Généralisation et apprentissage latent dans les systèmes de classeurs

Pierre Gérard[*,] — Olivier Sigaud[**]**

[*] *LIP6, AnimatLab, 8 rue du Capitaine Scott 75015 PARIS*
{pierre.gerard, olivier.sigaud}@lip6.fr

[**] *Dassault Aviation, DGT/DPR/ESA, 78 Quai Marcel Dassault 92552 St-Cloud*
pierre.gerard@dassault-aviation.fr

RÉSUMÉ. *La principale originalité des systèmes de classeurs vis-à-vis d'autres techniques d'apprentissage par renforcement est leur capacité de généralisation. Dans cet article, nous présentons les différences entre plusieurs types de systèmes de classeurs. Alors que certains traitent directement les problèmes d'apprentissage par renforcement, d'autres apprennent un modèle de la dynamique des interactions entre l'agent et son environnement. Un tel modèle peut être utilisé pour accélérer l'apprentissage par renforcement proprement dit, si bien que ces deux types d'apprentissage sont complémentaires. Nous montrons ici que le concept de généralisation est très différent selon que le système anticipe (comme ACS ou YACS) ou non (comme XCS). Nous montrons en outre certaines limitations du formalisme utilisé par ACS etYACS, et nous en proposons un nouveau, propre à modéliser de nouvelles régularités. Nous décrivons MACS, un nouveau système de classeurs qui utilise ce formalisme pour l'apprentissage latent et nous comparons expérimentalement MACS avec YACS.*

ABSTRACT. *The main originality of Learning Classifier Systems (LCS) with respect to other Reinforcement Learning (RL) techniques is to provide the system with a generalization capability. In this paper, we highlight the differences between two kinds of LCSs. Some are used to directly perform RL while others learn latently a model of the interactions between the agent and its environment. Such a model can be used to speed up the core RL process. Thus, these two kinds of learning process are complementary. We show here how the notion of generalization differs whether the system anticipates (like ACS and YACS) or not (like XCS). Moreover, we show some limitations of the ACS and YACS formalism, and propose a new one which allows to represent new regularities. We describe MACS, a new LCS using this formalism for latent learning and we compare experimentally MACS with YACS.*

MOTS-CLÉS : *systèmes de classeurs, apprentissage latent, anticipation, généralisation.*

KEYWORDS: *learning classifier systems, latent learning, anticipation, generalization.*

1. Introduction

L'apprentissage par renforcement étudie l'apprentissage d'un comportement par un agent adaptatif impliqué dans une boucle sensori-motrice. Un tel agent perçoit sa situation dans l'environnement grâce à des senseurs et utilise ces perceptions pour choisir une action motrice. Il reçoit en retour une récompense scalaire et la nouvelle situation résultant de l'action accomplie. La tâche dévolue à l'agent est l'apprentissage d'une *politique* optimale. C'est-à-dire qu'il doit apprendre quelle action effectuer dans chaque situation de manière à maximiser le cumul des récompenses au long terme, dans un environnement *a priori* inconnu.

Le fait de chercher à optimiser l'action n'exclut pas la prise en compte de l'extraction de connaissances sur le problème que l'agent doit résoudre. En effet, une façon d'accélérer l'apprentissage de la politique est d'apprendre simultanément à cette politique un modèle de la dynamique des interactions entre l'agent et son environnement [SUT 98]. Dans cet article, nous allons précisément nous concentrer sur la problématique de l'extraction d'un modèle le plus compact possible des interactions entre l'agent et son environnement.

Ce qui caractérise l'apprentissage par renforcement vis-à-vis d'autres techniques d'apprentissage est le fait que l'agent doit apprendre à partir de son expérience. Dans ce cadre, l'agent ne dispose pas d'une base de données d'instances sur laquelle appuyer son apprentissage. Il doit apprendre sur la base de son expérience : il doit affiner son modèle et son comportement de manière incrémentale, en prenant en compte les unes après les autres les données fournies par ses senseurs au fil de son action. En particulier, la nécessité d'adaptation à un environnement éventuellement changeant lui interdit ainsi d'utiliser une mémoire de la totalité de son expérience.

L'idée d'extraire un modèle de la dynamique des interactions avec l'environnement prend sa source dans des travaux de psychologie. Ainsi, en s'opposant aux théories béhavioristes, Tolman [TOL 32] affirmait qu'apprendre peut être défini comme le processus qui permet de découvrir ce qui mène à quoi, ce qui le conduisait à affirmer que les animaux développent une représentation interne de leur interaction avec l'environnement. Toujours dans l'étude du comportement animal, Seward [SEW 49] a fait le premier la preuve de l'existence d'un *apprentissage latent* : un apprentissage qui intervient sans aucune récompense ou punition.

Ainsi, pour un agent logiciel faisant face à un problème dont la résolution nécessite plusieurs pas de temps, la conséquence d'une action ne consiste pas seulement en une récompense, mais aussi en une nouvelle situation. Dès lors, l'agent peut apprendre à *anticiper* ce qui arrive immédiatement après l'exécution d'une action et il peut ainsi construire un modèle des transitions entre situations successives. Cette information peut être utilisée à des fins de planification, ou encore pour accélérer l'apprentissage de la politique en réalisant des séquences d'actions simulées comme dans [SUT 98]. Un tel modèle peut aussi être utilisé dans le cadre de la programmation dynamique [BEL 57]. L'apprentissage latent est ainsi complémentaire de l'apprentissage d'une

politique et permet de mettre à jour la politique très rapidement dès que les récompenses changent.

Dans le cadre général de l'apprentissage par renforcement, la capacité à *généraliser* est la principale originalité des systèmes de classeurs vis-à-vis d'autres systèmes. Cette capacité de généralisation permet de représenter plusieurs situations avec une description commune, de sorte que la représentation du problème devient plus compacte. Nous allons montrer comment, dans le cadre des systèmes de classeurs, une nouvelle tendance focalisée sur l'apprentissage latent conduit à reconsidérer le concept même de généralisation.

Dans cet article, nous étudions l'apprentissage latent en environnement markovien, en tirant parti de la généralisation telle qu'elle est conçue dans les systèmes de classeurs. Cela revient à se poser le problème de l'extraction du modèle le plus compact possible des interactions avec l'environnement. Nous proposons en particulier une nouvelle manière de considérer la généralisation et nous montrons que le concept sous-jacent est très différent selon que le système anticipe ou non.

Dans la section 2, à travers XCS [WIL 95], nous présentons brièvement la généralisation dans le cadre des systèmes de classeurs classiques comme une extension du *Q-learning* [WAT 89]. Dans la section 3, nous présentons le formalisme utilisé par ACS[1] [STO 98] et par YACS[2] [GER 01b]. Ce formalisme permet de combiner généralisation et apprentissage latent. Nous donnons également un aperçu des principaux mécanismes de YACS. Dans la section 4, nous présentons un type de régularités dans la dynamique des interactions avec l'environnement que ni ACS ni YACS ne sont capables de représenter et nous proposons un nouveau formalisme qui permet d'exprimer ce type de régularités. Dans la section 5, nous décrivons MACS : un nouveau système de classeurs qui utilise ce formalisme et dans la section 6, nous donnons des résultats expérimentaux propres à comparer ce nouveau système à YACS.

2. Du Q-learning aux systèmes de classeurs

Le premier système de classeurs dédié aux problèmes d'apprentissage par renforcement, CS1, est décrit dans [HOL 78]. Wilson [WIL 95] a introduit dans ce cadre un algorithme similaire au *Q-learning* [WAT 89], pour remplacer le traditionnel *Bucket Brigade* [HOL 85]. Ces travaux ont amorcé un regain d'intérêt pour les systèmes de classeurs puisque XCS permet de résoudre les problèmes de sur-généralisation des précédents systèmes [WIL 89], grâce à son approche fondée sur la qualité de la *prédiction* de la récompense attendue. Dans cette section, nous montrons comment un système de classeurs comme XCS peut être vu comme une extension du *Q-learning* offrant des capacités de généralisation.

1. Anticipatory Classifier System.
2. Yet Another Classifier System.

2.1. *Le Q-learning*

Une façon courante de représenter formellement les relations entre l'agent et l'environnement dans un problème d'apprentissage par renforcement est d'utiliser un Processus de Décision Markovien défini par :

– un espace fini d'état S ;

– un ensemble fini d'actions A ;

– une fonction de transition $T : S \times A \rightarrow \Pi(S)$ où $\Pi(S)$ est une distribution de probabilités sur l'espace des états S ;

– une fonction de récompense $R : S \times A \times S \rightarrow \Re$ qui associe une récompense immédiate à chaque transition possible.

Parmi ceux qui sont fondés sur cette représentation, un des algorithmes d'apprentissage par renforcement les plus populaires est le *Q-learning* [WAT 89]. Cet algorithme met à jour une *Q-table* qui représente une fonction de qualité $Q : S \times A \rightarrow \Re$. $Q(s, a)$ représente le cumul des récompenses attendues dans le futur quand l'agent exécute l'action a dans l'état s et agit ensuite selon la meilleure politique.

À l'étape de temps t, les qualités sont mises à jour selon la formule suivante, basée sur l'équation de Bellman utilisée en programmation dynamique [BEL 57] :

$$Q(s_{t-1}, a_{t-1}) \leftarrow (1-\beta)Q(s_{t-1}, a_{t-1}) + \beta(r_t + \gamma max_{a \in A} Q(s_t, a)) \qquad (1)$$

où s_t est l'état résultant de l'action a_{t-1} dans l'état précédent s_{t-1}. β est le taux d'apprentissage d'une règle de Widrow-Hoff[3] et r_t est la récompense associée. $\gamma \in [0, 1]$ est le facteur d'atténuation utilisé dans l'équation de Bellman [BEL 57]. Si ce facteur est strictement inférieur à 1, il a pour effet la diminution des qualités à mesure que les paires (s, a) mènent à un état distant de toute source de récompense immédiate.

