi, en plus d'avoir la formule de mise à jour classique :

$$Q(e, a) \leftarrow (1 - \alpha)Q(e, a) + \alpha[r + \gamma Max_{a'}Q(e', a')]$$

nous aurons les mises à jours suivantes :

pour $c \in \{\text{concepts } c' \mid c'Ie\}$ **faire**
  $Q(c, a) \leftarrow (1 - \alpha)Q(c, a) - \alpha[r + \gamma Max_{a'}Q(e', a')]$;
fin

$0 \leq \alpha \leq 1, 0 \leq \gamma \leq 1$

Avec une représentation classique de $Q(e, a)$, pour choisir l'action à effectuer dans l'état e, l'agent n'a le choix que parmi les actions a_i possibles dans l'état e. Avec la représentation de $Q(e, a)$ par treillis de Galois, si l'agent est dans l'état e, il devra choisir un concept parmi tous les concepts c qui sont en relation avec e avant de sélectionner une action. Nous allons donc introduire quatre notions (la précision, la pertinence, la fiabilité et l'encouragement) qui permettront de différencier les concepts, et qui serviront à l'agent lors du choix de l'action.

5.2 Précision d'un concept

Dans un environnement stable, les valeurs de renforcement fournies par l'environnement amènent les valeurs estimées $\hat{Q}(état, action)$ à converger vers leur vrai valeur $Q(état, action)$. Se pose alors la question de la qualité de la convergence. Intuitivement, on pourrait dire que plus la différence entre la valeur estimée $\hat{Q}(état, action)$ et la valeur de renforcement est faible, meilleur est l'apprentissage. Introduisons donc la notion de précision pour les concepts ainsi que pour les états.

Définition 6 Précision
$Précision(\hat{Q}(état, action)) =$ *moyenne (corrections de* $Q(état, action)$*) avec* $Q(état, action)$ *comme valeurs de renforcement (si les vraies valeurs* $Q(état, action)$ *sont utilisées).*
$\hat{P}récision(\hat{Q}(état, action)) =$ *moyenne (corrections de* $\hat{Q}(état, action)$*) avec* $\hat{Q}(état, action)$ *comme valeurs de renforcement (si les valeurs estimées* $\hat{Q}(état, action)$ *sont utilisées).*
La définition est similaire pour les concepts.

Le problème est que la précision ne donne pas une bonne approximation de la convergence. En effet, étant donné que les valeurs estimées plutôt que les valeurs réelles sont utilisées comme valeur de renforcement, celles-ci peuvent être fausses, notamment à cause d'un manque d'exploration. Cependant, nous verrons dans le prochain paragraphe que cette mesure peut être utile.

5.3 Pertinence d'un Concept

Parmi tous les $couples(concept, action)$ que l'agent a à sa disposition, certains sont "utiles" relativement à la tâche à accomplir, d'autres ne le sont pas (voir figure 5 pour un exemple). Du point de vue de l'Apprentissage par Renforcement, ces $couples(concept, action)$ "utiles", sont ceux qui sont informatifs sur l'environnement par rapport à la tâche à accomplir. Dans le cadre d'un environnement stable et si les vraies valeurs de renforcement étaient disponibles (plutôt que les valeurs estimées), de tels $couples(concept, action)$ recevraient toujours la même valeur de renforcement. Autrement dit, chaque fois qu'un tel $concept$ est vu sur un $état$, l'$action$ dans cet $état$ donne toujours la même $valeur$ de renforcement. Ainsi, nous pouvons définir plus formellement la notion de concept pertinent.

Définition 7 *Un* $Q(concept, action)$ *est* **pertinent***, relativement à une tâche dans un environnement stable si pour tout état | état I concept,* $Q(état, action) = V \in \mathbb{R}$.

