Les systèmes de classeurs

Une présentation générale

Cédric Buche — Cyril Septseault — Pierre De Loor

Laboratoire d'Informatique des Systèmes Complexes (LISyC)
Centre Européen de Réalité Virtuelle (CERV)
BP 38 F-29280 Plouzané
{buche, septseault, deloor}@enib.fr

RÉSUMÉ. *Cet article propose une présentation des systèmes de classeurs qui sont des outils destinés à l'apprentissage d'interactions. Il en existe de nombreuses variantes qui sont présentées par le biais d'une approche chronologique introduisant les nouveaux défis ou concepts abordés par chaque modèle. Nous décrivons les concepts de base que sont les algorithmes génétiques et l'apprentissage par renforcement. L'article présente ensuite les systèmes de base tels que le ZCS ou le XCS, les systèmes à anticipation, ainsi que les classeurs hiérarchiques ou hétérogènes.*

ABSTRACT. *This article concerns the presentation of classifiers systems built to learn interactions. Different kinds of classifiers systems are presented, using a chronological approach introducing new challenges or concepts approached by every model. We describe used concepts : genetic algorithms and reinforcement learning. The article presents basics systems such as ZCS or XCS, systems for anticipation, as well as hierarchicals or heterogenous classifiers.*

MOTS-CLÉS : *système de classeurs, adaptabilité, apprentissage, dynamicité.*

KEYWORDS: *classifiers system, adaptability, learning, dynamicity.*

RSTI - TSI – 25/2006. Evolution artificielle, pages 963 à 990

1. Introduction

Les systèmes de classeurs sont des architectures de contrôle et d'apprentissage non supervisé d'interactions entre un système artificiel et son environnement. Ils offrent un apprentissage inductif qui généralise un ensemble de règles comportementales en combinant l'apprentissage par renforcement (Sutton, 1992) et les algorithmes génétiques (Holland, 1975). Plus précisément, à partir d'un retour de l'environnement, ils permettent de définir des règles et de les pondérer. Les conditions de ces règles consistuent un filtre perceptif généralisant. Ils peuvent alors faire de la classification (*one step problems*) ou de l'apprentissage comportemental (*multi-step problems*). Ce dernier cas caractérise des apprentissages de successions d'interactions.

En classification, les systèmes de classeurs ont été utilisés pour effectuer des diagnostics médicaux (Bonelli *et al.*, 1990). En économie, ils ont permis d'étudier des scénarios de négociation (achat/vente d'actions) et d'apprendre des stratégies économiques (Schulenburg *et al.*, 2001). Mais c'est dans la problématique de l'apprentissage de comportements en situation autonome qu'ils sont les plus pertinents. Dans (Girard *et al.*, 2001) ils constituent le mécanisme décisionnel de vaisseaux spatiaux. Sanza les utilise pour contrôler la sélection d'actions d'entités virtuelles coopératives appliquées dans un jeu de football virtuel où chacun des 22 acteurs possède un système de classeurs (Sanza, 2001). Enfin, Robert les implique dans la sélection d'action de personnages de jeu en ligne persistant et massivement multi joueurs (Robert *et al.*, 2002). Cette liste d'exemples, non exhaustive, permet d'évaluer le large champ d'application de ces systèmes. Elle explique en partie les raisons pour lesquelles il existe une multitude de modèles et d'algorithmes au milieu desquels il n'est pas aisé de se retrouver. Cet article propose une vue d'ensemble au travers d'une présentation progressive identifiant les différents problèmes rencontrés par les différentes versions de systèmes de classeurs. L'article est articulé comme suit.

La section 2 présente les principes utilisés dans les systèmes de classeurs. Elle fait un rappel sur les algorithmes génétiques et l'apprentissage par renforcement. Ensuite, la section 3 présente une formalisation des systèmes de classeurs. La section 4 montre différentes versions de systèmes de classeurs : le ZCS, le XCS, les systèmes à anticipation et les systèmes hiérarchiques et hétérogènes. Enfin, la section 5 dresse un bilan.