2.2. *Le formalisme des systèmes de classeurs*

Dans les problèmes étudiés par les systèmes de classeurs, les *états* se composent de plusieurs attributs représentant autant de propriétés perceptibles de l'environnement. Nous préférons alors parler des *situations* perçues plutôt que d'*états*. Par exemple, un agent dans un monde de cases comme un labyrinthe peut percevoir huit attributs, chacun exprimant l'absence ou la présence d'un obstacle sur une des huit cases adjacentes. Ainsi, une situation est un ensemble ordonné de plusieurs valeurs discrètes, une pour chacun des attributs perçus. Les actions sont aussi divisées en attributs représentant différents effecteurs.

3. Une règle de Widrow-Hoff utilise un taux d'apprentissage $\beta \in [0, 1]$. Par une telle règle, un scalaire x est incrémenté selon la formule : $x \leftarrow (1-\beta)x + \beta$. Il est décrémenté selon la formule $x \leftarrow (1-\beta)x$.

Dès lors, il est possible d'agréger plusieurs situations en une seule condition en utilisant des symboles "#"[4]. Un tel symbole ne fait porter aucune contrainte sur la valeur de l'attribut considéré.

Nous disons qu'un symbole "#" est *compatible* avec n'importe quelle valeur particulière de l'attribut. La transformation d'un attribut en symbole "#" rend la condition plus générale (elle est compatible avec un plus grand nombre de situations). Par exemple, en supposant que les attributs des situations peuvent seulement prendre des valeurs `0` ou `1`, la condition`[#01]` est plus générale et elle est compatible avec les situations spécialisées `[001]` et `[101]`. La condition `[#0#]` est compatible avec 4 situations : `[000]`, `[001]`, `[100]` et `[101]`. Deux attributs sont dits *incompatibles* lorsqu'ils sont tous les deux spécialisés mais qu'ils prennent une valeur différente. Deux conditions sont incompatibles si au moins deux attributs correspondants l'un à l'autre sont incompatibles.

Dans [LAN 00a], Lanzi montre comment il est possible de passer dans représentation tabulaire (comme dans le cas du *Q-learning*) d'un problème à une représentation à base de règles (ou *classeurs*). Alors que le *Q-learning* tabulaire considère des triplets $(s, a, q) \in S \times A \times \Re$, les systèmes de classeurs comme XCS considèrent des classeurs `Condition-Action-p`. La partie `Condition` peut contenir des symboles "#" et être compatible avec plusieurs situations, l'action `Action` représente la commande sur les effecteurs, et `p` est la prévision du cumul des récompenses à long terme (ou récompense attendue) quand l'action considérée est choisie dans n'importe laquelle des situations compatibles avec la condition. Pour maximiser la récompense attendue, quand il perçoit une situation, le système choisit de préférence les actions proposées par les classeurs avec la prévision la plus élevée parmi les classeurs dont la partie `Condition` est compatible avec la situation.

Grâce aux symboles "#", il est possible de construire un modèle des récompenses attendues dont la taille en nombre de règles est plus petite que le nombre de triplets possibles d'une représentation tabulaire. Le plus souvent, la prévision de la récompense attendue `p` est calculée grâce à divers algorithmes de rétropropagation comme le *Bucket Brigade* [HOL 85]. C'est XCS [WIL 95] qui introduit l'utilisation de la formule (1) dans les systèmes de classeurs.

2.3. *Présentation d'XCS*

Le problème de la généralisation dans un système de classeurs consiste à trouver les parties `Condition` et `Action` de sorte que les symboles "#" soient bien placés. Pour ce faire, les systèmes de classeurs utilisent souvent des algorithmes génétiques [HOL 76, GOL 89] pour faire évoluer une population de classeurs : chaque classeur est est évalué tout au long de l'interaction de l'agent avec l'environnement. Ces algorithmes utilisent des opérateurs génétiques classiques comme le croisement [5] et la

4. Grâce aux symboles "#", certaines valeurs d'attributs peuvent être omises.
5. Le nouveau classeur est une combinaison de différents segments de ses parents.

mutation[6]. Ils utilisent aussi un mécanisme de sélection qui repose sur des *valeurs sélectives* : un classeur avec une basse valeur sélective est supprimé et ne pourra pas se reproduire.

La valeur sélective des classeurs est ajustée à chaque pas de temps et le système ne conserve pas de mémoire des informations produites dans le passé par la boucle sensori-motrice. Le processus d'apprentissage est donc incrémental et permet d'aborder des environnements changeants.

Dans un système de classeurs traditionnel comme ZCS [WIL 94], la valeur sélective d'un classeur est égale à la récompense attendue. Les classeurs avec la plus petite récompense attendue sont ainsi supprimés. Cette suppression pose problème parce que le facteur d'atténuation (γ dans l'équation 1) a pour effet une décroissance des récompenses attendues à mesure que les classeurs sont applicables loin d'une récompense immédiate[7]. Le problème de surgénéralisation qui en résulte est résolu par XCS en décorrélant la récompense attendue et la valeur sélective. Dans XCS, trois mesures sont associées à chaque classeur au lieu d'une seule :

– une récompense attendue `p`,

– une erreur de prévision `e`,

– une valeur sélective `f` qui est fonction de l'erreur de prévision et représente la *précision* du classeur.

De cette façon, la valeur sélective d'un classeur ne dépend pas de l'importance de la récompense attendue, mais de sa capacité de la prévoir avec précision.

Kovacs [KOV 97] a montré que XCS découvre des classeurs *fiables* et *aussi généraux que possible*. Un classeur est dit fiable si, dans chaque situation correspondant à la condition, choisir l'action proposée mène toujours effectivement au même cumul de récompense. Il est dit aussi général que possible dès lors qu'il ne peut pas contenir plus de symboles "`#`" tout en restant fiable. Un classeur compatible avec plusieurs paires (s, a) à différents niveaux de récompense attendue n'est pas fiable parce qu'il est trop général.

De plus, XCS tend vers des ensembles de classeurs *complets*, où à chacune des paires (s, a) possibles correspond au moins un classeur puisque la valeur sélective se fonde sur l'exactitude de la prévision plutôt que sur l'optimalité de l'action.

Ainsi, XCS construit une carte complète des récompenses attendues et utilise la généralisation pour réduire le nombre de classeurs. Il décide de la fiabilité d'un classeur de telle manière que la généralisation ne peut se produire que parmi les paires (s, a) à un même niveau de récompense attendue. Nous montrons dans la section 3 comment ajouter une partie `Effet` aux classeurs mène à une conception différente de

6. Chaque attribut d'un nouveau classeur créé peut prendre aléatoirement une valeur quelconque.
7. Le mécanisme de taxes dans l'algorithme *Bucket Brigade* mène au même problème.

la généralisation qui permet de généraliser parmi les paires (s, a) à différents niveaux de récompense attendue.

D'autres systèmes d'apprentissage par renforcement, comme celui présenté dans [DZE 01], offrent une représentation compacte et explicite de la solution obtenue. Ce système d'apprentissage par renforcement relationel produit des arbres de décision d'une manière comparable à C4.5 [QUI 93] en permettant d'aborder les problèmes d'apprentissage par renforcement dont la résolution nécessite plusieurs pas de temps. Toutefois, ce système ne possède pas de mécanisme propre à reconsidérer des spécialisations sous-optimales et ne peut donc pas réaliser d'apprentissage incrémental pour l'instant. L'incrémentalité permet au contraire à XCS de conserver un modèle optimal dans un environnement changeant, grâce à sa capacité à regénéraliser.

Par ailleurs, sur des problèmes d'extraction de connaissances, Bernardó [BER 01] a montré que pour une quinzaine de bases de données usuelles, XCS produit le plus souvent des classifications de meilleure qualité que 6 autres algorithmes classiques, dont C4.5 [QUI 93]. En effet, des extensions non binaires d'XCS peuvent aussi être utilisées pour apprendre de façon supervisée une classification à partir d'une base d'exemples[8].

Sans avoir recours à une mémoire d'instances, XCS est donc capable de trouver une classification explicite offrant une bonne généralisation, tout en restant capable de réaliser un apprentissage incrémental.

3. Un aperçu de YACS

Certains problèmes ne nécessitent qu'une seule action pour être résolus : l'agent perçoit sa situation, choisit une action et reçoit une récompense scalaire qui indique dans quelle mesure l'action était adaptée. Mais, dans des problèmes nécessitant plusieurs actions successives, l'agent ne reçoit d'indication sur la pertinence des actions choisies que quelques pas de temps plus tard. Dans des problèmes de ce type, en plus d'une récompense, l'agent reçoit également la nouvelle situation résultant de sa dernière action. Dans ce cadre, il est possible d'utiliser un système de classeurs pour apprendre à anticiper les nouvelles situations.

Pour apprendre un tel modèle de la dynamique des interactions entre l'agent et son environnement, Holland [HOL 90] a proposé une approche implicite. Avec les messages internes, il est possible d'utiliser des étiquettes qui indiquent si une action postée sur la liste de messages indique une action effective ou une anticipation. Riolo [RIO 91] a mis en application cette idée dans CFSC2 et a exhibé une capacité d'apprentissage latent. Un lien plus explicite entre situations successives est utilisé dans CXCS [TOM 00], où les classeurs sont liés de manière statistique. Ces liens offrent une représentation implicite des transitions entre situations.