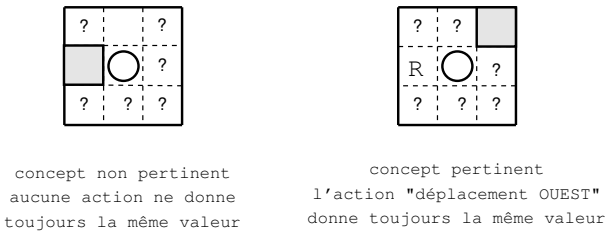

FIG. 5 – Exemple de pertinence de concepts

Nous voulons conserver la propriété forte de l'Apprentissage par Renforcement qui est qu'un agent n'a pas à avoir de modèle de son environnement. En particulier, pour trouver les $couples(concept, action)$ pertinents parmi tous les $\hat{Q}(concept, action)$, l'agent n'aura pas d'information sur la distribution des états, ainsi bien sûr que sur la distribution des concepts. Ceci implique que la plupart des méthodes décrites dans Mitchel (1997) [7] pour évaluer l'apprentissage ne peuvent être utilisées. Par conséquent, l'agent aura à sa disposition pour trouver les $\hat{Q}(concept, action)$ pertinents :
– Le nombre de mise à jour de $\hat{Q}(concept, action)$
– Les valeurs de renforcement
– L'ordre partiel $\leq$ des concepts
Comme l'agent n'a pas d'information sur la distribution des exemples, les valeurs des $\hat{Q}(concept, action)$ ne peuvent être comparées qu'entre les concepts en relation par $\leq$. L'agent peut alors utiliser la précision pour trouver les $\hat{Q}(concept, action)$ non pertinents.

Théorème 1 *Considérons deux concepts c_1 et c_2 avec $c_1 > c_2$ et une action a. Si $Précision(\hat{Q}(c_1, a)) < Précision(\hat{Q}(c_2, a))$, alors c_1 n'est pas pertinent.*

Preuve 1 *Si $c_1 > c_2$, chaque fois que $\hat{Q}(c_2, a)$ est mis à jour, $\hat{Q}(c_1, a)$ est aussi mis à jour. Si $\hat{Q}(c_1, a)$ est pertinent pour une tâche dans un environnement stable, alors $Q(c_1, a)$ existe. Le nombre de mises à jour de $\hat{Q}(c_1, a) >$ nombre de mises à jour de $\hat{Q}(c_2, a)$, on devrait avoir $Précision(\hat{Q}(c_1, a)) \geq Précision(\hat{Q}(c_2, a))$. Donc $Q(c_1, a)$ n'est pas pertinent.*

La propriété est vraie pour tous les concepts $c > c_1$ et concernant l'action a.
Nous utiliserons cette propriété pour marquer les concepts non pertinents pour une tâche. Comme les valeurs de renforcement seront les valeurs estimées $\hat{Q}(c, a)$ plutôt que $Q(c, a)$, des concepts seront marqués non pertinents alors qu'ils le sont, à cause du manque d'exploration. Comme nous le verrons dans l'algorithme 2, la propriété de pertinence sera vérifiée pour chaque $couple(concept, état)$ après chaque mise à jour de $\hat{Q}(concept, état)$.

5.4 Meilleurs Concepts

Parmi les $\hat{Q}(concept, action)$ pertinents, certains sont particulièrement intéressants : les plus généraux. Voir figure 6 pour un exemple.

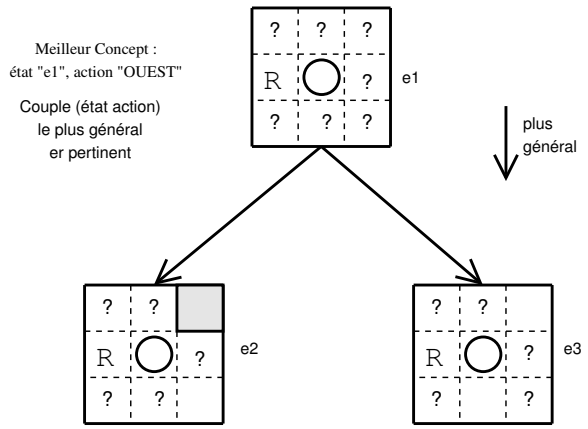

FIG. 6 – Exemple de meilleur concept

Définition 8 $\hat{Q}(c_1, a) \geq_Q \hat{Q}(c_2, a) \Leftrightarrow c_1 > c_2, \hat{P}récision(c_1, a) > \hat{P}récision(c_2, a)$ *et* $\hat{Q}_{Max}(c_1, a) = \hat{Q}_{Max}(c_2, a) = a$