2. Principes de base

Cette section a pour objet de synthétiser les principes de base sur lesquels reposent les systèmes de classeurs. Cet exercice délicat ne prétend pas à l'exhaustivité, mais à l'introduction des concepts amonts aux développement des systèmes de classeurs que sont les algorithmes génétiques (section 2.1) et l'apprentissage par renforcement (section 2.2). Le lecteur rompu à ces notions pourra sans regret passer à la section 3.

2.1. *Les algorithmes génétiques*

Les algorithmes génétiques furent introduits par (Holland, 1975). L'auteur se base sur l'observation de l'évolution des organismes vivants complexes comme étant le résultat de deux processus : la sélection naturelle et la reproduction. A partir de ce mécanisme évolutionniste, Holland eut l'idée de définir des algorithmes de recherche stochastiques. Les solutions potentielles d'un problème sont choisies aléatoirement. Ces solutions sont alors des individus qui forment la population initiale. L'algorithme fait évoluer cette population en sélectionnant les meilleurs individus et en les faisant se reproduire. Cette opération de croisement est une fonction permettant de générer un individu « enfant », en combinant des informations présentées par des individus « parents ». Les progénitures ainsi générées vont contenir, combiner et optimiser les facteurs qui contribuaient à la sélection de leurs parents. Enfin, pour assurer une scrutation de l'espace des solutions, on introduit, avec une faible probabilité, des erreurs sur les progénitures générées, c'est l'opération de mutation. L'algorithme de base est représenté figure 1.

```
Initialisation de la population
Évaluation de la population
Faire
   Sélectionner les individus parents (éliminer les plus mauvais)
   Créer de nouveaux individus (progéniture) par croisement des parents
   Opérer un opérateur de mutation
   Évaluer les progénitures
   Remplacer la population par les progénitures
Jusqu'à la condition de terminaison
```

Figure 1. *Principe général des algorithmes génétiques*

Prenons l'exemple d'un chemin dont il faut optimiser la longueur pour rejoindre un point B à partir d'un point A (voir figure 2). Définissons les individus comme une succession de déplacements (H : haut, B : bas, D : droite et G : gauche). Deux individus potentiels sont représentés sur la figure, leur qualité est inversement proportionnelle à la distance qu'il reste à parcourir pour atteindre le point B. Leur croisement est effectué en combinant la partie droite de chacun d'eux avec la partie gauche de l'autre (il s'agit d'un exemple, de nombreuses définitions de croisement sont possibles). Il donne naissance à deux individus dont l'un est plus performant. De plus, une mutation sur son quatrième élément, le rend encore meilleur.

Les algorithmes génétiques reposent sur les hypothèses qu'il est possible d'évaluer la qualité d'une solution (fonction de fitness) et que la combinaison de deux solutions jugées acceptables peut déboucher sur une solution meilleure. Ces hypothèses, pour être respectées, demandent une définition appropriée au problème de la fonction de fitness et des opérations de sélection et de reproduction. Ceci a donné lieu à de nombreux développements (Jong, 1975; Smith *et al.*, 1998; Goldberg, 1989) et en particulier à la théorie des algorithmes génétiques qui est une justification mathématique de ces

derniers et qui dépasse le cadre de cet article. Les travaux autour des algorithmes génétiques sont toujours d'actualité, en témoignent les conférences annuelles GECCO (*Genetic and Evolutionary Computation Conference*).