8. Il suffit d'ignorer l'action et de donner une récompense différente pour chaque classe [WIL 01].

Classeur		
[#0#1]	[0]	[=10=]
[0011]		[0101]
[1011]		[1101]
Situations	Action	Anticipations

Tableau 1. *Illustration du mécanisme d'anticipation dans YACS*

Par ailleurs, l'ALP (*Anticipatory Learning Process*) utilisé dans ACS est une application de la théorie du *contrôle du comportement par l'anticipation* présentée en psychologie par Hoffmann [HOF 93]. YACS [GER 01b] utilise la même idée et les deux systèmes créent des classeurs explicites de la forme `Condition-Action-Effet`. Ce formalisme est semblable à l'approche DynaQ+ de Sutton [SUT 91] ou aux règles `Context-Action-Result` de Drescher [DRE 91], mais avec la possibilité de généraliser plusieurs situations.

3.1. *Le formalisme de ACS et de YACS*

La partie `Effet` représente les effets de l'action `Action`, dans les situations compatibles avec la partie `Condition`. Elle représente les changements perçus dans l'environnement. Dans ACS et dans YACS comme dans la plupart des systèmes de classeurs, une partie `Condition` est une situation qui peut contenir des symboles "`#`". Une partie `Effet` est aussi composée de plusieurs attributs mais peut contenir des symboles "`=`" au lieu des symboles "`#`". Un tel symbole indique que l'attribut de la situation correspondant au symbole "`=`" demeure inchangé lorsque l'action proposée est choisie. Une valeur particulière dans la partie `Effet` signifie que la valeur de l'attribut correspondant *change* et prend la valeur indiquée, quelle que soit la partie `Condition`.

Considérons par exemple le classeur `[#0#1] [0] [=10=]`. Grâce aux symboles "`#`" dans sa partie `Condition`, il anticipe les effets de l'action [0] dans 4 situations possibles : (`[0001]`, `[0011]`, `[1001]` et `[1011]`). Selon la partie `Effet` :

– le premier attribut demeure inchangé quelle que soit la valeur initiale (`0` ou `1` à cause du symbole "`#`" dans la partie `Condition`) ;

– la valeur du deuxième attribut passera de `0` à `1` ;

– le troisième attribut prendra la valeur `1` quelle que soit la valeur initiale (quand les attributs peuvent prendre plus de deux valeurs, ce cas est différent du précédent) ;

– la valeur du dernier attribut reste `1`.

Ces cas sont illustrés dans la table 1.

Ce formalisme permet aux classeurs de représenter des régularités dans les interactions avec l'environnement, comme par exemple *« dans un monde de cases, quand l'agent perçoit un mur devant lui, quelles que soient ses autres perceptions, se dépla-*

cer vers l'avant le conduit à rester dans la même case (il heurte le mur), donc aucun changement ne sera perçu dans sa situation ».

Le processus d'apprentissage latent doit découvrir un ensemble de classeurs avec des parties `Condition` aussi générales que possible, tout en permettant de modéliser de façon fiable la dynamique des interactions de l'agent avec l'environnement. Comme dans XCS, un classeur est dit *aussi général que possible* s'il ne peut contenir aucun symbole "#" supplémentaire sans perdre sa fiabilité. Mais ici, il est dit *fiable* si, dans chaque situation compatible avec sa condition, choisir l'action proposée mène toujours aux mêmes changements entre deux situations successives. L'exactitude de la prévision dans ACS et YACS dépend de la situation attendue plutôt que d'une récompense comme dans XCS.

Ainsi, la *généralisation* ne désigne pas la même chose dans ACS et YACS que dans XCS. Dans ACS et YACS, la généralisation est possible grâce à l'utilisation simultanée des symboles "#" et "=". Elle permet de représenter des régularités dans les transitions entre les situations successives et permet au système :

– une forme d'*attention sélective* quand quelques situations peuvent être identifiées en prêtant attention à quelques attributs seulement ;

– de considérer plusieurs situations avec une seule situation, afin de réduire la taille du modèle de la dynamique de l'environnement.

Puisque YACS ne généralise pas en fonction de la récompense, il peut généraliser des situations qui sont à différents niveaux de récompense attendue. Mais, en conséquence, il est impossible de stocker des informations sur la récompense attendue au niveau des classeurs. La liste de classeurs modélise seulement les transitions dans l'environnement.

Afin d'apprendre une politique (l'apprentissage par renforcement proprement dit), YACS a besoin d'informations sur des situations spécifiques. Ainsi, il utilise une liste contenant chaque situation perçue pendant la "vie" de l'agent[9], avec un seul exemplaire de chaque situation déjà perçue. À chaque situation est associée la récompense attendue en atteignant la situation considérée. Cette liste contient uniquement les situations effectivement perçues plutôt que toutes les situations théoriquement possibles selon le nombre d'attributs et le nombre de valeurs qu'ils peuvent prendre. Dans un grand problème comme le problème multi-agent décrit dans [SIG 01], par exemple, le nombre de situations effectivement perçues est de 290 alors que le nombre de situations théoriquement possibles est de 8 192.

9. Pour réduire la taille de cette liste, on pourrait doter YACS d'un nouveau mécanisme de généralisation qui se fonde sur la récompense attendue, comme dans XCS.

3.2. *Les heuristiques utilisées dans YACS*

Dans les sections suivantes, nous présentons brièvement les principaux mécanismes de YACS relatifs à l'*apprentissage latent*, tels qu'ils ont été décrit plus précisément dans [GER 01b] et [GER 01a].

Pour découvrir des classeurs adaptés, ACS et YACS utilisent tous les deux l'information offerte par la succession des situations. Pour mieux prendre en compte cette information disponible, ils reposent sur des heuristiques plutôt que sur des algorithmes génétiques comme c'est généralement le cas dans les systèmes de classeurs. Les algorithmes génétiques ont l'avantage d'être plus généraux, mais ils n'utilisent l'information offerte par la boucle sensori-motrice que pour valider *a posteriori* des classeurs construits aléatoirement. Au contraire, YACS explore l'espace de solutions rationnellement, de manière à pouvoir aborder des problèmes aussi grands que celui décrit dans [SIG 01]. Comme il a été discuté dans [GER 01b], la principale différence entre ACS et YACS est que YACS décorrèle fortement la découverte des parties `Condition` et `Effet`.

3.2.1. *Ajout de parties* `Effet`

De nouvelles parties `Effet` sont créées par comparaison directe entre situations successives et permettent de créer de nouveaux classeurs rendant compte de l'effet de certaines actions sous certaines conditions.

En complément de ce mécanisme, YACS met à jour une trace T de marqueurs booléens *succès* et *échec* mémorisant les succès et les erreurs d'anticipation de chaque classeur. Cette trace fonctionne comme une file de longueur finie m.

3.2.2. *Sélection des classeurs fiables*

De manière à établir un ensemble de classeurs qui anticipent avec précision, YACS dispose d'un mécanisme de suppression des classeurs qui anticipent mal.

La trace T des marqueurs rend compte des capacités d'anticipation d'un classeur :

– si la trace d'un classeur est pleine et si elle contient seulement des marqueurs *échec*, alors YACS suppose que le classeur anticipe toujours mal, qu'il n'est pas fiable du tout, et le retire ;

– si la trace est pleine et si elle contient de bons et mauvais marqueurs, nous disons que le classeur *oscille* parce que sa partie `Condition` est trop générale. Dans ce cas, la condition doit être spécialisée davantage.

3.2.3. *Spécialisation des conditions*

Une partie `Condition` doit être aussi générale que possible de façon à représenter des régularités dans les transitions entre les situations successives. Mais elle doit être suffisamment spécifique de sorte que le classeur n'oscille pas. Le mécanisme de

spécialisation spécialise de manière incrémentale les parties `Condition` des classeurs afin d'atteindre le juste niveau de généralisation.

A l'initialisation, YACS ne fait aucune distinction entre les situations (les parties `Condition` ne contiennent que des symboles "`#`") et procède incrémentalement à des spécialisations des parties `Condition`. Ces spécialisations sont guidées par l'expérience au lieu d'être décidées au hasard puis évaluées *a posteriori* comme c'est le cas dans les algorithmes génétiques.

Le mécanisme de spécialisation de YACS utilise l'opérateur *mutspec* présenté par [DOR 94]. Cet opérateur choisit un attribut général de la partie `Condition` d'un classeur et produit un nouveau classeur pour chaque valeur spécifique possible de l'attribut choisi.

YACS améliore la sélection des attributs à spécialiser en utilisant une estimation i_s[10] associée à chaque symbole dans la partie `Condition` de chaque classeur. Cette valeur estime à quel point la spécialisation de l'attribut considéré permettrait de scinder l'ensemble des situations compatibles avec la partie `Condition` en plusieurs sous-ensembles de cardinalité égale.

3.2.4. *Généralisation des conditions*

Même si le mécanisme de spécialisation est prudent, il peut produire des classeurs avec une partie `Condition` à un niveau de généralisation sous-optimal. C'est surtout le cas au début de l'apprentissage, lorsque YACS spécialise alors qu'il n'a pas perçu beaucoup de situations parmi celles qui sont possibles. Ainsi, un mécanisme de généralisation est nécessaire pour revenir sur des spécialisations indues.

Comme le mécanisme de spécialisation, le mécanisme de généralisation met en œuvre des heuristiques de manière à utiliser autant que possible l'information de la boucle sensori-motrice. Ainsi l'approche de YACS diffère de celle d'ACS puisque son mécanisme de généralisation n'emploie pas d'algorithme génétique comme c'est le cas dans [BUT 00].

La généralisation repose elle aussi sur des estimations. Une évaluation i_g[11] est associée à chaque attribut spécialisé de la partie `Condition`. Il évalue dans quelle mesure la partie `Effet` du classeur demeurerait fiable si l'attribut était considéré comme non pertinent.