Définition 9 $\hat{Q}(c_2, a) = \textbf{Meilleur Concept}(\hat{Q}(c_1, a))$ *est le plus général et pertinent concept* $\in \{c \mid \hat{Q}(c, a) \geq_Q \hat{Q}(c_1, a)\}$

Meilleur Concept ($\hat{Q}(c, a)$) est trouvé par une recherche montante dans le treillis, en commençant par c et en utilisant la définition.
Ces concepts sont doublement intéressants :
– Il sont plus fiables. Etant donné que leur valeur est plus souvent mis à jour, les valeurs $\hat{Q}(c, a)$ doivent être plus proches de $Q(c, a)$. C'est donc une meilleure valeur de renforcement à utiliser pour les mises à jour.
– Les concepts sont plus généraux, donc sont en relation avec plus d'exemples, tout en restant pertinents. Si nous considérons **Q-Concept-Learning** comme un extracteur de concepts, ceux-ci sont les plus significatifs relativement à la tâche considérée.

5.5 Fiabilité d'un Concept

Les $\hat{Q}(concept, action)$ devraient être éviter d'être choisis s'ils n'ont pas été assez explorés. Ainsi, l'agent considérera des concepts plus généraux. Introduisons la définition de la fiabilité, mesurant l'exploration d'un $\hat{Q}(concept, action)$.

Définition 10 $\hat{Q}(concept, action)$ *est* **fiable** *si* $\hat{P}récision(\hat{Q}(concept, action)) < seuil\ de\ fiabilité \times \hat{Q}_{max}(c, a)$, $0 < seuil\ de\ fiabilité \leq 1$.

Ce qui veut dire que $\hat{Q}(concept, action)$ est fiable si la moyenne des corrections est plus petite que *seuil de fiabilité*, relativement à la récompense à terme espérée maximum pour les actions de ce concept. Ceci parce que une précision de 0,1 est meilleure si la valeur est de 1 que si la valeur est de 0,5. Comme l'agent ne connaît pas les valeurs réelles, nous utilisons les valeurs estimées.

5.6 Actions encouragées

Comme nous l'avons vu, La moyenne des corrections devrait converger vers 0 pour les $couples(concept, état)$ $pertinents$. Cependant, cela peut prendre longtemps. L'agent devrait être encouragé à choisir des actions qui ont donné un meilleur résultat que prévu, même si celles-ci ont été peu explorées. De telles actions sont caractérisées par des $\hat{Q}_{max}(concept, action)$ dont la correction est positive. Au contraire, si une action encouragée a donné un résultat plus mauvais que prévu, elle ne sera plus encouragée. Nous ne pouvons pas utiliser les méthodes telles que les valeurs initiales optimistes (Sutton, Barto (1998) [11]), car les $\hat{Q}(c, a)$ ne peuvent être comparées pour des concepts différents. Les actions seront donc explicitement encouragées par une valeur booléenne.

5.7 Algorithme

Finalement, l'algorithme 2 présente **Q-Concept-Learning**. On y retrouvera les notions développées précédemment : la valuation des concepts, la mise à jour de la précision pour les concepts, les tests de pertinence, de fiabilité et d'encouragement.

La formule de mise à jour classique de $\hat{Q}(e, a)$ entraîne une mise à jour par itération. Dans notre algorithme, d'autres calculs viennent s'ajouter :

- Mise à jour de $\hat{Q}(état, action)$
- Mise à jour de tous les $\hat{Q}(concept, action)$ avec concept $\in$ {concept I état}
- Mise à jour de $\hat{P}récision(concept, action)$ avec concept $\in$ {concept I état}
- Mise à jour de $\hat{Q}(concept, action)$ pour la pertinence et l'encouragement

La complexité exacte de ces opérations dépend de la taille du treillis, ainsi que du nombre d'états et de leur similarité. La relation exacte entre la similarité des exemples et la taille du treillis n'a pas été calculée. Néanmoins, plus les états seront similaires, moins il faudra de concepts pour les décrire, moins le treillis sera grand.

