Apprendre à anticiper dans un système multi-agent - Premiers résultats

Erwan Livolant, Michaël Piel, Serge Stinckwich
Équipe Modèles, Agents, Décision (MAD) - GREYC - Université de Caen

Résumé : L'anticipation est une faculté essentielle aux êtres vivants et artificiels. Elle permet à ceux-ci de prédire les conséquences de leurs actions sur leur environnement et sur eux-mêmes. Nous nous intéressons dans cette communication à une méthode d'apprentissage par renforcement, les systèmes de classeurs, qui présente l'originalité d'intégrer des capacités d'anticipation. Le méta-apprentissage enrichit les systèmes de classeurs en les rendant efficaces dans un environnement multi-agent. Cette méthode est basée sur le concept d'accélération du renforcement par rapport au temps. Nous présentons ici les premiers résultats d'expérimentation obtenus à l'aide d'une plate-forme, LCSTalk, que nous avons développé.

Mots clés : futur, systèmes anticipatoires, apprentissage par renforcement, système de classeurs, méta-apprentissage, systèmes multi-agents

1 Introduction

Comment prédire le futur à partir du passé ? Le domaine des systèmes anticipatoires a l'ambition de répondre scientifiquement à cette question. L'anticipation a un rôle important et est encore sous-estimée dans la compréhension des systèmes complexes et adaptatifs. Elle consiste pour un animat à exploiter la connaissance qu'il a du futur afin d'améliorer son comportement courant. Depuis le début du 20ième siècle, plusieurs travaux, de psychologues notamment, ont tenté de montrer que la prédiction d'évènements futurs et de leurs conséquences, appellée également «anticipation», joue un rôle fondamental dans la coordination et la réalisation des comportements adaptatifs des êtres vivants, ainsi que dans les mécanismes de mémoire et d'attention. Comme il a été montré dans les travaux du biologiste théoricien Robert Rosen [Ros85] (et de ceux qui ont poursuivi ses travaux comme Paul Davidsson [Dav96]), la réalisation d'un système artificiel doué de capacités d'anticipation n'est pas chose facile, car on s'attaque alors à la vue dominante du monde, inspirée par un modèle déterministe fondé sur la physique dans lequel il y a une distinction claire entre cause et effet[1].

Nous nous intéressons ici à la réalisation d'une classe particulière de systèmes anticipatoires utilisant un mécanisme d'apprentissage par renforcement (RL pour Reinforcement Learning) et nous nous attachons à montrer comment un système multi-agent peut apprendre à anticiper les

[1]Pour disposer d'un cadre plus général sur les problèmatiques de l'anticipation, on pourra se reporter au papier publié à Rochebrune 2003 ([Sti03]) par l'un des auteurs.

modifications de son environnement. La tâche d'un agent artificiel est alors d'apprendre une politique optimale, c'est-à-dire comment agir de façon à maximiser la récompense qu'il obtient à long terme.

Dans notre étude, nous avons utilisé les systèmes de classeurs (LCS pour Learning Classifier Systems) comme mécanisme de RL. Leur principal intérêt, par rapport à d'autres approches d'apprentissage par renforcement comme les processus décisionnels de Markov, est de limiter l'explosion du nombre d'états à explorer et d'offrir la possibilité de généralisation de leurs règles de comportement. Comme ils sont utilisables en ligne, les systèmes de classeurs s'adaptent plus facilement aux modifications de l'environnement. La politique de comportement de l'agent dans son environnement est determinée par une population de classeurs (ou règles) dont l'évolution est assurée à l'aide des paradigmes classiques des algorithmes génétiques. Pour une situation donnée, l'agent exécute l'action associée à un classeur choisi selon sa qualité.

Les travaux initiés par le psychologue Edward Tolmann en 1932 ([Tol32]) puis Wolfgang Stolzmann ([Sto98]) sur l'apprentissage latent ont permis récemment d'incorporer dans le modèle des classeurs la notion d'anticipation. Ce mécanisme d'apprentissage s'attache à décrire la dynamique d'interaction de l'agent avec son environnement indépendamment de la notion de récompense (ou punition) à la différence de l'apprentissage par renforcement. Dans l'architecture ACS (Anticipatory Classifier System), les classeurs anticipent la situation future de l'agent si l'action associée est réalisée.

Le problème est que ces approches ne sont plus du tout adaptées dans un contexte d'agents multiples interagissant dans un même environnement, i.e. un environnement dit non-markovien dans lequel l'ensemble des perceptions disponibles à un instant donné ne suffisent pas à prendre la meilleure décision. En effet, le mécanisme d'apprentissage des LCS implique que l'agent «pense» être le seul à modifier son environnement [BGS99].

Nous proposons dans cette communication d'adapter l'apprentissage par classeurs aux systèmes multi-agents à l'aide d'une technique de méta-apprentissage (ML pour MetaLearning) issue de travaux de Jürgen Schmidhuber. Le méta-apprentissage (i.e. apprendre à apprendre) consiste à permettre au système d'apprendre à modifier par lui-même la politique de comportement apprise et ceci indépendamment de l'environnement. Le système tient alors compte du fait que ce qu'un agent apprend à un instant donné affecte les conditions d'apprentissage des autres agents et également de lui-même pour le futur. Pour cela, des actions non-environnementales sont intégrées aux systèmes. Ces actions auto-référencielles modifient, indépendamment de l'environnement, la politique de comportement (les classeurs) qui les met en œuvre (en même temps que les actions environnementales). Un algorithme dit de «Success Story» permet par la suite de valider les modifications entreprises si celles-ci ont assuré une accélération du renforcement par rapport au temps. Avec un tel mécanisme, les agents peuvent apprendre à anticiper la dynamique de l'environnement engendrée par les autres agents.