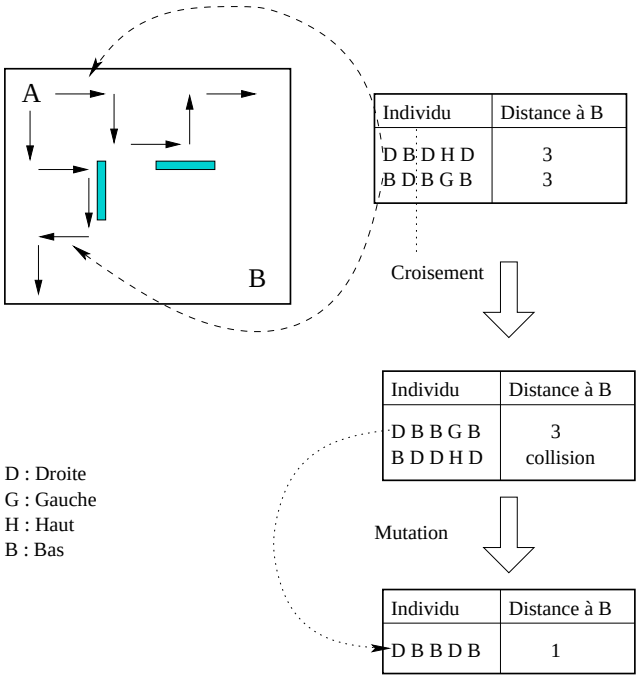


Figure 2. *Exemple d'algorithme génétique : optimiser un chemin*

2.2. *L'apprentissage par renforcement*

L'apprentissage par renforcement (Sutton, 1991) est élaboré par essai-erreur et se positionne parmi les apprentissages dits « non supervisés », c'est-à-dire qui ne nécessitent ni une série d'exemples à apprendre ni une élaboration d'une fonction de *fitness* comme c'est le cas avec les algorithmes génétiques. La notion de renforcement a été introduite par les psychologues dans des expériences d'apprentissage animal (Skinner, 1938). Ces expériences, poursuivant les travaux de (Pavlov, 1927) sur le conditionnement, montraient que si dans une situation donnée, un animal recevait une récompense ou une punition dépendant de l'action qu'il entreprenait, il apprenait progressivement à choisir les actions apportant le maximum de récompenses. Il arrivait donc à associer situation et action par mémorisation et l'amélioration de ses performances était fonction de l'intensité des récompenses et de leur répétitivité. Les psychologues Rescola et Wagner ont proposé une équation relatant ces mécanismes. Tolman nuança l'effet de la récompense en montrant que les animaux mémorisaient

leur environnement de façon latente en l'absence de récompense et pouvaient exploiter cette mémorisation *a posteriori* (Tolman *et al.*, 1930). Les algorithmes d'apprentissage par différence temporelle (Klopf, 1972; Sutton, 1984; Sutton, 1988) unifient les théories de l'apprentissage par renforcement et les résultats de la programmation dynamique (Bellman, 1957; Rust, 1996) [1]. L'algorithme « apprend » l'environnement au fur et à mesure d'une exploration effectuée en exécutant des actions. Après chaque action, l'environnement se retrouve dans un état qui est généralement différent et dépendant de l'action exécutée. L'algorithme considère alors les récompenses fournies par l'environnement à chaque état.

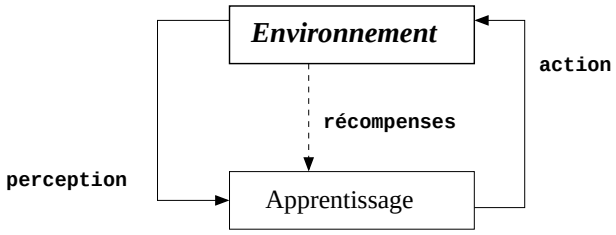


Figure 3. *L'apprentissage par renforcement*

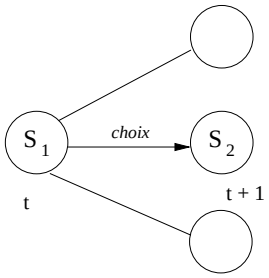
Les deux difficultés rencontrées sont :

1) la mise en place d'une politique d'équilibre entre l'exploration et l'exploitation. En effet, durant l'apprentissage, il faut choisir une action permettant d'apprendre en explorant (prendre une action au hasard pour voir si elle permet d'optimiser les gains) tout en exploitant (considérer que l'apprentissage est terminé et que l'on connaît la meilleure action pour chaque état) ;

2) la prise en compte d'environnements ne fournissant des renforcements que pour certains états. Dans ce cas, il faut optimiser des gains sur un terme plus ou moins long et définir une espérance de gain adéquate. Concrètement une qualité initiale $V(s)$, correspondant à une espérance de gains pour l'état s, est fixée à une valeur quelconque pour chaque état. Cette espérance doit être définie en fonction du problème, elle peut par exemple correspondre à la moyenne des renforcements espérés sur une fenêtre des n prochaines actions. On utilise ensuite un apprentissage par différence temporelle : la qualité d'un état évaluée à un instant *t* va être corrigée par la valeur de celle de l'état suivant atteint à l'instant *t+1*. L'objectif étant d'obtenir un gradient entre les qualités des états permettant d'identifier les chemins maximisant les renforcements (voir figure 4).