Ces estimations sont utilisées pour créer de nouveaux sous-ensembles de classeurs à partir de sous-ensembles de classeurs initiaux avec des parties `Action` et `Effet` identiques. Les nouveaux classeurs sont plus généraux, leur nombre est plus petit et ils n'entrent pas en conflit avec d'autres classeurs du système.

10. Qui mesure l'amélioration attendue par la spécialisation de l'attribut considéré.
11. Qui mesure l'amélioration attendue par la généralisation de l'attribut considéré.

4. Représenter de nouvelles régularités

4.1. *Régularités non modélisables par ACS et YACS*

Jusqu'ici, nous avons expliqué comment YACS réalise un apprentissage latent propre à établir un modèle des transitions dans l'environnement. Il établit une liste de classeurs, chacun d'eux représentant une ou plusieurs transitions. La généralisation permet de représenter des régularités dans la dynamique des interactions avec l'environnement. A ce titre, optimiser la généralisation, c'est permettre au système d'extraire la connaissance la plus compacte possible du problème auquel il est confronté. Mais la capacité de généralisation d'un système dépend directement du type de régularités que lui permet de représenter son formalisme. Or les régularités représentées dans le formalisme d'ACS et YACS sont différentes de celles représentées par le formalisme de XCS. En effet, la fiabilité d'un classeur dépend de sa capacité à anticiper les changements dans les perceptions de l'environnement plutôt que de prévoir la récompense attendue.

Dans YACS, la généralisation est possible grâce à l'utilisation simultanée des symboles "`#`" dans la partie `Condition` et des symboles "`=`" dans la partie `Effet` des classeurs.

Comme ACS, YACS peut déterminer si un attribut particulier change ou pas, mais il ne peut pas représenter des régularités impliquant plusieurs attributs parce qu'il considère chaque situation et chaque condition comme un tout insécable. Pour éclaircir ce point, considérons un agent dans un monde de cases. Chaque case peut être libre ou contenir un obstacle. L'agent perçoit les huit cases adjacentes (`0` pour une case libre et `1` pour un obstacle). Les huit attributs perçus sont pris dans le sens des aiguilles d'une montre, en commençant par la case en face de l'agent. L'agent peut avancer ou tourner de 90 degrés dans le sens des aiguilles d'une montre ou dans le sens contraire. Tourner dans le sens des aiguilles d'une montre induit un décalage sur la situation perçue de deux positions vers la gauche des valeurs des attributs. Par exemple, l'agent peut modéliser des transitions comme `[11001100]` [$\curvearrowright$] `[00110011]`.

Dans ce cas, chaque attribut change et le formalisme de ACS et YACS est incapable de représenter de telles régularités. Pourtant, le changement dans les situations perçues est effectivement une régularité de la dynamique des interactions : quelle que soit la situation, quand l'agent tourne dans le sens des aiguilles d'une montre, la valeur du premier attribut devient celle du troisième, la valeur du deuxième devient celle du quatrième, etc.

La particularité d'une telle régularité est que la nouvelle valeur d'un attribut dépend de la valeur d'un autre attribut. La généralisation telle qu'elle est induite dans ACS et YACS par les symboles "`=`" ne permet pas de représenter de telles régularités. Dans le formalisme d'ACS et YACS, les seules régularités que l'on peut représenter sont celles où la nouvelle valeur d'un attribut dépend de la valeur correspondante dans la partie `Effet` et de la valeur précédente du même attribut dans la partie `Condition`, ce qui n'est pas le cas en règle générale.

Afin de surmonter ce problème, il est nécessaire de dissocier les attributs dans les parties `Effet` alors que, dans ACS et YACS, les classeurs anticipent tous les attributs en même temps.

4.2. *Vers un nouveau formalisme*

Afin de dissocier les attributs, nous proposons de considérer les parties `Effet` comme des situations, sans symboles "=" mais avec de nouveaux symboles " ?". Un symbole " ?" dans une partie `Effet` signifie que le classeur ne cherche pas à anticiper l'attribut en question. De cette façon, le classeur `[####1###]` [↷] `[ ? ?1 ? ? ? ? ?]` signifie que, juste après une rotation vers la droite, l'agent perçoit toujours un mur à sa gauche quand il a perçu un mur derrière lui, quelle que soit la valeur des autres attributs (grâce au symbole "#"). Par ailleurs, il ne fournit pas d'information au sujet des nouvelles valeurs d'autres attributs (grâce au symbole " ?"). Grâce à ces nouveaux symboles, un classeur peut n'anticiper que quelques attributs et le système global devient capable de représenter les nouvelles régularités en question, grâce à l'utilisation simultanée des symboles "#" et " ?". Par rapport au symbole "=", le symbole " ?" dans la partie `Effet` a un fonctionnement plus proche de celui de "#" dans la partie `Condition`. Pour autant, nous avons choisi de les distinguer parce que leur fonction est très différente au sein du classeur. Si un symbole "#" permet d'étendre le domaine de validité d'un classeur grâce à la généralisation, un symbole " ?" a au contraire vocation à restreindre le champ d'un classeur en ne lui permettant pas de se prononcer sur certains attributs.

Encore une fois, cette proposition pour un nouveau formalisme mène à une nouvelle conception de la généralisation. Comme d'habitude, un classeur est dit *aussi général que possible* s'il ne peut contenir plus de symbole "#" sans osciller. Mais maintenant, on dit qu'il est *fiable* si, dans chaque situation compatible avec sa condition, choisir l'action proposée mène toujours effectivement les attributs à prendre la valeur indiquée dans la partie `Effet`, à condition que l'attribut en question ne soit pas un symbole " ?".

Puisqu'un classeur n'anticipe pas une situation complète, l'unité anticipatrice n'est plus le classeur mais le système entier. Etant donné une situation et une action, un classeur seul ne peut pas prévoir la situation à venir. Le système a besoin d'un mécanisme qui *intègre* les propositions de chaque classeur afin de proposer une situation anticipée complète, sans aucun symbole " ?", comme décrit dans la table 2.

5. Description de MACS

Comme il a été défini dans la section 4, une partie `Effet` peut contenir plusieurs symboles " ?" et plusieurs valeurs spécifiques. Mais nous avons adopté un point de vue radical en n'autorisant qu'une et une seule valeur spécialisée (différente de " ?")

[11001100]		← *Situation*
[1#######]	[⌒]	[? ? ? ? ? ? 1 ?]
[#1######]	[⌒]	[? ? ? ? ? ? ? 1]
[##0#####]	[⌒]	[0 ? ? ? ? ? ? ?]
[###0####]	[⌒]	[? 0 ? ? ? ? ? ?]
[####1###]	[⌒]	[? ? 1 ? ? ? ? ?]
[#####1##]	[⌒]	[? ? ? 1 ? ? ? ?]
[######0#]	[⌒]	[? ? ? ? 0 ? ? ?]
[#######0]	[⌒]	[? ? ? ? ? 0 ? ?]
[#######0]	[⌒]	[? ? ? ? ? 1 ? ?]
Anticipations →		[00110011]
		[00110111]

Tableau 2. *Dans le mécanisme d'intégration, le système examine les parties* `Effet` *de chaque classeur tel que sa partie* `Action` *correspond à l'action souhaitée et tel que sa partie* `Action` *est compatible avec la situation. La valeur de chaque attribut qui n'est pas un symbole " ? " est une possibilité pour cet attribut. Le procédé d'intégration établit toutes les situations à venir possibles en fonction des valeurs possibles de chaque attribut. Si tous les classeurs sont fiables, alors une seule valeur est proposée pour chaque attribut, et ce mécanisme propose seulement une anticipation possible. Dans le cas contraire, comme ce peut être le cas si les deux derniers classeurs sont en cours de validation, le mécanisme d'intégration propose plusieurs valeurs possibles. Ces valeurs sont déterminées en fonction de valeurs proposées par les classeurs pour chaque attribut*

dans une partie `Effet`. Ainsi, chaque classeur ne peut prévoir la valeur que d'un seul attribut.

Dans cette section, nous détaillons chacun des mécanismes d'apprentissage latent de MACS, un nouveau système de classeurs conçu pour utiliser le formalisme proposé dans la section 4. La plupart d'entre eux sont très semblables à ceux de YACS (voir [GER 01b] et [GER 01a]). Les changements ne concernent que l'apprentissage latent. La figure 1 offre une vue globale des différents mécanismes permettant à MACS de trouver des classeurs adéquats et généraux.