2 Systèmes de classeurs

Nous rappellons dans cette première partie les théories classiques d'apprentissage tirées de l'étude des comportements animaux, qui vont fonder les modèles informatiques que nous allons présenter par la suite. Puis nous détaillons plus précisemment les définitions et les principes qui gouvernent les modèles d'apprentissage ici, les LCS, qui prennent appui sur ces théories. Nous nous intéresserons plus particulièrement aux propriétés de généralisation et d'anticipation des LCS.

2.1 Apprentissage par renforcement vs apprentissage latent

Des études sur le comportement animal ont mis en évidence deux types d'apprentissage : l'apprentissage par renforcement et apprentissage latent.

Le RL est une classe de problèmes dans laquelle un agent autonome situé améliore au fur et à mesure son comportement en maximisant une fonction de récompense à partir d'une suite de réponses scalaires fournies par l'environnement (récompense ou punition). Cette approche est issue des travaux sur les conditionnements «béhavioristes» de Pavlov (1927) et de Skinner (1938) [Ger02] où l'apprentissage n'est motivé que par une récompense. Les mécanismes les plus connus pour faire de l'apprentissage par renforcement sont le Q-Learning et les LCS ([MH94]).

En 1949, Seward[Sew49] met en évidence un autre phénomène d'apprentissage qu'il qualifia de latent[2] en examinant le comportement de rats explorant des labyrinthes qu'il expliqua par l'existence d'un modèle interne de l'environnement pour les rats. Cet apprentissage latent fut formalisé en 1992, par Hoffman [Hof93] en psychologie cognitive par *la théorie du contrôle anticipatif du comportement.*

Différentes implémentations artificielles ont été tirées de ces études psychologiques pour résoudre informatiquement des problèmes de décision nécessitant plusieurs actions successives. En 1989, Watkins propose l'algorithme incrémental *Q-learning* [Wat89] qui permet à un agent, de la même manière que lors d'un conditionnement opérant à la Skinner[3], d'apprendre selon une récompense. En se basant sur le fait que les conséquences d'une action ne se résument pas à une seule récompense mais également à une nouvelle situation, Sutton propose l'algorithme *DynaQ+* [Sut91] où le modèle environnemental construit par apprentissage latent permet l'accélération de l'apprentissage par rapport au *Q-learning.*

Les systèmes de classeurs comme nous allons le voir maintenant offrent l'avantage de pouvoir combiner ces deux types d'apprentissage dans un même contexte.

2.2 Principe des systèmes de classeurs d'apprentissage

Les LCS sont des systèmes à base de règles qui permettent de résoudre des problèmes d'apprentissage par renforcement et d'apprentissage latent [Ger02]. Tout en reprenant les principes des deux algorithmes précédents, les LCS ont des propriétés de généralisation des règles de comportement et offrent des capacités d'anticipation.

Un exemple d'environnement utilisé comme test pour l'apprentissage avec les LCS est présenté à la figure n° 1. Dans celui-ci, l'agent doit apprendre à rejoindre la position de récompense en un minimum de déplacements. Par déplacement, on entend ici un mouvement d'une case à une autre ou un déplacement vers un mur (auquel cas l'agent ne bouge pas).

L'agent, situé dans un environnement, a une boucle sensori-motrice qui lui permet de percevoir et d'agir sur son milieu à chaque instant. La situation dans laquelle se trouve l'agent est décrite par un ensemble d'attributs de perception locale. Dans notre exemple de la figure n° 1, cette perception locale est [00011000]. On décrit par convention l'environnement à partir de la case située au nord de l'agent en tournant dans le sens des aiguilles d'une montre.

[2] L'*apprentissage latent* est l'apprentissage sans récompense ou punition à la différence de l'apprentissage par renforcement.

[3] Le conditionnement opérant permet à un animal d'apprendre des relations entre une action et sa conséquence. Ce mécanisme a été mis en évidence par la fameuse boîte de Skinner, contenant un pigeon et un levier délivrant de la nourriture. Plus le pigeon déclenche le levier, plus la relation de cause-à-effet entre le levier et l'apparition de la nourriture se renforce.