En pratique, la qualité est associée non pas à un état mais à un couple *état-action*. L'algorithme le plus connu est le Q-Learning de (Watkins *et al.*, 1992). En Q-learning, la qualité d'un couple *état-action* $Q(s, a)$ est mise à jour en fonction du couple $Q(s', a')$ (ou s' est l'état suivant et a' la meilleure action possible partant de s').

1. La programmation dynamique s'intéresse à l'optimalité d'une solution en fonction des probabilités de gain dans un environnement connu.



$$\boxed{V^{t+1}(S_1) = V'(S_1) + \alpha * (\gamma * V^{t+1}(S_2) - V'(S_1) + r)}$$

$0 < \alpha < 1$: Taux d'apprentissage

$0 < \gamma < 1$: Remise

r : Récompense (éventuelle)

Figure 4. *Apprentissage par différence temporelle*

Le Q-learning possède de fortes similarités avec les systèmes de classeurs que nous aborderons en section 3.

Une des difficultés importante de ce type d'algorithme réside dans le fait qu'ils impliquent une représentation exhaustive de tous les états de l'environnement. Une autre est l'abord d'environnements *non-markovien* consécutif à une perception partielle de ceux-ci. Dans un tel cas, un même état perçu par l'algorithme correspond à deux états effectifs de l'environnement. Même s'ils sont abordés dans certains algorithmes d'apprentissage par renforcement classiques, nous verrons que les systèmes de classeurs apportent des solutions intéressantes à ces problèmes.

3. Principe d'un système de classeurs

Nous allons introduire les principes de base des systèmes de classeurs au travers des premiers modèles, proposés par (Holland, 1976). La première implémentation, appelée *Cognitive System Level One* (CS1), apparaît en 1978 (Holland *et al.*, 1978) : une situation ou « état perçu » est stockée dans une mémoire perceptive de taille limitée, assimilée à la « mémoire à court terme » des sciences cognitives (appelée liste des messages). Une base de règles (« condition-action ») constitue l'équivalent de la mémoire à long terme. A chaque cycle de fonctionnement, un mécanisme d'appariement entre les conditions des règles et les perceptions permet d'identifier la ou les actions pouvant être appliquées (action(s) stockée(s) dans la liste des messages). L'exemple support proposé porte sur l'exploration d'un labyrinthe au sein duquel se trouve de la nourriture qui rapporte des points. L'objectif est d'apprendre en ligne (ou par expérimentation) l'association de conditions et d'actions maximisant la récolte de points. Pour éviter l'explosion combinatoire du nombre de règles possibles, Holland utilise des algorithmes génétiques recherchant des règles généralisantes. Le système devient adaptatif grâce à l'ajout d'un paramètre préférentiel dynamique (souvent appelé force) à chacun des classeurs. Les forces des règles sont alors modifiées dynamiquement par le biais d'un apprentissage par renforcement. Les règles de force importante ont plus de chance d'être choisies par le système, tandis que les règles de faible force sont détruites par l'algorithme génétique.

La figure 5 synthétise les interactions entre un système de classeurs et l'environnement. Nous introduisons ici deux interfaces qui seront utilisées lors de notre formalisation : l'interface d'entrée, qui permet de représenter une perception de l'état de l'environnement, et celle de sortie qui permet d'exécuter l'action choisie et donc d'agir sur l'environnement. Elle montre également les flux d'informations : l'opération d'appariement, l'élection d'action, le renforcement et l'adaptation par algorithmes évolutionnistes.