5.8 Choix d'une action

Dans les algorithmes classiques d'Apprentissage par Renforcement, les agents choisissent leurs actions en utilisant une politique $\epsilon-gloutonne$. Une très simple (Sutton, Barto (1998) [11]) serait ($0 \leq x \leq 1$) :

- L'agent a x chances de choisir la meilleure action de l'état considéré
- L'agent a $1 - x$ chances de choisir une action aléatoire

Le choix doit être plus élaboré dans notre problème, car l'agent a le choix parmi différents concepts pour sélectionner l'action la plus appropriée. Le problème pour l'agent sera de choisir le concept le plus adéquat.

L'idée de la fonction $choisir\ une\ action\ a$ dans un état e consiste à choisir le bon (c, a), $c \in \{cIétat\}$ avec :

- $\hat{Q}(c, a)$ pertinent
- $\hat{Q}(c, a)$ fiable
- c le plus spécifique possible

c doit être pertinent pour les raisons exposées dans le paragraphe précédent. L'agent choisit les concepts les plus spécifiques car ils correspondent plus précisément aux états. Cependant, si $\hat{Q}(c, a)$ n'est pas assez fiable, l'agent choisira un concept plus général mais plus fiable. Ainsi, si l'agent se retrouve avec dans un état peu exploré, il considérera celui-ci du point de vue de concepts plus généraux pour sélectionner son action.

Finalement, l'algorithme 3 décrit comment l'agent choisit l'action a dans l'état e avec **Q-Concept-Learning**.

5.9 Induction

Comme nous l'avons montré ci-dessus, **Q-Concept-Learning** permet la généralisation de l'apprentissage, donc il permet à l'agent d'agir, même dans un environnement partiellement inconnu. Mais ce n'est pas le seul avantage. Tous les algorithmes d'Apprentissage par Renforcement améliore le comportement d'un agent. Nous pouvons l'évaluer grâce aux récompenses obtenues. Cependant, dans la plupart de ces méthodes, l'agent ne peut pas expliquer pourquoi il a obtenu un meilleur comportement. Ceci est particulièrement vrai pour les réseaux de neurones. Dans le TD-Gammon décrit dans Tesauro (1994) [9], l'algorithme se classe parmi les meilleurs joueurs du monde au Back-Gammon, mais nous devons seulement admettre qu'il joue bien. Il ne peut pas expliquer ses tactiques. En fait, la forme même du réseaux de neurones donne des indications à l'agent sur la façon de jouer au Back-Gammon. Dans notre algorithme, l'agent peut améliorer son comportement avec l'expérience, mais il peut également retourner les concepts qu'il juge pertinents pour une tâche donnée. Ce n'est plus que de l'apprentissage par coeur, l'agent apprend une partie du modèle de son environnement. Contrairement à Dyna-Q (Sutton, Barto (1998) [11]), il ne peut néanmoins pas savoir que telle action mène dans tel état.

6 Expérimentation

L'exemple donné dans la partie 3 est utilisé pour l'expérimentation. C'est un exemple simple, mais le nombre de $Q(état, action)$ est important : $\mid Q(e, a) \mid = 9 \times 8 \times 7 \times 4 = 2016$, c'est le nombre de situations différentes que l'agent a à rencontrer avec les actions possibles dans chacun des cas. Cependant, des situations sont similaires (si l'agent est à l'ouest d'une récompense, il devrait prendre l'action *est*, le reste de la description est sans importance).

L'agent reçoit la description de l'environnement sous forme de graphe étiqueté comme décrit dans la partie 4. Le L-Langage utilisé est l'ensemble des chemins de $longueur \leq 8$ (l'agent perçoit l'ensemble de son environnement).

6.1 Resultats

La figure 7 montre la moyenne des récompenses obtenues par l'agent avec une **marche aléatoire**, un **Q-Learning** classique et **Q-Concept-Learning**.