5.1. *Evaluation et sélection des classeurs fiables*

Pour que l'apprentissage latent élabore un modèle compact et fiable des interactions entre l'agent et l'environnement, il est nécessaire d'évaluer la fiabilité des classeurs et de les supprimer au besoin. A cette fin, deux valeurs entières `g` et `b` sont associées à chaque classeur :

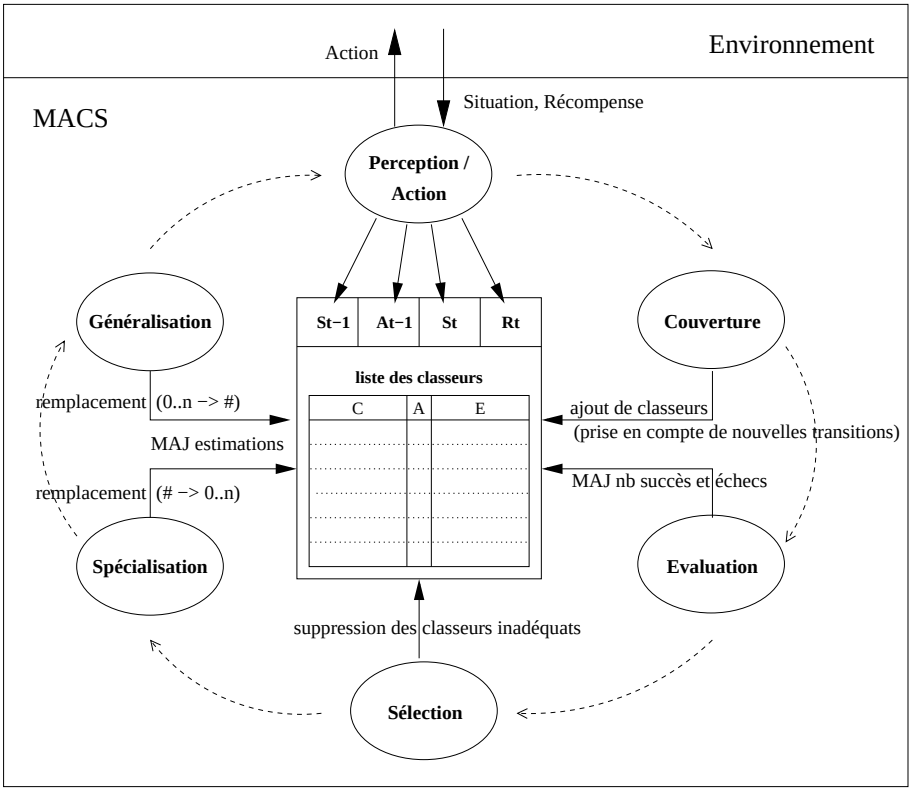

Figure 1. *Les différents mécanismes de MACS*

– `g` pour le nombre d'anticipations *correctes* depuis la création du classeur ;

– `b` pour le nombre d'erreurs dans l'anticipation depuis sa création.

MACS garde en mémoire la dernière situation perçue et la dernière action choisie. Ainsi, à chaque pas de temps, il connaît la situation actuelle S_t résultant de l'action A_{t-1} dans la situation S_{t-1}.

Avec cette information, YACS contrôle chaque classeur tel que sa partie `Condition` correspond à S_{t-1} et sa partie `Action` à A_{t-1} :

– si sa partie `Effet` est compatible avec S_t (un symbole " `?`" est compatible avec n'importe quelle valeur), alors `g` est augmenté ;

– si sa partie `Effet` n'est pas compatible avec S_t, alors `b` est augmenté.

Un classeur qui anticipe toujours mal après quelques évaluations est supposé inadéquat et il est supprimé. Ce nombre d'évaluations est un paramètre du système, noté e_r. Un classeur est supprimé lorsque `g`$= 0$ et `b`$= e_r$. Un autre paramètre du système

(e_a) indique combien d'évaluations sont nécessaires pour décider qu'un classeur est adéquat.

5.2. *Spécialisation des conditions*

Comme dans YACS, un classeur qui anticipe parfois bien et parfois mal *oscille* parce que sa partie `Condition` correspond à un trop grand nombre de situations. Dès lors, il doit être spécialisé. Ici encore, ce mécanisme n'est pas guidé par un algorithme génétique mais par des heuristiques utilisant des évaluations.

5.2.1. *Les estimations utilisées par MACS pour la spécialisation*

Une estimation i_s est associée à chaque attribut général (symbole "`#`") de la partie `Condition` de chaque classeur. Cette valeur estime dans quelle mesure spécialiser l'attribut correspondant permettrait de séparer en deux sous-ensembles de cardinalités égales l'ensemble des situations correspondant à la partie `Condition`.

Considérons un classeur qui anticipe les effets d'une action dans plusieurs situations. Si la valeur d'un attribut de la situation quand le classeur anticipe bien est toujours différente de sa valeur quand il anticipe mal, alors cet attribut est approprié pour distinguer les situations correspondant à la partie `Condition`. Ainsi, celle-ci doit être spécialisée selon cet attribut particulier et l'évaluation i_s doit avoir une valeur élevée.

Afin de calculer ces estimations, chaque classeur mémorise la situation $BadS$ précédant la dernière erreur d'anticipation et la situation $GoodS$ précédant la dernière anticipation correcte. Pour chaque classeur tel que sa partie `Condition` est compatible avec S_{t-1} et sa partie `Action` avec A_{t-1} :

– si le classeur anticipe bien, pour chaque attribut :

- si l'attribut de $BadS$ est égal à l'attribut correspondant de S_{t-1}, alors l'estimation correspondante i_s est *décrémentée* ;

- si l'attribut de $BadS$ est différent de l'attribut correspondant de S_{t-1}, alors l'estimation correspondante i_s est *incrémentée* ;

– si le classeur anticipe mal, pour chaque attribut :

- si l'attribut de $GoodS$ est égal à l'attribut correspondant de S_{t-1}, alors l'estimation correspondante i_s est *décrémentée* ;

- si l'attribut de $GoodS$ est différent de l'attribut correspondant de S_{t-1}, alors l'estimation correspondante i_s est *incrémentée*.

Les estimations i_s sont incrémentées et décrémentées selon une règle de Widrow-Hoff. Les valeurs initiales sont 0.5. Un attribut spécialisé prend la valeur par défaut 0.5.

5.2.2. *Le mécanisme de spécialisation de MACS*

Un classeur est dit *oscillant* quand $\mathtt{g}+\mathtt{b}>e_o$ et $\mathtt{g}\times\mathtt{b}>0$, où e_o est un paramètre du système représentant le nombre d'évaluations nécessaires pour déclarer un classeur comme oscillant.

Dès qu'un tel classeur est identifié, l'opérateur *mutspec* lui est appliqué. Il est ainsi remplacé par plusieurs classeurs plus spécialisés. Certains des classeurs produits par l'opérateur *mutspec* ne seront pas fiables, mais ils seront éliminés par le mécanisme de *sélection* des classeurs fiables. Un nouveau classeur dont la partie `Condition` ne correspond à aucune des situations déjà perçues n'est pas ajouté. Cette propriété peut être vérifiée en consultant la liste `P` des situations déjà perçues pendant la vie de l'agent (voir la section 3).

Contrairement à l'opérateur *mutspec* d'origine [DOR 94], l'attribut à spécialiser n'est pas choisi aléatoirement mais en fonction des évaluations i_s : l'attribut spécialisé est celui dont la spécialisation est la plus susceptible d'améliorer le système.

Le mécanisme de spécialisation de MACS est différent de celui de YACS en cela que ce dernier est plus prudent. Il attend que plusieurs classeurs *oscillent* avant de les spécialiser. Dès lors, les estimations de chacun des classeurs sont prises en compte pour décider de l'attribut à spécialiser. Par contre, MACS spécialise les classeurs un par un, sans attendre de confronter les estimations de plusieurs classeurs. Si la prudence de YACS garantit un meilleur guidage de la spécialisation, elle peut en revanche nuire à la vitesse d'apprentissage. C'est notamment le cas lorsque différents classeurs qui doivent être spécialisés ensemble ne peuvent être évalués que dans des situations très éloignées de l'environnement[12]. Pour spécialiser, le système doit alors attendre que chacun de ces classeurs ait été suffisamment évalué. Ce problème devient critique dans le cas où la topologie de l'environnement n'est pas *ouverte*. Nous entendons par là que les situations définissant le problème peuvent être regroupées en sous-graphes. Ces sous graphes sont ici tels que les situations sont fortement connectées les unes avec les autres par des actions au sein du sous graphe, mais ces sous-graphes sont faiblement connectés entre eux. Pour ce type de topologie dont un exemple est donné par ma figure 4, si l'agent explore son environnement de façon aléatoire, il risque de devoir effectuer un grand nombre d'actions avant de recueillir l'information nécessaire à l'évaluation des différents classeurs impliqués dans une spécialisation.

5.3. *Généralisation des conditions*

Le mécanisme de spécialisation de MACS peut donc produire des classeurs avec une partie `Condition` à un niveau sous-optimal de généralisation, particulièrement dans les cas d'*exploration locale*, quand l'agent n'a visité qu'une partie de l'environnement. Ainsi MACS a besoin d'un mécanisme de généralisation qui soit capable de

12. Les situations sont *éloignées* dans le sens où beaucoup d'actions successives sont nécessaires pour aller d'une situation à l'autre.

revenir sur des spécialisations sous-optimales. Comme la spécialisation, la généralisation utilise des évaluations et des heuristiques afin d'exploiter au mieux l'expérience pour guider le mécanisme.

5.3.1. *Les estimations utilisées par MACS pour la généralisation*

Afin de calculer les estimations i_g, MACS contrôle à chaque pas de temps les classeurs dont la partie `Action` est égale à A_{t-1} et dont la partie `Condition` ne correspond *pas* à S_{t-1}.

Etant donné de tels classeurs, pour chaque attribut spécialisé de la partie `Condition`, MACS examine si la partie `Condition` du classeur deviendrait compatible avec S_{t-1} si l'attribut considéré était un symbole "`#`". Dans ce cas, l'estimation i_g considérée est mise à jour :

– si la partie `Effet` du classeur correspond à S_t, alors un classeur avec une partie `Condition` plus générale aurait bien anticipé tout de même et l'estimation est incrémentée.

– si la partie `Effet` du classeur ne correspond pas à S_t, alors un classeur avec une partie `Condition` plus générale aurait mal anticipé et l'estimation est décrémentée.

Les estimations sont incrémentées et décrémentées selon une règle de Widrow-Hoff. Les valeurs initiales sont 0.5. Un attribut général prend une valeur par défaut de 0.5.

Avec ce mécanisme, MACS est capable de décider si un attribut d'une partie `Condition` peut être généralisé ou pas.

5.3.2. *Le mécanisme de généralisation de MACS*

Le mécanisme de généralisation considère des ensembles de classeurs ayant les mêmes parties `Action` et `Effet`. Avec un tel ensemble de classeurs, MACS en crée un nouveau dont les éléments sont plus généraux ou identiques aux initiaux. Les parties `Condition` de ces nouveaux classeurs sont telles qu'elles ne provoquent pas de conflits avec celles d'autres classeurs partageant la même partie `Action` mais qui ont une partie `Effet` incompatible.