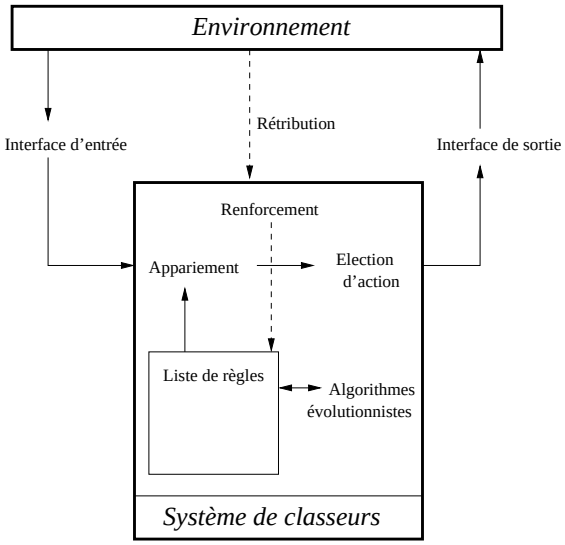


Figure 5. *Interactions entre l'environnement et le système de classeurs*

3.1. *Formalisation*

Dans cette partie, nous allons proposer une formalisation incrémentale et générique des systèmes de classeurs.

3.1.1. *Structure de base*

La structure d'un système de classeurs, présentée en figure 6, est constituée de différents ensembles jouant un rôle dans le choix de l'action à réaliser relative à une situation donnée. Formellement, un système de classeurs est un 7-uplet (`Ie`, `[P]`, `[M]`, `[A]`, `Comparaison`, `Sélection`, `Io`) :

– `Ie` est l'interface d'entrée, faisant correspondre à toute `Perception` de l'environnement un code interprétable par le système (il s'agit le plus souvent d'un code binaire),

– `[P]`, appelé *population*, est l'ensemble des classeurs du système, codés par une succession de n bits, même si certains systèmes travaillent sur d'autres alphabets

(Matteucci, 1999; Wilson, 2000; Heguy *et al.*, 2002). Les représentations généralisantes contiennent des symboles # correspondant à une valeur indéterminée. Une règle est un couple (C, A) avec $C \cup A \in \{0, 1, \#\}^n$ avec :

- C : la condition d'application de la règle,

- A : la ou les actions associées à l'application de la règle.

Prenons l'exemple d'un robot qui possède quatre capteurs tout ou rien, et une action. L'interface d'entrée transforme l'état des capteurs en une valeur binaire, et l'interface de sortie déclenche l'action en fonction de la valeur du bit d'action. Ainsi, une règle $\{011\#, 1\}$ signifie que la règle est applicable si le premier capteur est inactif et les deux suivants actifs. L'état du quatrième capteur n'a pas d'influence, et l'application de la règle déclenche l'action.

– `[M]` $\subseteq$ `[P]` est l'ensemble des classeurs dont la partie condition s'apparie avec les informations perçues de l'environnement pour un cycle de sélection. Il est appelé *Match-set*.

– `[A]` $\subseteq$ `[M]` est l'ensemble des classeurs représentant l'action sélectionnée. Il est appelé *Action-set*.

– `Comparaison` est le mécanisme permettant de passer de `[P]` à `[M]`. Il s'agit généralement d'une règle d'appariement entre `C` et l'information provenant de `Ie`. Cette règle sait interpréter les symboles de généralisation composant les conditions des classeurs.

– `Sélection` est le mécanisme permettant de passer de `[M]` à `[A]`. Il détermine, en fonction de critères spécifiques aux différentes versions de systèmes de classeurs, l'action choisie.

– `Io` est l'interface de sortie faisant correspondre à un code binaire l'activation d'`Action(s)`.

Chaque type de système de classeurs définit ses propres mécanismes de `Comparaison` et de `Sélection`.