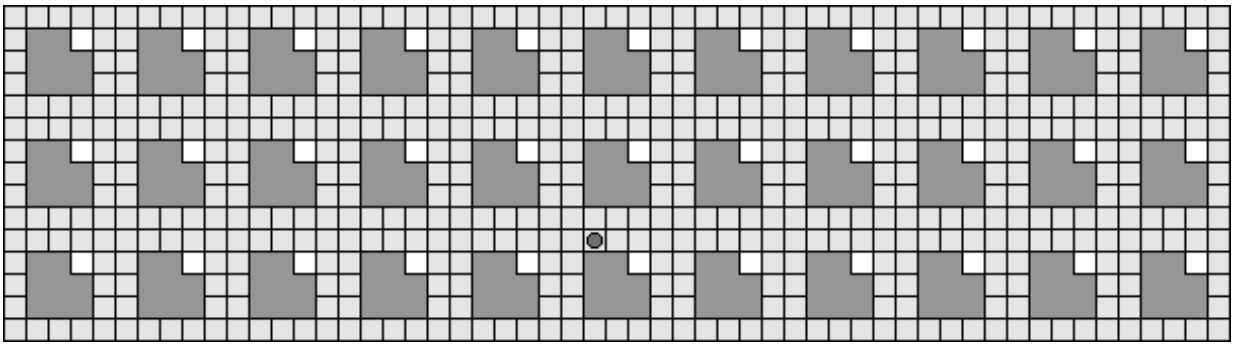

FIG. 1 – Environnement 2D « Woods1 ». L'agent (disque situé vers le centre) doit apprendre à rejoindre le point de nourriture (cases blanches) le plus proche, où il obtient une récompense, en évitant les obstacles (parties foncées).

Les règles de comportement des LCS sont appelées «classeurs». Chaque classeur est composé au moins d'une partie [Condition], d'une partie [Action] et d'une valeur sélective (voir figure n° 2). L'agent recherche dans son ensemble de règles si une ou plusieurs [Condition]s s'apparient avec sa perception courante. S'il en trouve, il choisit de réaliser l'[Action] qui correspond au classeur qui possède la plus forte valeur sélective (celle qui lui apportera potentiellement la plus grande récompense). Sinon, une nouvelle règle est créée pour cette situation et on réalise une action au hasard. La valeur sélective représente la capacité du classeur à résoudre un problème.

Condition	Action	Valeur Sélective
[00011000]	[↓]	0.8

FIG. 2 – Représentation minimale d'un classeur.

L'architecture générale d'un LCS, illustrée par la figure n° 3, est composée de quatre éléments :
- l'interface d'entrée qui formate les perceptions de l'environnement en messages pouvant être interprétés par le système,
- l'interface de sortie qui traduit les messages en actions effectives sur l'environnement,
- la liste des messages permettant de stocker tous les messages d'entrée, de sortie et internes,
- la liste des classeurs représentant la politique de l'agent.

À chaque pas de temps, les messages sont appariés aux classeurs. Les messages mis en correspondance disparaissent alors de la liste. Les messages d'actions, choisis parmi les classeurs activés par le mécanisme d'appariement, sont postés sur la liste. Dans notre exemple de la figure n° 3, on peut remarquer un message $M0$ qui n'est pas un message de sortie. C'est en fait un message interne qui sera apparié aux classeurs au pas de temps suivant.

Une fois que l'agent est arrivé à la case de nourriture, il reçoit une récompense de la part de l'environnement. Celle-ci est rétropropagée à toutes les actions qui ont mené à cette récompense par un algorithme comme le *Bucket Brigade* [Ger02].

La base de règles doit évoluer tout au long de l'interaction de l'agent avec son environnement. Pour cela, le système utilise généralement des algorithmes génétiques. Pour favoriser l'exploration de