Si chaque évaluation i_g d'un classeur est inférieure à 0.5, ce n'est pas un bon candidat pour la généralisation et on l'ajoute sans modification dans le nouvel ensemble de classeurs. Dans le cas contraire, un nouveau classeur est créé. Un attribut de la partie `Condition` est généralisé si l'estimation i_g qui lui est associée est la plus grande parmi les autres estimations i_g du classeur en question.

Il peut se trouver que la nouvelle partie `Condition` soit en conflit avec celle d'autres classeurs ayant la même partie `Action` mais une partie `Effet` incompatible. Deux parties `Effet` sont dites incompatibles s'il y a au moins un attribut pour lequel les symboles sont différents de "`=`" et différents l'un de l'autre. Deux parties `Condition` sont en conflit si elles sont toutes les deux compatibles avec au moins une

des situation possibles. MACS trouve les situations possibles dans l'ensemble `P` de toutes les situations perçues pendant les évolutions de l'agent (voir la section 3).

En cas de conflit, MACS n'ajoute pas le nouveau classeur général au nouvel ensemble, mais conserve le classeur initial.

Jusqu'ici, MACS crée un nouvel ensemble de classeurs tels que chacun d'eux est soit plus général, soit identique au classeur initial, et tels qu'aucun d'eux n'est en conflit avec d'autres classeurs du système. Reste à choisir lesquels de ces nouveaux classeurs sont conservés. Pour ce faire, MACS contrôle itérativement chaque paire de classeurs possible. Quand la partie `Condition` d'un classeur est plus générale que celle de l'autre, il est conservé et l'autre est retiré.

Ce processus permet de remplacer plusieurs classeurs par d'autres classeurs en nombre inférieur. La disjonction des parties `Condition` des nouveaux classeurs est compatible avec les mêmes situations que celles couvertes par les anciens classeurs. Ainsi, elles ne sont pas en conflit avec d'autres classeurs du système.

5.4. *Prise en compte par MACS de transitions nouvelles*

Pour être complet, le modèle doit prendre en compte chaque transition. Pourtant, il peut être incomplet pour les raisons suivantes :

– le système est initialisé avec une liste de classeurs vide ;

– le mécanisme de *sélection des classeurs fiables* peut retirer des classeurs simplement oscillants en cas d'exploration locale ;

– le mécanisme ajoute un classeur spécialisé seulement s'il correspond à au moins une situation déjà perçue. Aussi longtemps que l'agent n'a pas perçu chacune des situations possibles, certains classeurs qui seraient fiables si tout avait été exploré ne peuvent pas être ajoutés.

Ainsi, à chaque pas de temps, si nécessaire, le système prend en compte la transition découlant de S_{t-1}, A_{t-1} et S_t. Pour chaque attribut de S_t, il crée un effet tel que le seul attribut spécialisé (qui n'est pas un symbole " `?`") prend la valeur correspondante dans S_t. Etant donné cet effet, parmi les classeurs dotés d'une partie `Action` correspondant à A_{t-1}, le système vérifie qu'il existe au moins un classeur tel que sa partie `Condition` soit compatible avec S_{t-1} et tel que sa partie `Effet` égale l'effet calculé. Si ce n'est pas le cas, MACS prend en compte la transition en ajoutant dans le système un nouveau classeur.

La partie `Action` de ce nouveau classeur est égale à A_{t-1} et sa partie `Effet` est égale à l'effet considéré. La partie `Condition` est telle qu'elle est aussi générale que possible en étant compatible avec S_{t-1} sans pour autant être compatible avec la partie `Condition` d'aucun autre classeur avec les mêmes parties `Effet` et `Action`.

6. Etude expérimentale

Jusqu'ici, nous avons décrit MACS, un nouveau système de classeurs qui utilise un type de généralisation différent de ceux de XCS, ACS ou YACS. Parmi les systèmes de classeurs capables d'anticipation, dans [GER 01a] et surtout [GER 01b], nous avons déjà comparé les capacités de ACS et de YACS à apprendre un modèle de la dynamique entre l'agent et son environnement [GER 01b]. Nous avons montré à cette occasion que, bien qu'utilisant le même formalisme, YACS apprenait plus vite qu'ACS tout en généralisant mieux.

Dans cette partie, nous comparons les capacités de YACS et MACS à extraire des modèles compacts des environnements Maze228, Maze252, Maze288 et Maze324, décrits dans la section 6.1. Les résultats sont donnés dans la section 6.2 et discutés dans la section 6.3.

6.1. *Les environnements Maze228, Maze252, Maze288 et Maze324*

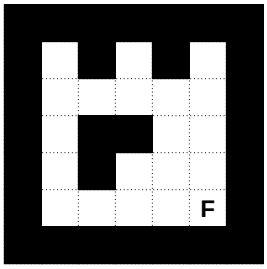

Figure 2. *L'environnent Maze228*

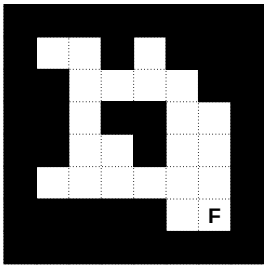

Figure 3. *L'environnement Maze252*

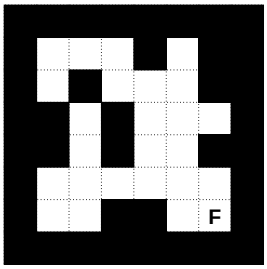

Figure 4. *L'environnent Maze288*

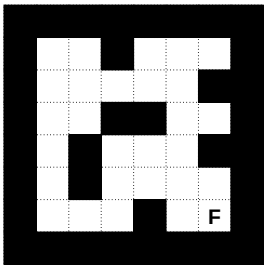

Figure 5. *L'environnement Maze324*

Comme les problèmes de "*Wilson woods*" souvent utilisés comme références et sujets d'étude dans les systèmes de classeurs, Maze228, Maze252, Maze288 et Maze324 sont des mondes de cases. De tels environnements permettent une représentation intuitive d'automates à états finis dans lesquels les états (ou situations) sont composés d'attributs et dans lesquels les transitions offrent quelques régularités. Cette plus grande

facilité à étudier un labyrinthe qu'un automate à états finis quelconque est illustrée par Lanzi dans [LAN 00b].

Chaque case de la grille peut être vide, contenir un obstacle ■ ou de la nourriture F. L'agent est situé sur une case et il est orienté dans une des quatre directions cardinales. Il perçoit sa situation comme la conjonction de neuf attributs, huit correspondant aux cases directement autour de lui et un concernant la case sur laquelle il se trouve. Chaque attribut peut prendre trois valeurs : 0 (case vide), 1 (obstacle) ou 2 (nourriture). Les attributs sont ordonnés dans le sens des aiguilles d'une montre, en commençant face à l'agent. Celui-ci a trois actions à sa disposition : tourner d'un quart de tour sur sa droite ou sur sa gauche, ou avancer d'une case vers l'avant. Dans ce cas, si la case devant lui contient un obstacle, l'agent le heurte et reste dans la même case.

La topologie des environnements étudiés est illustrée par les figures 2 3, 4 et 5. Quantitativement, Maze228 contient 19 cases vides à partir desquelles il est possible d'agir (toutes les cases vides). L'agent peut être situé dans chacune de ces cases selon quatre directions et, dans chacun de ces cas, il a l'opportunité de choisir entre trois actions. En conséquence, il y a $19 \times 4 \times 3 = 228$ transitions à modéliser pour Maze228. Comme Maze252 présente 21 cases non terminales, cet environnement offre 252 transitions. De même, Maze288 et Maze324 présentent respectivement 25 et 28 cases vides, donc 288 et 324 transitions possibles.

D'un point de vue qualitatif, la particularité topologique de Maze288 par rapport à Maze228, Maze252 et Maze324, est qu'il est moins *ouvert* (voir 5.2.2). Sa partie gauche est constituée de deux zones, chacune accessible d'une seule façon. Par contre, dans les autres environnements, il est plus difficile de circuler d'une zone à l'autre. Cette particularité de Maze288 conduit un agent guidé de manière aléatoire à parcourir moins régulièrement l'ensemble des situations possibles.

Les expériences sont divisées en *essais*. L'agent commence un essai dans une case vide choisie au hasard, avec une orientation aléatoire. L'essai est terminé lorsque l'agent atteint la case avec de la nourriture. Celle-ci ne disparaît pas, l'agent reçoit une récompense de 1, perçoit la nouvelle situation et un nouvel essai commence. Pour comparer différents apprentissages latents, l'exploration purement aléatoire présente l'avantage d'offrir aux différents systèmes une qualité d'information du même ordre.

6.2. *Résultats expérimentaux*

Pour rendre compte de l'évolution de la fiabilité et de la complétude du modèle au cours du temps, nous utilisons une mesure du *pourcentage de connaissance* fourni par le modèle. Pour chaque transition possible dans l'environnement, nous vérifions si le système de classeurs la modélise correctement, c'est-à-dire s'il anticipe une seule situation, celle qui est effectivement observée. Le pourcentage de connaissance est le taux de transitions bien modélisées par le système. Cette mesure ne peut pas être effectuée par le système lui-même puisqu'elle suppose une connaissance parfaite de

l'environnement. Elle n'est possible que dans la mesure où les environnements étudiés sont des environnements simulés.

Comme le présent article est centré sur l'*apprentissage latent*, l'agent choisit ses actions de manière aléatoire. Tous les résultats présentés sont des moyennes sur 100 expériences. Pour YACS, comme dans [GER 01a], la taille de la mémoire m est égale à 5 et les taux d'apprentissage[13] sont de 0.1. Pour MACS, nous avons également utilisé des taux d'apprentissage de 0.1. Les paramètres e_o, e_r et e_a sont tous égaux à 5 (voir la section 5).