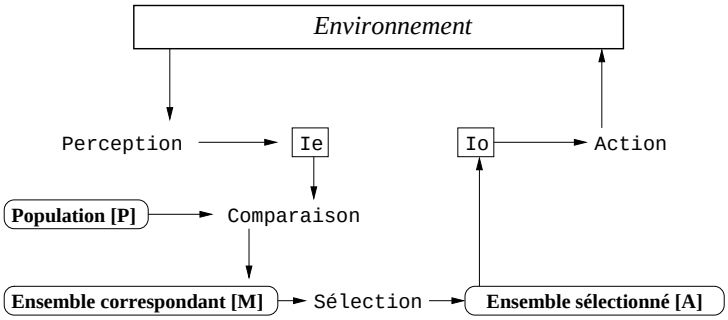


Figure 6. *Structure et dynamique d'un système de classeurs*

3.1.2. *Dynamique d'un système de classeurs*

Les différents mécanismes impliqués dans la dynamique d'un système de classeurs, s'enchaînent cycliquement selon l'ordre `Perception` / `Comparaison` / `Sélection` / `Action`. Ce cycle est répété et peut être soumis à un critère de terminaison (temps maximal de résolution du problème par exemple). Un pseudocode équivalent est défini en figure 7, où la variable *t*, représentant le numéro de l'itération courante, est introduite pour faciliter l'abord de l'apprentissage (section 3.2).

```
t := 0 ;                                    // compteur d'itérations
initClassifierPopulation( P(t) ) ;          // population de classeurs initiale
while not done do                           // critère de terminaison (temps ...)
   t := t +1 ;
   // Perception : 1. codage de la perception
   Ie(t) := readDetectors(t) ;
   // Comparaison : 2. comparaison de [P] et Ie et sauvegarde les appariements dans [M]
   M(t) := matchClassifiers( Ie(t),P(t),Comparaison ) ;
   // Sélection : 3. sélection des règles dans [A]
   A(t) := selectClassifiers( M(t),Sélection ) ;
   // Actions : 4. envoie de l'action via Io
   Io(t) := sendEffectors( A(t) ) ;
```

Figure 7. *Pseudocode représentant les quatre étapes du cycle d'un système de classeurs*

3.2. *Introduction de l'apprentissage*

L'apprentissage par renforcement, présenté en section 2.2, est la base de l'apprentissage associé aux systèmes de classeurs. L'objectif est de favoriser l'enchaînement de classeurs maximisant les renforcements de l'environnement. Des forces sont donc associées aux règles qui sont alors assimilées aux qualités des couples *état-action* d'un Q-learning. Ces forces évoluent grâce à un mécanisme de `Rétribution`.

La généralisation (symbole #) implique qu'un classeur représente une multitude de couples *état-action* (un état possible de l'environnement s'appariera avec plusieurs classeurs). De façon réciproque, plusieurs classeurs vont désigner une action identique. Par conséquent, la chaîne de rétribution n'est plus linéaire comme en apprentissage par différence temporelle, mais arborescente. La technique de distribution des rétributions la plus connue est l'algorithme du *Bucket Brigade*, mais il existe d'autres variantes.

L'explosion des combinaisons implique que tous les classeurs *possibles* ne sont pas représentés. Pour introduire de nouveaux classeurs, la population évolue au fur et à mesure des expériences par des mécanismes dédiés à la `Génération` de classeurs. Ces mécanismes n'existent pas en apprentissage par différence temporelle et peuvent

être élaborés en fonction de différentes hypothèses concernant la notion de qualité d'un classeur.