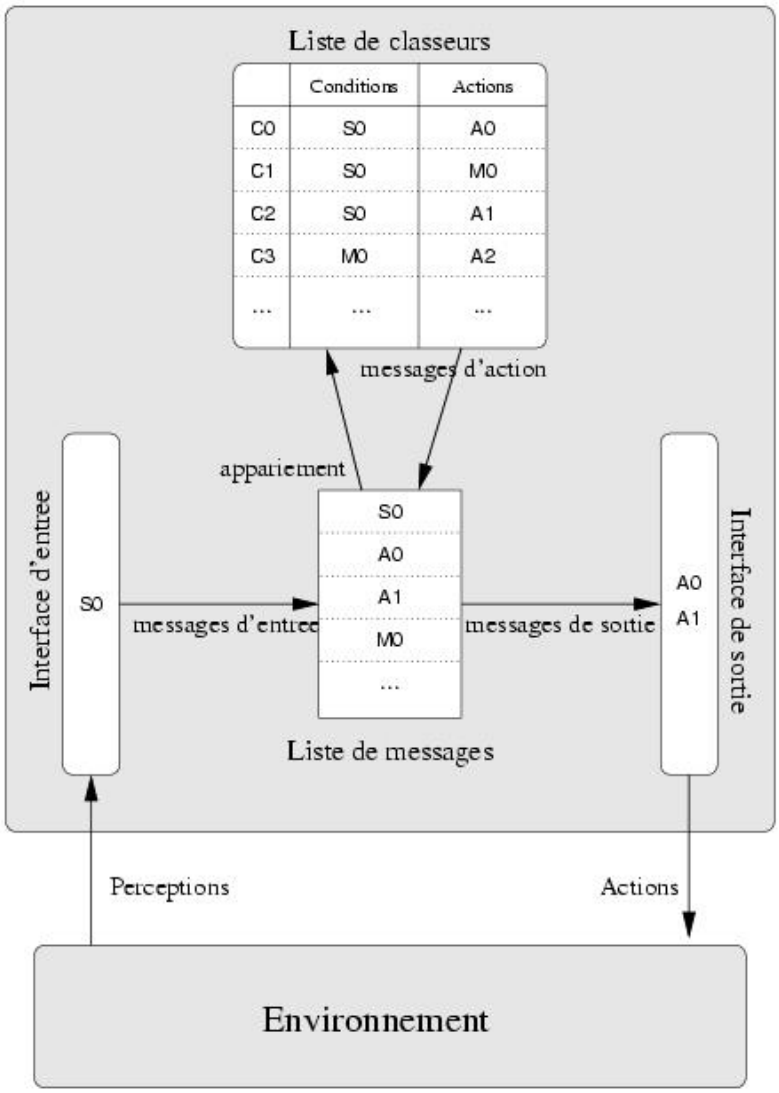

FIG. 3 – Architecture générale des systèmes de classeurs

l'environnement par l'agent, le LCS élimine un certain nombre de classeurs les moins bien adaptés à chaque pas de temps et les remplace par d'autres. Ces nouveaux classeurs sont créés par des opérateurs de croisements et mutations des classeurs qui ont les plus fortes valeurs sélectives. Les «gènes» considérés ici sont les attributs de la [Condition], ou la valeur de l'[Action].

Intéressons nous maintenant aux propriétés de généralisation et d'anticipation dans les LCS.

2.3 Généralisation

Par rapport aux algorithmes d'apprentissage par renforcement de la famille Q-learning, les LCS ont l'avantage de posséder un mécanisme de généralisation qui exploite les régularités dans la dynamique d'interactions entre l'agent et l'environnement.

Dans la partie [Condition] d'une règle, chaque symbole spécifie la valeur de l'attribut correspon-

dant pour que le classeur soit choisi. Le symbole #, appelé «peu importe» (don't care), exprime le fait que l'on peut ignorer cet attribut dans le processus d'appariement d'un classeur avec la situation courante de l'agent. Un classeur dont la partie [Condition] contient des symboles # est plus général que ceux qui n'en ont pas car il correspond à plusieurs situations de l'agent. Ainsi, un classeur contenant [11#0011#] peut être associé à quatre situations différentes de l'agent : [11000110], [11000111], [11100110], [11100111].

Le mécanisme de généralisation réalise un groupement de plusieurs classeurs. Les classeurs en question possèdent la même action et la même valeur sélective. On généralise en ne gardant dans la partie [Condition] que les attributs identiques. Les autres symboles sont remplacés par un #. On obtient ainsi, à la fin de cette opération, un unique classeur qui convient à plusieurs situations.

Différentes architectures de LCS bénéficiant de ce mécanisme de généralisation tels que ZCS[4] et XCS[5] ne se distinguent que par leur modalité de calcul de la valeur sélective des classeurs.

2.4 Anticipation

Si un animat considère les prédictions des différents gains qu'il peut obtenir en effectuant certaines actions afin de décider quel comportement il va adopter, il possède déjà une forme réduite d'anticipation[6]. On peut donc dire que même un classeur simple comme XCS ou ZCS dispose déjà de cette forme d'anticipation rudimentaire, car limitée à une seule valeur, l'anticipation de la récompense.

Des formes plus élaborées d'anticipation (i.e. où l'on anticipe plusieurs valeurs) peuvent être mises en œuvre en intégrant au LCS, la notion d'apprentissage latent.

Le rôle de l'anticipation dans la psychologie animale est pour la première fois conceptualisé par le psychologue Edward Tolmann en 1932 [Tol32] qui pensait que le stimuli-réponse behavioriste classique alors à la mode était trop limité pour expliquer certains comportements animaux. Il introduit donc la notion de SRE-units (Situation-Reaction-Effect) qui représente la connaissance d'un organisme que dans la situation S, la réaction R produira la conséquence E.

Hoffman ([Hof93]) a proposé une théorie du comportement animal, appellée «contrôle anticipatif du comportement» où l'animal forme constamment une prédiction des conséquences de ses actions. Hoffmann a montré comment l'anticipation peut être apprise et contrôler le comportement, ce qu'il formule de la manière suivante :

> «un comportement intentionel (R) est toujours accompagné de l'anticipation des effets (E_{ant}) escomptés par rapport à une expérience précédente dans la situation (S). L'anticipation des effets (E_{ant}) est toujours comparée à l'effet réel (E_{real}). Les anticipations correctes doivent provoquer le renforcement de la relation entre la situation (S) et les effets anticipés (E_{ant}) alors que les anticipations incorrectes doivent provoquer au contraire une différenciation des conditions liées à la situation.» (voir figure n° 4).

Cette théorie a ensuite servi de base pour Stolzmann ([Sto98], [But01]) pour concevoir des classeurs anticipatoires (ACS). Les classeurs d'ACS possèdent en plus de ceux des autres LCS, une

[4] Zeroth level Classifier System, c'est-à-dire classeur de niveau zéro, où l'on n'a gardé que le strict minimum du fonctionnement d'un classeur (pas de mémoire notamment).

[5] L'originalité de XCS est de distinguer la force d'un classeur de sa valeur sélective.

[6] Anticipations payantes ou Payoff Anticipations selon Butz, Sigaud, Gérard ([BSG03]).