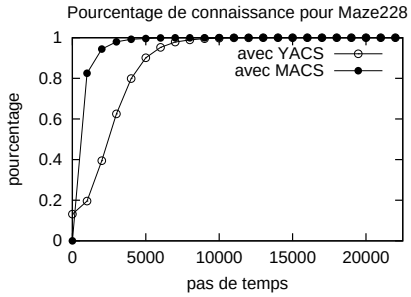

Figure 6. *Evolution du pourcentage de connaissances dans Maze228*

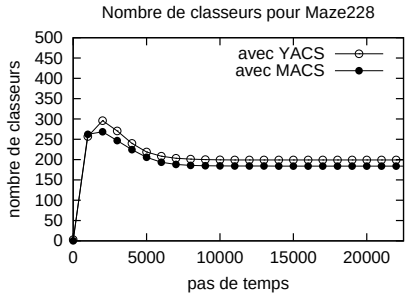

Figure 7. *Evolution du nombre de classeurs dans Maze228*

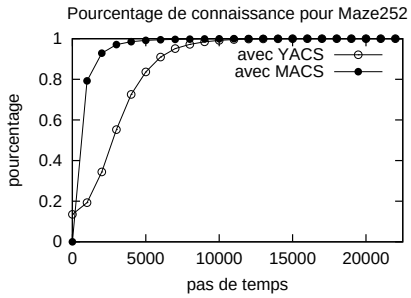

Figure 8. *Evolution du pourcentage de connaissances dans Maze252*

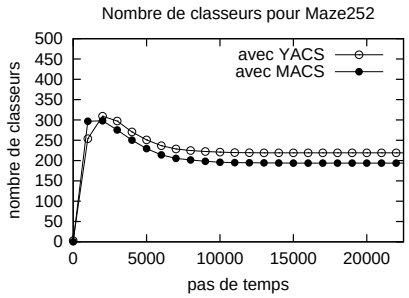

Figure 9. *Evolution du nombre de classeurs dans Maze252*

Les figures 6, 8, 10 et 12 présentent l'évolution moyenne sur 100 expériences du pourcentage de connaissance quand YACS et MACS interagissent l'un et l'autre avec

13. Nous appelons "*taux d'apprentissage*" les facteurs β utilisés dans les règles de Widrow-Hoff pour permettre la mise à jour des différentes estimations utilisées pour la spécialisation et la généralisation.

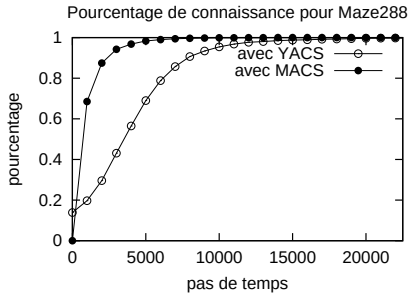

Figure 10. *Evolution du pourcentage de connaissances dans Maze288*

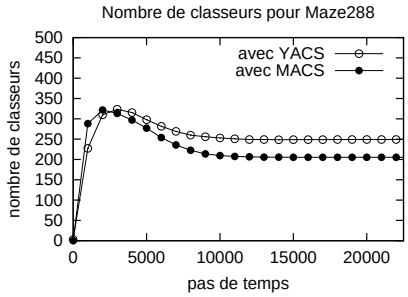

Figure 11. *Evolution du nombre de classeurs dans Maze288*

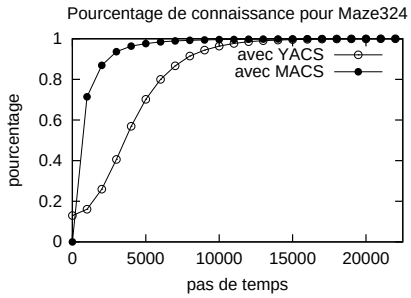

Figure 12. *Evolution du pourcentage de connaissances dans Maze324*

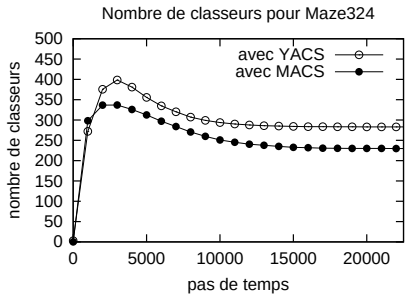

Figure 13. *Evolution du nombre de classeurs dans Maze324*

Maze228, Maze252, Maze288 et Maze324. Les deux systèmes convergent toujours vers un modèle complet et fiable des transitions.

Les figures 7, 9, 11 et 13 présentent l'évolution moyenne du nombre de classeurs quand YACS et MACS interagissent l'un et l'autre avec Maze228, Maze252, Maze288 et Maze324.

Le tableau 3 résume les résultats expérimentaux obtenus au cours des différentes expériences. Ils présentent la moyenne sur les 100 expériences (et la variance associée) du temps mis par YACS et MACS pour atteindre une connaissance parfaite de l'environnement. Ils montrent aussi la moyenne (et la variance associée) du nombre de classeurs nécessaires pour modéliser les interactions entre le système et son environnement.

Les figures 14 et 15 reprennent ces résultats et présentent les relations entre la taille de l'environnement d'une part, et le nombre de classeurs et le temps de convergence vers un modèle complet et fiable d'autre part.

	Tps. Convergence		Nb. Classeurs	
	Moyenne	Variance	Moyenne	Variance
Maze228 - YACS	9295	3770	199	0
Maze228 - MACS	4960	1764	184.8	5.39
Maze252 - YACS	11466	4096	219	0
Maze252 - MACS	6695	2413	193.8	6.40
Maze288 - YACS	20983	9679	249	0
Maze288 - MACS	8472	3594	205.3	6.96
Maze324 - YACS	15740	4385	283	0
Maze324 - MACS	11398	5197	229.6	6.91

Tableau 3. *Synthèse des résultats expérimentaux*

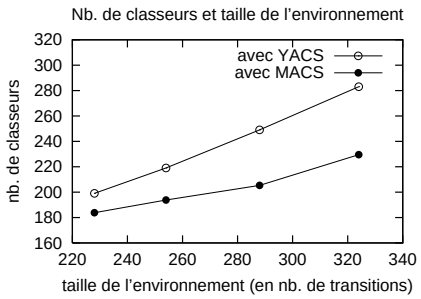

Figure 14. *Nombre de classeurs obtenus en fonction de la taille de l'environnement*

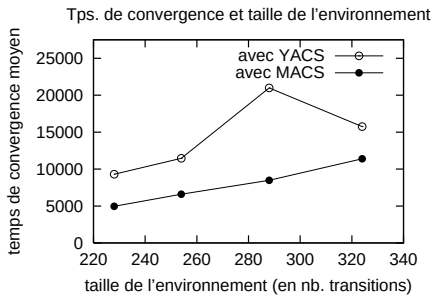

Figure 15. *Temps de convergence en fonction de la taille de l'environnement*

6.3. *Discussion*

La sémantique des classeurs est différente dans YACS et dans MACS. Avec le formalisme de YACS, les classeurs modélisent les transitions comme un tout alors que, dans le formalisme de MACS, chaque classeur anticipe la valeur d'un seul attribut. Dans les environnements avec peu de régularités impliquant des attributs différents, le nombre de règles découvertes par MACS devrait être plus important.

De plus, le formalisme de MACS n'utilise pas de symbole "=". En conséquence, les régularités comme "*se déplacer vers un mur ne provoque aucun changement dans la situation*" sont représentées avec plus de classeurs dans MACS que dans YACS : un pour chaque valeur de chaque attribut en utilisant la capacité de MACS à représenter des régularités impliquant différents attributs. Toutefois, le nombre de classeurs de ce type demeure identique quelle que soit la taille de l'environnement. Malgré cela, dans Maze228 comme dans Maze288, MACS converge vers un plus petit nombre de

classeurs que YACS grâce aux nouvelles régularités exprimables dans son formalisme. Ceci illustre le fait que le formalisme de MACS est plus puissant que le précédent. En effet, le rapport entre le nombre de classeurs découverts par MACS et par YACS est de plus en plus à l'avantage de MACS à mesure que l'environnement grandit en termes de nombre de transitions possibles. MACS montre ainsi sa capacité à représenter ces régularités générales de la structure de chaque environnement du même type :

– dans MACS, chaque transition impliquant une action de rotation est modélisée simplement avec le même nombre de classeurs, indépendamment du nombre de cases de l'environnement. Puisque YACS ne peut pas représenter les régularités impliquant plusieurs attributs, il a besoin de plus de classeurs pour modéliser ces transitions lorsque la taille de l'environnement augmente ;

– les seuls attributs qu'il est difficile d'anticiper pour MACS correspondent aux trois cases en face de l'agent, quand il avance et qu'aucun mur ne se trouve juste en face de lui. L'information à modéliser est alors dépendante de la topologie particulière de chaque labyrinthe plutôt que de la structure générale des labyrinthes. Afin de diminuer le nombre de classeurs nécessaires pour anticiper dans ces cas précis, une solution pourrait consister à pourvoir MACS d'un mécanisme qui lui permette de créer des parties `Effet` avec plusieurs symboles spécialisés, comme nous l'avions initialement proposé dans la section 4.

Les régularités du premier type sont découvertes très rapidement par MACS. Ainsi, durant les 1 000 premiers pas de temps des expériences (pour chacun des environnements testés), le pourcentage de connaissance croît très rapidement avant de ralentir et le modèle complet est appris plus rapidement dans MACS que dans YACS.