Données : d une description dans un L-Langage
Résultat : e, l'état de l'environnement correspondant à d dans Q avec une structure de treillis de Galois
$\emptyset$ si $s \notin Q$
début
si $Q = \emptyset$ **alors** retourner $\emptyset$;
$queue$.ajouter(concept le plus général de Q);
$candidats \leftarrow \{s \in Q\}$;
stop $\leftarrow$ **Faux**;
tant que *queue* $\neq \emptyset$ **et** $\neg$*stop* **faire**
$concept\ courant \leftarrow queue$.premier();
si $d \geq$ *concept courant.intention()* **alors** $candidats \leftarrow candidats \cap concept\ courant$.extension();
$voisins\ inférieurs \leftarrow$ {concept $\in Q$ | concept $\prec$ *concept courant* **et** concept $\neg$visité};
set {concept $\in voisins\ inférieurs$} visités;
$queue$.ajouter($voisins\ inférieurs$);
sinon
$candidats \leftarrow candidats - concept\ courant$.intention();
set {concept $\in \{concept < concept\ courant\}$ } visités;
fin
si $candidats = \emptyset$ **alors** *stop* $\leftarrow$ **Vrai**;
fin
si $candidats = \emptyset$ **alors** retourner $\emptyset$;
sinon
retourner $candidats$ (qui doit être un singleton);
fin
fin

Algorithme 1: Trouver un état de l'environnement e d'une description d dans $Q(e, a)$ représenté avec un treillis de Galois

Initialiser $Q(e, a)$;
répéter
Initialiser e;
Choisir une action a en utilisant une politique dérivée de Q;
Observer l'environnement (e);
si $e \notin Q$ **alors** mettre à jour Q;
Observer la récompense (r);
$meilleure\ action \leftarrow$ Meilleure Action($Q(e', a')$);
$Q(e, a) \leftarrow (1 - \alpha)Q(e, a) - \alpha[r + \gamma$Meilleur Concept($Q(e', meilleure\ action)$)];
pour $c \in$ {concepts $c' \mid c'Ie$} **faire**
si $correction > 0$ **alors** marquer $Q(c, a)$ comme encouragé;
sinon
marquer $Q(c, a)$ comme $\neg$ encouragé;
fin
$Précision(c, a) \leftarrow Précision(c, a) + \frac{correction - Précision(c,a)}{t+1}$;
$Q(c, a) \leftarrow (1 - \alpha)Q(c, a) - \alpha[r + \gamma Max_{a'}Q(e', a')]$;
marquer {concept | $c <$ concept **et** $Précision(c, a) > Précision(concept, a)$} comme $\neg$ Pertinent **et** $\neg$ encouragé;
fin
$s \leftarrow s'$;
jusqu'à e soit terminal;

Algorithme 2: Q-Concept-Learning : Apprentissage par Renforcement avec valuation de concepts

Données : $e \in$ {Etats de l'environnement connus dans Q}

Résultat : a, l'action devant être exécutée dans l'état e parmi les actions de $Q(e, a)$, les actions possibles pour l'agent dans l'état e

si e est Fiable(seuil de fiabilité) **alors** Choisir l'action $a \mid Q(e, a) = Max_{a'}Q(e, a')$;
sinon
  c← le concept le plus spécifique $\mid cIe$;
  Recherche ascendante dans $\{concept > c\}$;
  $l \leftarrow$ la couche la plus spécifique avec au moins une action encouragée;
  si l existe **alors** Choisir a, comme la meilleure action encouragée de l;
  sinon
    Recherche ascendante dans $\{concept > c\}$;
    $l \leftarrow$ la couche la plus spécifique avec au moins un $\hat{Q}(c, a)$ Fiable (seuil de fiabilité) et pertinent;
    si l existe **alors** Choisir a comme la meilleure $\hat{Q}(c, a)$ Fiable(seuil de fiabilité) et pertinent de l;
    sinon
      Choisir a aléatoirement;
    fin
  fin
fin

1 En pratique, les deux recherches ascendantes se font en même temps.

Algorithme 3: Sélectionner une action, a dans l'état e avec Q-Concept-Learning

- ϵ (cf partie 5.8), le facteur d'exploration pour la stratégie ϵ-gloutonne : $\epsilon = \frac{1}{t}$ si $\frac{1}{t} > 0.1$, $\epsilon = 0.1$ sinon.
- Les valeurs initiales de $\hat{Q}(état, action)$ sont égales à 0.
- Les valeurs initiales de $\hat{Q}(concept, action)$ sont les valeurs des concepts dont ils sont issus, 0 s'ils sont complètement nouveaux.
- $Seuil\ de\ Fiabilité = 0.1$
- $\alpha = 0.5$
- $\gamma = 0.6$