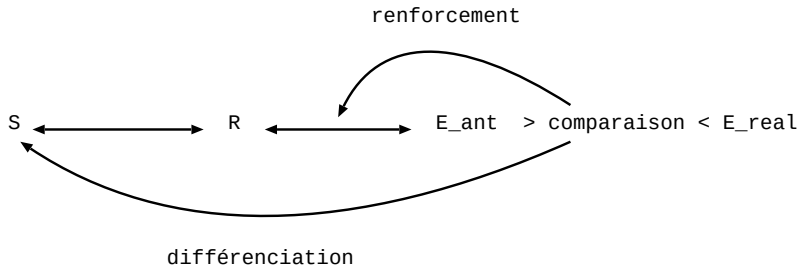

FIG. 4 – Contrôle anticipatif du comportement

partie [Effet] qui anticipe la situation future de l'agent si l'[Action] du classeur est réalisée. L'agent apprend donc la dynamique de ces interactions avec son milieu.

Dans la partie [Effet], la future situation de l'agent est décrite avec les mêmes attributs que pour la [condition]. Le symbole #[7] («don't change») est utilisé dans la partie [Effet] du classeur pour exprimer le fait qu'un attribut reste inchangé une fois l'action réalisée.

Un classeur ACS représente donc bien une unité SRE selon Tolman :

- une partie condition S qui contient des informations sur des attributs des états de l'environnement, qui vont déclencher le classeur,
- une partie action R qui contient des instructions pour les effecteurs,
- une partie attente ou conséquence C^{ant} qui spécifie la prochaine anticipation de l'état de l'environnement,

De plus, à chaque classeur c sont associées deux forces : la force de récompense s_c, évaluation de la récompense attendue et la force d'anticipation s_c^{ant} décrivant la qualité de la correction des conséquences anticipées.

Les règles produites permettent à l'agent d'anticiper dans n'importe quelle situation : le système peut déterminer à l'avance la séquence qui lui permet d'atteindre son but. De part son origine liée à des théories psychologiques, ACS à la différence des systèmes de classeurs précédents n'utilise pas d'algorithmes génétiques, mais des heuristiques.

À priori, l'apprentissage latent permet une convergence plus rapide que l'apprentissage par renforcement, puisque l'agent n'est pas obligé d'attendre la récompense qui peut arriver longtemps après le choix d'une action.

3 Méta-apprentissage dans les systèmes de classeurs

Notre objectif est d'utiliser les systèmes de classeurs dans un contexte multi-agents. Les travaux sur l'utilisation des LCS dans des environnements non-markoviens [8] s'intéressent essentiellement au problème où l'agent ne peut pas distinguer des situations différentes du fait de l'absence d'informations globales. Ceci est résolu généralement en ajoutant une mémoire temporaire au LCS. Les systèmes multi-agents sont des environnements non-markoviens plus complexes car un agent n'a pas accès à l'état interne des autres agents. Parmi les rares travaux sur l'adaptation des classeurs

[7] Le symbole est le même que pour la condition mais la sémantique diffère.

[8] Un environnement est dit markovien si chaque entrée perceptive correspond à un état distinct de l'environnement.

en environnements multi-agents, on peut citer ceux sur OCS (Organizational Classifier System) ([TRS00]).

Nous proposons une nouvelle façon de résoudre les difficultés liées à un environnement non markovien en utilisant une technique de méta-apprentissage due à Jürgen Schmidhuber. Nous présentons maintenant l'adaptation que nous en avons faite aux LCS et allons discuter les choix qui ont été effectués.

3.1 Accélération du renforcement au cours du temps

Le méta-apprentissage selon Jürgen Schmidhuber [SZW96] [ZS96] [Sch96] consiste à permettre au système d'apprentissage de modifier par lui-même la politique de comportement qu'il apprend, indépendamment de l'environnement. Schmidhuber introduit plusieurs notions permettant la réalisation d'un ML que nous définissons dans ce paragraphe.

Cette méthode est applicable quelque soit l'algorithme d'apprentissage par renforcement. Dans [SZW96] par exemple, le ML est conjugué au *Q-learning*. De plus, le ML permet d'adapter l'apprentissage par renforcement aux applications multi-agents.

L'idée est de permettre à l'agent d'améliorer par un apprentissage la façon dont il apprend un comportement en se basant sur la notion d'accélération de la récompense acquise. Cette accélération est mesurée à l'aide du rapport renforcement/temps. Entre deux instants de la vie de l'agent t_1 et t_2 où $t_2 > t_1$, il est possible de calculer l'accélération du renforcement :

$$Q(t) = \frac{R(t_2) - R(t_1)}{t_2 - t_1}$$

Les directives imposées par le ML, pendant ce laps de temps, à la politique apprise par l'algorithme de RL classique sont validées grâce à ce critère si elles assurent l'accélération du renforcement.

Pour que le système puisse appliquer ces directives, il doit pouvoir réaliser, en plus des actions sur son environnement, des actions sur son propre comportement. Ces dernières sont appelées actions auto-modificatrices (AAM).

De plus, le système doit pouvoir vérifier si les modifications qu'il réalise sont intéressantes pour le reste de la vie de l'agent. Pour cela, Schmidhuber propose un algorithme avec «backtrack» nommé «Success-Story Algorithm» (SSA) qui assure l'accélération de l'apprentissage.

3.2 Actions auto-modificatrices

Les AAM sont des actions qui modifient la politique que l'agent construit à l'aide d'un algorithme d'apprentissage par renforcement. La particularité principale de ces actions, dans la proposition de Schmidhuber, est qu'elles sont du même niveau que les actions sur l'environnement. C'est une seule et même politique qui permet d'effectuer les actions sur l'environnement et les AAM. Ainsi les AAM sont auto-référentielles puisqu'elles agissent sur la politique qui les engendre et donc potentiellement sur elles-mêmes.