La rapidité d'apprentissage des actions de rotation tient au fait que, malgré une exploration aléatoire, les exemples pertinents abondent puisque les régularités concernées sont indépendantes de la topologie particulière de chaque environnement. Ceci est très différent en ce qui concerne les anticipations liées à la topologie particulière de chaque environnement. A mesure que les classeurs sont plus spécialisés, les exemples pertinents se font plus rares. Ainsi, en raison de l'exploration aléatoire, le système expérimente beaucoup de transitions ne lui apportant aucune information supplémentaire, et la rapidité de l'apprentissage s'en trouve diminuée.

Malgré cette exploration aléatoire, la vitesse de l'apprentissage par MACS d'un modèle complet semble linéaire par rapport à la taille des environnements testés. Pour YACS, par contre, la topologie particulière de Maze288 pose problème en exploration aléatoire (voir 6.1). La figure 15 montre que MACS ne souffre pas du problème évoqué à la section 5.2.2. Ainsi, la spécialisation immédiate des classeurs qui *oscillent* permet à MACS d'être moins gêné que YACS par la topologie de Maze288 qui est telle que certains sous-graphes situations/actions sont faiblement connectés aux autres sous-graphes.

Afin d'accélérer la convergence de MACS, nous envisageons d'utiliser des algorithmes de programmation dynamique afin de guider l'exploration de l'environnement. Cette *exploration active* se fonde sur les transitions déjà modélisées avec exac-

titude et sur une récompense interne dépendant du gain en information attendu dans chaque situation. Ce gain est fonction de la quantité d'évaluations de chaque classeur dont la partie `Condition` est compatible avec la situation considérée. Avec un tel mécanisme, l'agent peut utiliser un modèle partiel pour accélérer l'apprentissage du modèle complet, en explorant activement son environnement grâce à la rétropropagation du gain en information.

7. Conclusion

Dans cet article, nous avons décrit plusieurs systèmes de classeurs, chacun d'eux conduisant à une interprétation différente de la notion de généralisation. La propriété de généralisation est centrale dans les systèmes de classeurs puisqu'elle permet d'élaborer des représentations plus compactes que ne le fait l'apprentissage par renforcement classique. En particulier, nous avons montré comment des systèmes comme XCS considèrent la généralisation en terme de récompense attendue alors que des systèmes comme ACS ou YACS la considèrent en terme d'effets attendus au sein de nouvelles situations. Nous avons aussi mis en lumière quelques limitations du formalisme d'ACS et YACS. Pour surmonter ces limitations, nous avons proposé MACS, un nouveau système qui emploie un formalisme capable d'exprimer davantage de généralisations afin d'apprendre le modèle de la dynamique de l'environnement le plus compact possible.

Par une étude expérimentale, nous avons montré que le nouveau formalisme permet effectivement une généralisation plus puissante que YACS, sans perdre en rapidité d'apprentissage. Pour permettre à MACS d'apprendre plus vite, nous projetons de définir des stratégies d'exploration. Ces stratégies d'exploration utiliseront des Processus de Décision Markoviens fondés sur des modèles incomplets et partiels de l'environnement afin de rechercher activement l'information appropriée, et présenter au système d'apprentissage des exemples plus pertinents que ne le fait l'exploration aléatoire.

Cependant, YACS et MACS sont encore limités au traitement de problèmes markoviens et déterministes. Pour le dernier point, nous pensons que l'absence d'un mécanisme d'ajout de parties `Effet` dans MACS devrait faciliter la prise en compte d'environnements stochastiques. Dans l'immédiat, nous envisageons de développer un système de classeurs modulaires qui utilise sa capacité d'anticipation pour résoudre des problèmes d'ambiguïté perceptuelle tels qu'ils apparaissent dans les environnements non markoviens.

8. Bibliographie

[BEL 57] BELLMAN R. E., *Dynamic Programming*, Princeton University Press, Princeton NJ, 1957.

[BER 01] BERNARDÓ E., X. L., M. G. J., « XCS and GALE : a comparative study of two Learning Classifer Systems with six other learning algorithms on classification tasks »,

LANZI P. L., STOLZMANN W., WILSON S. W., Eds., *Proceedings of the fourth international workshop on Learning Classifer Systems*, 2001.

[BUT 00] BUTZ M. V., GOLDBERG D. E., STOLZMANN W., « Introducing a Genetic Generalization Pressure to the Anticipatory Classifier System Part I : Theoretical Approach », *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, 2000.

[DOR 94] DORIGO M., « Genetic and Non-Genetic Operators in ALECSYS », *Evolutionary Computation*, vol. 1, n° 2, 1994, p. 151-164.

[DRE 91] DRESCHER G. L., *Made-Up Minds : A constructivist approach to Artificial Intelligence*, MIT Press, Cambridge MA, 1991.

[DZE 01] DZEROSKI S., DE RAEDT L., DRIESSENS K., « Relational Reinforcement Learning », *Machine Learning*, vol. 43, n° 1-2, 2001, p. 7-53.

[GER 01a] GERARD P., SIGAUD O., « Adding a Generalization Mechanism to YACS », SPECTOR L., GOODMAN E., WU A., LANGDON W. B., M. V. H., GEN M., SEN S., DORIGO M., PEZESHK S., GARZON M., BURKE E., Eds., *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO01)*, San Francisco, CA, july 2001, Morgan Kaufmann, p. 951-957.

[GER 01b] GERARD P., STOLZMANN W., SIGAUD O., « YACS : a new Learning Classifier System using Anticipation », *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, , 2001, to appear.

[GOL 89] GOLDBERG D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley, 1989.

[HOF 93] HOFFMANN J., *Vorhersage und Erkenntnis [Anticipation and Cognition]*, Hogrefe, 1993.

[HOL 76] HOLLAND J. H., « Adaptation », *Progress in theorical biology*, , 1976.

[HOL 78] HOLLAND J. H., REITMAN J. S., « Cognitive Systems based on adaptive algorithms », *Pattern Directed Inference Systems*, vol. 7, n° 2, 1978, p. 125-149.

[HOL 85] HOLLAND J. H., « Properties of the bucket brigade algorithm », GREFENSTETTE J. J., Ed., *Proceedings of the 1st international Conference on Genetic Algorithms and their applications (ICGA85)*, L.E. Associates, july 1985, p. 1-7.

[HOL 90] HOLLAND J. H., « Concerning the emergence of tag mediated lookahead in Classifier Systems », *Special Issue of Physica D*, vol. 42, 1990, p. 188-201.

[KOV 97] KOVACS T., « XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. Technical Report Version. », rapport, 1997, chool of Computer Science, University of Birmingham, Birmingham, U.K.

[LAN 00a] LANZI P. L., « Learning Classifier Systems from a Reinforcement Learning Perspective », rapport, 2000, Dip. di Elettronica e Informazione, Politecnico di Milano.

[LAN 00b] LANZI P. L., « Toward Optimal Performance in Classifier Systems », *Evolutionary Computation Journal*, vol. 8, n° 4, 2000, p. 393-418, MIT Press.

[QUI 93] QUINLAN R., *C4.5 : Programs for Machine Learning*, Morgan Kaufmann Publisher, 1993.

[RIO 91] RIOLO R. L., « Lookahead planning and latent learning in a Classifier System. », MEYER J.-A., WILSON S. W., Eds., *Proceedings of the First International Conference on Simulation of Adaptative Behavior*, MIT Press, 1991, p. 316-326.

[SEW 49] SEWARD J. P., « An Experimental Analysis of Latent Learning », *Journal of Experimental Psychology*, , 1949.

[SIG 01] SIGAUD O., GERARD P., « Being Reactive by Exchanging Roles : an Empirical Study », HANNEBAUER M., WENDLER J., PAGELLO E., Eds., *LNAI 2103 : Balancing reactivity and Social Deliberation in Multiagent Systems*, Springer-Verlag, 2001.

[STO 98] STOLZMANN W., « Anticipatory Classifier Systems », KOZA J. R., BANZHAF W., CHELLAPILLA K., DEB K., DORIGO M., FOGEL D. B., GARZON M. H., GOLDBERG D. E., IBA H., RIOLO R., Eds., *Genetic Programming*, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[SUT 91] SUTTON R., « Reinforcement Learning Architectures for Animats », MEYER J. A., WILSON S. W., Eds., *From animals to animats : Proceedings of the First International Conference on Simulation of Adaptative Behavior*, Cambridge, MA, 1991, MIT Press.

[SUT 98] SUTTON R. S., BARTO A. G., *Reinforcement Learning : An Introduction*, MIT Press, 1998.

[TOL 32] TOLMAN E. C., *Purposive behavior in animals and men*, Appletown, New York, 1932.

[TOM 00] TOMLINSON A., BULL L., « CXCS », LANZI P. L., STOLZMANN W., WILSON S. W., Eds., *Learning Classifier Systems : from Foundations to Applications*, p. 194-208, Springer-Verlag, Heidelberg, 2000.

[WAT 89] WATKINS C. J., « Learning with delayed rewards », PhD thesis, Psychology Department, University of Cambridge, England, 1989.

[WIL 89] WILSON S. W., « A Critical Review of Classifier Systems », *Proceedings of the Third International Conference on Genetic Algorithms*, Los Altos, California, 1989, Morgan Kaufmann, p. 244-255.

[WIL 94] WILSON S. W., « ZCS, a zeroth level Classifier System », *Evolutionary Computation*, vol. 2, n° 1, 1994, p. 1-18.

[WIL 95] WILSON S. W., « Classifier fitness based on accuracy », *Evolutionary Computation*, vol. 3, n° 2, 1995, p. 149-175.

[WIL 01] WILSON S. W., « Mining oblique data with XCS », STOLZMANN W., LANZI P. L., WILSON S. W., Eds., *LNAI 1996 : Advances in Classifier Systems*, Springer Verlag, 2001.