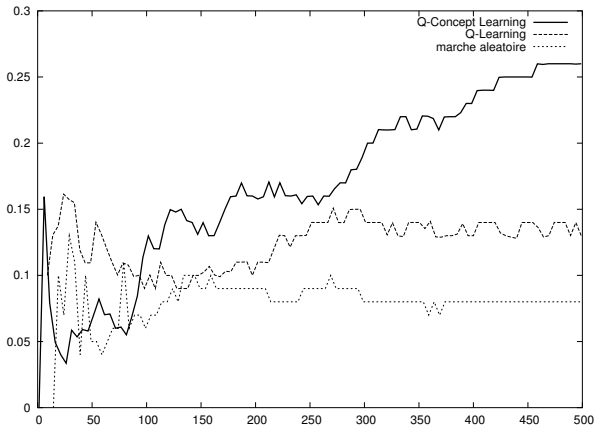

FIG. 7 – Comparaison des récompenses moyennes pour une marche aléatoire, Q-Learning and Q-Concept-Learning

Après 500 itérations, la moyenne des récompenses est deux fois meilleure pour **Q-Concept-Learning**, comparée à **Q-Learning**. Ce résultat n'est pas surprenant, puisque les situations de départ sont presque toutes différentes. Ainsi, avec le **Q-Learning**, l'agent n'a pas pu accumuler assez d'expériences avec seulement 500 itérations.

Pour cette expérimentation :

- La **Marche Aléatoire** a obtenu 44 récompenses.
- Le **Q-Learning** a obtenu 71 récompenses, 242 états différents et 968 différentes actions ont été vus.
- Le **Q-Concept-Learning** a obtenu 135 récompenses, 279 états différents décrits avec 2791 concepts, 11866 actions dont 971 sont fiables et pertinentes ont été vus.

Nous pouvons voir que la politique obtenue grâce à **Q-Concept-Learning** n'est pas optimale. On le voit lorsque la courbe décroît même vers la fin. Il y a deux explications :

- ϵ (Le facteur d'exploration) n'est jamais égal à 0. (Ce n'est pas le problème le plus significatif)
- De nouvelles situations donnent de nouveaux concepts qui sont encouragés s'ils donnent une meilleure valeur de renforcement que prévue. C'est souvent le cas avec les concepts totalement nouveaux (s'ils contiennent des attributs jamais vus sur les autres états) avec une valeur initiales de 0.

De temps en temps, **Q-Concept-Learning** donne de moins bons résultats que ceux montrés. Cela arrive si la politique d'exploration du départ n'a pas permis d'explorer les quatre façons d'avoir la récompense (aller au nord en étant au sur d'une récompense, ...). L'agent préférera des actions plus fiables aux actions non explorées. L'exploration de la politique ϵ-gloutonne permettra néanmoins à l'agent de changer son comportement. Ce problème peut-être éviter en orientant l'agent au début de son apprentissage. Si on présente 4 exemples à l'agent au début de son apprentissage avec comme seules actions possibles celles qui permettent d'obtenir une récompense, la courbe est toujours au moins aussi bonne que celle qui est présentée.

6.2 Détails Techniques

- Le programme est écrit en Java 1.4, en utilisant la plateforme Eclipse (www.eclipse.org)
- La visualisation des mouvements ainsi que ses résultats au cours de l'apprentissage sont faits en utilisant la plateforme multi-agents MadKit (www.madkit.org)
- La visualisation des treillis est faites grâce à une extension du paquetage Jgraph (jgraph.sourceforge.net)
- Les expérimentations ont été réalisées à l'aide d'un PC sous Linux Mandrake 9.1 (kernel 2.4.21), processeur AMD Athlon XP 1700, RAM : 384 Mo

6.3 Critiques

La première critique évidente concerne le fait qu'un treillis soit stocké en mémoire. La construction du treillis, même incrémentale, devient rapidement une opération coûteuse. Néanmoins, après une période d'apprentissage, le treillis entier n'est plus utile. Les concepts non pertinents peuvent être supprimés. De plus, tous les concepts du treillis ne sont pas nécessaires pour décrire les exemples. Les $\bigwedge$-irréductibles du treillis sont suffisants pour séparer l'ensemble des états (preuve dans Liquière & Sallantin (1998) [5]). Le problème vient du fait que l'ensemble des $\bigwedge$-irréductibles peut évoluer si de nouveaux exemples arrivent.
Si on dispose de critères de convergence ou de limitation de mémoire, la construction du treillis peut être stoppée. Ainsi, les concepts non pertinents et ceux qui ne sont pas $\bigwedge$-irréductibles peuvent être supprimés.