Généralement les modifications réalisées sur la politique sont des actions d'incrémentation ou de décrémentation de la valeur discriminante qui détermine le choix d'action de l'agent suivant une situation.

Un processus de modification de la politique (PMP) commence lorsque l'une de ces AAM est effectuée. Il s'arrête lorsqu'une AAM particulière est choisie : l'action d'arrêt de mode auto-référentiel

(ESM pour *End Self Mode*). Une deuxième phase du ML se déclenche alors. Elle se terminera lorsqu'une autre AAM modifiante sera exécutée.

Cette période peut être comparée à une phase d'exploitation du ML avant que ce dernier soit validé par l'algorithme SSA. Il est important de souligner que la durée de ces deux phases est variable et choisie également par le système puisque l'occurence des AAM est déterminée par la politique de l'agent.

Les actions permettant d'apprendre à apprendre doivent pouvoir faire évoluer la politique de l'agent indépendamment de l'environnement. Dans le cas des LCS, la prise de décision de l'animat se fait au moyen de la valeur sélective des classeurs. C'est donc sur cet élément des classeurs que les AAM doivent agir. Nous avons introduit trois types d'AAM :

- incrémentation de la valeur sélective,
- décrémentation de cette même valeur,
- arrêt du mode auto-référentiel.

Dans l'architecture de Schmidhuber, les AAM modifient une politique par un pourcentage variable suivant un paramètre. Dans notre proposition, nous avons choisi de ne pas avoir ce pourcentage de modification variable mais constant. La valeur de ce pourcentage est à définir expérimentalement. Le choix d'un pourcentage fixe permet de ne pas alourdir la représentation des classeurs. Cependant, cette restriction peut impliquer une perte de performance car les possibilités d'exploration du ML sont réduites. Toutefois, le fonctionnement global de l'algorithme est conservé.

Condition	Action	Paramètre	Valeur Sélective
[#011#0##]	[r]	[]	0.8
[10#100#1]	[l]	[]	**0.3**
...	...	...	...
[11#0#100]	[i]	[#00100#1]	0.6
[011##10#]	[f]	[]	0.4
↓	↓	↓	↓
[#011#0##]	[r]	[]	0.8
[10#100#1]	[l]	[]	**0.6**
...	...	...	...
[11#0#100]	[i]	[#00100#1]	0.6
[011##10#]	[f]	[]	0.4

FIG. 5 – Exemple d'une population de classeurs avec une action auto-référentielle i incrémentant la valeur sélective du classeur (le second) pouvant être appariée au motif [#00100#1]. Le valeur du pourcentage de modification est ici 100%. Les actions f, r et l sont environnementales.

Pour qu'une AAM s'applique à un classeur, il est nécessaire de disposer d'un moyen de le sélectionner dans la population de classeurs, c'est-à-dire d'avoir une représentation d'un classeur ou d'un ensemble de classeurs dans le LCS (voir la figure n° 5). Ses différentes composantes sont définies ainsi :

- une condition (situation de l'animat),

- une action (sur l'environnement ou sur la politique de l'agent),
- un paramètre (vide pour les actions environnementales) désignant un pattern de classeurs sur lequel se porte la modification.
- les différentes valeurs permettant de calculer la valeur sélective du classeur (suivant ZCS, XCS, ACS) et cette dernière.

Le choix d'utiliser un motif de condition pour associer une AAM à des classeurs s'explique par le problème du renouvellement de la population de classeurs. En effet, les classeurs sont constamment évalués par le système et les plus faibles sont éliminés. Cette élimination engendre un problème potentiel pour les AAM si celles-ci ne concerne qu'un unique classeur. Ces AAM deviendraient obsolètes et devraient être supprimées à leur tour. Grâce au pattern, les AAM du ML peuvent perdurer indépendamment du mécanisme de suppression des classeurs inadaptés à l'environnement.

3.3 Adaptation de l'algorithme «Success-Story»

Cet algorithme permet d'assurer une accélération des performances du système au cours du temps. La notion de temps est déterminée ici par le nombre d'actions réalisées (y compris les actions auto-modificatrices).

L'idée directrice de cet algorithme avec «backtrack» consiste en la sauvegarde de la politique courante avant la modification de celle-ci par une AAM. Si les modifications réalisées lors d'un PMP se révèlent inefficaces à accélérer le renforcement, elles seront invalidées et les anciens classeurs seront restaurés. Pour sauvegarder toutes les modifications m correspondant aux PMP_i, on utilise une pile S contenant les informations suivantes :

- le renforcement : $S(m).R$,
- le temps : $S(m).t$,
- l'ancien classeur avant modification : $S(m).old$,
- la position dans la pile de la première modification du PMP : $S(m).f$.

La validation d'un PMP se fait à chaque fin de la phase d'exploitation du ML. SSA calcule l'évolution du renforcement donné par l'environnement à l'aide du rapport renforcement/temps $Q(t)$.

L'algorithme vérifie si le PMP_i courant a été bénéfique par rapport aux précédents à l'aide du critère «Success-Story Criterion» (SSC) défini par les deux inégalités suivantes :

- $Q(i,t) > \frac{R(t)}{t}$, le rapport renforcement/temps de l'unique PMP existant dans la pile est supérieur au renforcement moyen depuis la «naissance» de l'agent.
- $Q(i,t) > Q(k,t)$, avec $k < i$, le rapport renforcement/temps du PMP courant est supérieur à celui du PMP précédent dans le cas où au moins deux PMP existent dans la pile.

Cet algorithme peut être décrit par l'itération suivante :

Algorithme *SSA*

Tant que (sp != 0) **et**

$$\left(\frac{R(t) - S(S(sp).f).R}{t - S(S(sp).f).t} \leq \frac{R(t) - S(S(S(sp).f - 1).f).R}{t - S(S(S(sp).f - 1).f).t}\right)$$

Alors $sp = sp - 1$;
$P \leftarrow S(sp).old$;

Le premier élément de la pile, qui selon l'algorithme ne peut être dépilé, est une modification fictive. L'utilisation de cette modification fictive est utile pour n'avoir qu'un seul algorithme pour la validation des deux inégalités du SSC. Au début de sa vie, l'agent a les valeurs d'attributs suivantes :

- $S(0).t = 0$,
- $S(0).R = 0$ car la récompense est nulle à la naissance de l'agent,
- $S(0).f = 0$,
- $S(0).old$ non initialisé.

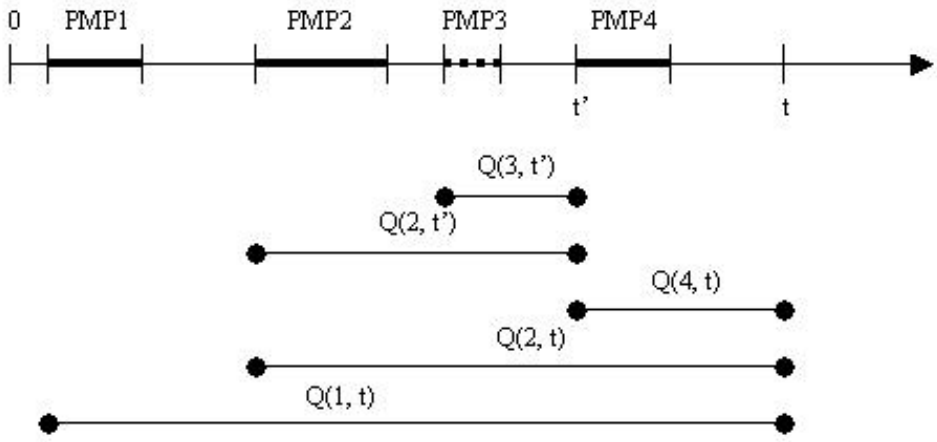

FIG. 6 – Exemple de fonctionnement de l'algorithme SSA

La figure n° 6 illustre le fonctionnement de SSA par un exemple. Dans celui-ci, lorsque le PMP2 commence, l'algorithme valide le PMP1 avec la première inégalité. Au début du PMP3, la seconde inégalité valide le deuxième processus. Ensuite, au temps t', on obtient $Q(3, t') \leq Q(2, t')$. Le PMP3 s'est révélé inefficace, la politique obtenue par le PMP2 est alors restaurée. Les modifications de ce dernier sont conservées car $Q(2, t') > Q(1, t')$. Enfin, au temps t, le PMP4 est conservé car $Q(4, t) > Q(2, t) > Q(1, t)$.

4 Conclusion

Pour valider ce travail, une librairie complète de classeurs (anticipatoires ou non) et d'environnements d'expérimentation nommé LCSTalk[9] a été développée au sein de notre équipe. Cette

[9] http ://www.iutc3.unicaen.fr/serge/LCSTalk

plate-forme nous permet d'évaluer notre architecture d'apprentissage dans diverses configurations (markoviennes ou non, mono ou multi-agents). Cette plateforme écrite en Smalltalk sera prochainement disponible librement.

L'algorithme de méta-apprentissage que nous proposons a été implémenté avec LCSTalk et les résultats d'expérimentations sont en cours d'analyse. Le problème qui se pose à nous est celui des critères de comparaisons à utiliser entre les différentes approches de classeurs ? comment mesurer les performances d'un classeur et ce qu'il apprend ? Quel est l'apport de l'anticipation et de son apprentissage ? Comment paramétrer convenablement le LCS ? Ces interrogations rejoignent celles soulevées récemment dans la communauté des classeurs par T. Kovacs ([Kov02]). Les paramètres définis dans CLRI[10] par Jose Vidal ([VD03]) sont peut-être une voie pour résoudre ce problème.

Nous envisageons de mettre en œuvre prochainement les algorithmes de meta-apprentissage décrit dans ce papier sur des applications plus concrètes que celles que nous avons utilisé jusqu'à présent pour nos tests (de type labyrinthe en 2D). Une des applications envisagée est celle de la reconfiguration de robots modulaires du projet MAAM[11] financé par Robea et dans lequel, notre équipe participe. L'idée est également de tester ces algorithmes en ligne dans un contexte embarqué (i.e. où le robot dispose de ressources limitées de calcul et de mémoire).

D'un point de vue théorique, une possibilité fructueuse de poursuite de ce travail serait d'utiliser des classeurs anticipatoires plus performant qu'ACS comme ceux introduit récemment par Pierre Gérard dans sa thèse ([Ger02]) : YACS (Yet Another Classifier System) et MACS (Modular Anticipatory Classifier System).