7 Conclusion et Travaux Futurs

Premièrement, nous avons exposé les bases de l'Apprentissage par Renforcement et des treillis de Galois. Nous avons expliqué comment les treillis de Galois pouvaient être utilisées pour l'Apprentissage par Renforcement, ainsi que les conséquences algorithmiques.
La première avancée de cette méthode est la possibilité d'utiliser un langage plus structuré que les langage par attributs-valeur, pour décrire l'environnement de l'agent, en utilisant les L-Langages (Liquière (2002) [4]). Ensuite, nous avons présenté **Q-Concept-Learning**, un algorithme d'Apprentissage par Renforcement permettant de généraliser l'apprentissage c'est à dire d'utiliser des connaissances apprises sur des situations nouvelles. Cet algorithme conserve la représentation symbolique, contrairement aux approximations de fonctions ou aux réseaux de neurones. De plus, **Q-Concept-Learning** permet la construction d'une partie d'un modèle de l'environnement par l'agent. Il peut trouver quels sont les concepts utiles pour la tâche qu'il a appris. Nous avons implémenté **Q-Concept-Learning** sur un exemple simple permettant de montrer les améliorations dues à la généralisation, ainsi que de retourner des concepts utiles.
Dans de futur travaux, nous montrerions en quoi **Q-Concept-Learning** peut être vu comme une recherche par renforcement de l'Espace des Versions, permettant d'extraire d'un apprentissage les concepts cohérents avec une tâche. De plus, nous verrions comment limiter la taille du treillis. Enfin, nous combinerions cet algorithme avec d'autres méthodes comme Dyna-Q (Sutton, Barto(1998) [11]), afin que l'agent puisse construire un modèle plus complet de son environnement à l'aide d'un Apprentissage par Renforcement.

Références

[1] Ganter & Wille (1999) Formal Concept Analysis : Mathematical Foundations. Springer Verlag, Berlin – Heidelberg

[2] Godin & Missaoui, Alaoui (1995) Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence, 11(2)

[3] Kaebling, Littman, Moore (1996) Reinforcement Learning : A survey. J. Art Intel. Res. 4 :237-285 ; http ://www.cs.washington.edu/research/jair/volume4/kaelbling96a-html/rl-survey.html

[4] Liquière (2002) Recherche du Noyau Ontologique. CAP 2002

[5] Liquière & Sallantin (1998) Structural machine learning with Galois lattice and Graphs" ,Machine Learning : Proceedings of the 1998 International Conference (ICML 98) Morgan Kaufmann Ed,pp 305-313

[6] Liquière & Sallantin (1989) INNE : a structural leraning algorithm for noisy esxamples, Procceedings of the Fourth European Working Session on Learning (Montpellier - 1989)

[7] Mitchel (1997) Machine Learning, McGraw-Hill International Published

[8] Stone, Sutton (2001). Scaling reinforcement learning toward RoboCup soccer. Proceedings of the 18th International Conference one Plots Learning

[9] Tesauro (1994) Neural Computation, volume 6, (no 2), pages 215-19 in 1994

[10] Baxter & Tridgell & Weaver (2000) Learning to Play Chess Using Temporal-Differences. In Machine Learning, ISSN 0885-6125, Vol. 40 No. 3, September 2000, pages 243-263.

[11] Sutton, Barto (1998). Reinforcement Learning : an Introduction. MIT Press.

[12] Watkins (1989) Learning from Delayed Rewards. PhD thesis, Cambridge Univ., Cambridge, England

[13] Watkins & Dayan (1992). Q-Learning. Learning machine, 8 (3/4), 279-292

[14] Miyamoto & Uehara (1998) Discovering New Features to improve Q-Learning More Efficiently in the Special Interest Group Notes IPSJ of, Flight. 98, No. 105, pages 57-62