Références

[BGS99] Martin Butz, D. E. Goldberg, et Wolfgang Stolzmann. New challenges for anticipatory classifier system : Hard problems and possible solution. Rapport technique 99019, ILLiGAL, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, 1999.

[BSG03] Martin V. Butz, Olivier Sigaud, et Pierre Gérard. Internal models and anticipations in adaptive learning systems. Dans Martin V. Butz, Olivier Sigaud, et Pierre Gérard, éditeurs, *Anticipatory Behavior in Adaptive Learning Systems*, volume 2684 de *Lecture Notes in Artificial Intelligence*, pages 86–109. Springer-Verlag, 2003.

[But01] Martin Butz. An implementation of the anticipatory classifier system ACS2 in C++. Rapport technique 2001026, ILLiGAL, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, 8 2001.

[Dav96] Paul Davidsson. A linearly quasi-anticipatory autonomous agent architecture : Some preliminary experiments. Dans *Distributed Artificial Intelligence Architecture and Modelling*, numéro 1087 dans Lecture Notes in Computer Science, pages 189–203. Springer Verlag, 1996.

[Ger02] Pierre Gerard. *Systèmes de Classeurs : Étude de L'apprentissage Latent.* Thèse de doctorat, spécialité informatique, Université Pierre et Marie Curie Paris 6, 2002.

[Hof93] Joachim Hoffmann. *Vorhersage und Erkenntnis.* Hogrefe, 1993.

[10] C : Change rate, L : Learning rate, R : Retention rate, I : Impact

[11] MAAM signifie Molécule = Atome ou (Atome+Molécule). On trouvera un descriptif complet du projet à cet URL : http ://www.iutc3.unicaen.fr/serge/ProjetMAAM

[Kov02] T. Kovacs. What should a classifier system learn and how should we measure it ? *Soft Computing*, 6 :171–182, 2002.

[MH94] Dorigo Marco et Bersini Hugues. A comparison of Q-learning and classifier systems. Dans *Proceedings of From Animals to Animats, Third International Conference On Simulation of Adaptive Behavior*, 1994.

[RA03] Gabriel Robert et Guillot Agnès. MHiCS, a modular and hierarchical classifier systems architecture for bots. Dans Q. Mehdi, N. Gough, et S. Natkin, éditeurs, *Game-on 2003*, pages 140–144, 2003.

[Ros85] Robert Rosen. *Anticipatory Systems*. Pergamon Press, Oxford, 1985.

[Sch96] Jürgen Schmidhuber. A general method for incremental self-improvement and multi-agent learning. Dans X. Yao, éditeur, *Evolutionary Computation : Theory and Applications*. Scientific Publishing Company, 1996.

[Sew49] John P. Seward. An experimental analysis of latent learning. *Journal of Experimental Psychology*, 39 :177–186, 1949.

[Sti03] Serge Stinckwich. L'anticipation dans les systèmes complexes. conséquences pour la simulation. Dans *Le statut épistémologique de la simulation (1Oième Journées de Rochebrune : Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels)*, pages 261–277, 2003.

[Sto98] Wolfgang Stolzmann. Anticipatory classifier systems. Dans John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Deb Kalyanmoym, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, et Rick Riolo, éditeurs, *Genetic Programming 3 : Proceedings of the Third Annual Conference, University of Wisconsin, Madison*, pages 658–664. Morgan Kaufmann, 1998.

[Sut91] R. S. Sutton. Reinforcement learning architectures for animats. Dans Meyer J.-A. et S.W. Wilson, éditeurs, *From animals to animats : Proceedings of the First International Conference on Simulation of Adaptative Behavior*, pages 288–296. MIT Press, 1991.

[SZW96] Jürgen Schmidhuber, J. Zhao, et M. Wiering. Simple principle of metalearning. Rapport technique 69-96, IDISIA, Corso Elvezia 36, CH-6900-Lugano, Switzerland, 6 1996.

[Tol32] Edward Tolman. *Purposive Behavior in Animals and Men*. Appletown, New York, 1932.

[TRS00] K. Takadama, T. Rerano, et K. Shimohara. Learning classifier systems meet multiagent environments. Dans L. Lanzi, W. Stolzmann, et S.W. Wilson, éditeurs, *Third International Workshop on Learning Classifier Systems (IWLCS-200)*, pages 192–210, 2000.

[VD03] José M. Vidal et Edmund H. Durfee. Predicting the expected behavior of agents that learn about agents : the CLRI framework. *Autonomous Agents and Multi-Agent Systems*, 6(1) :77–107, 1 2003.

[Wat89] C. Watkins. *Learning from Delayed Rewards*. Thèse de doctorat, University of Cambridge, England, 1989.

[ZS96] J. Zhao et Jürgen Schmidhuber. Incremental self-improvement for life-time multi-agent reinforcement learning. Dans Pattie Maes, Maja Mataric, Jean-Arcady Meyer, Jordan Pollack, et Stewart W. Wilson, éditeurs, *From Animals to Animats 4 : Proceedings*

of the Fourth International Conference On Simulation of Adaptive Behavior, Cambridge, MA, pages 516–525. MIT Press, Bradford Books, 1996.