L'introduction de la Rétribution et de la Génération modifie le cycle générique de fonctionnement des systèmes de classeurs qui est alors celui représenté en figure 8.

```
t := 0 ;                                    // compteur d'itérations
initClassifierPopulation( P(t) ) ;          // population de classeurs initiale
while not done do                           // critère de terminaison (temps ...)
   t := t +1 ;
   // Perception : 1. codage de la perception
   Ie(t) := readDetectors(t) ;
   // Comparaison : 2a. comparaison de [P] et Ie et sauvegarde les appariements dans [M]
   M(t) := matchClassifiers( Ie(t),P(t),Comparaison ) ;
   // Génération (1) Covering : 2b. créer des règles s'appariant avec Ie (si M vide ou criterion1)
   if ( M.size < criterion0 || criterion1 ) then
                              M(t) := cover( Ie(t),P(t),Covering ) ;
   // Sélection : 3. sélection des règles dans [A]
   A(t) := selectClassifiers( M(t),Sélection ) ;
   // Actions : 4. envoie de l'action via Io
   Io(t) := sendEffectors( A(t) ) ;
   // Rétribution (1) : 5. réception de la rétribution de l'environnement
   r := receivePayoff(t) ;
   // Rétribution (2) : 6. distribution de la rétribution aux classeurs
   P(t) := distributeCredit( r,P(t),P(t-1),Rétribution ) ;
   // Génération (2) A.E. : 7. éventuellement (selon t) un Algorithme Évolutionniste est utilisé sur [P]
   if ( criterion2 ) then
         P(t+1) := reviseRules( P(t),Algorithme_Evolutionniste ) ;
```

Figure 8. *Pseudocode représentant les sept étapes du cycle d'un système de classeurs apprenant classique*

3.3. *Principaux paramètres et mécanismes*

3.3.1. *Paramètres des classeurs*

Pour chaque classeur, des paramètres permettant de définir la qualité de la règle sont définis. Ils permettent de préciser les mécanismes de Sélection, de Rétribution et de Génération. On peut, par exemple, enrichir une règle avec une valeur de *Force* f identique à la création de tous les classeurs et évoluant au cours de l'apprentissage. Ainsi $R = (C, A, f)$. La qualité d'une règle influence deux aspects du système :

– la Sélection des classeurs de [M] vers [A], et par conséquent le comportement à court terme du système ;

– l'extraction des classeurs de [P] servant de bases aux algorithmes de Générations (*covering* ou algorithmes génétiques, cf. ci-après) et par conséquent le comportement à long terme du système.

D'autres critères tels que la spécificité (proportionnelle au nombre de 0 ou de 1 que contient le classeur) ou la prédiction de paiement, sont utilisés.

3.3.2. *Mécanisme de Sélection*

Le mécanisme de Sélection permet d'effectuer le passage de l'ensemble de classeurs appariés [M] à l'ensemble [A]. Les classeurs de [M] sont regroupés par paquets, chacun étant constitué de règles dont la partie *Action* s'apparie avec les autres règles du même paquet. Une combinaison des paramètres des classeurs de chaque paquet est utilisée pour définir leur sélectivité. En phase d'exploitation, ce critère est utilisé directement pour sélectionner l'ensemble [A]. En phase d'exploration, un mécanisme de roue de la fortune[2] est généralement appliqué, ce qui signifie que chaque paquet a une probabilité proportionnelle à sa sélectivité d'être sélectionné. Considérons N classeurs, la fonction de sélection est : $p(R) = f/(\sum_1^N f)$. De plus, des mécanismes de Rétribution et de Génération sont appliqués.

3.3.3. *Mécanisme de Rétribution*

La rétribution, appelée *credit assignment*, modifie les forces des classeurs existants dans la population [P]. La répartition de la rétribution de l'environnement favorise ou défavorise les règles selon leur efficacité. Le problème de la rétribution revient à décider quelles sont les règles qui, à un temps t, ont été nécessaires et suffisantes pour atteindre le but à un temps $t + n$, sachant que différentes règles ont été actives à chaque pas. Dans la recherche de but dans un labyrinthe, on peut se demander quels mouvements précédents ont permis d'atteindre le but. La plupart des algorithmes modifient la force des classeurs.

3.3.4. *Mécanisme de Génération*

Un des objectifs des systèmes de classeurs apprenant est de limiter le nombre de règles en supprimant les moins efficaces, mais également en cherchant des meilleures. Les meilleures règles sont celles qui sont les plus généralisantes possibles tout en ayant une force maximale. Pour cela deux mécanismes de génération, appelés *rule discovery*, sont utilisés ; il s'agit du *covering* et des *algorithmes génétiques*.

– Le *covering* permet de créer des règles lorsqu'aucun classeur ne s'apparie à la perception de l'environnement. Cela signifie que la population [P] possède un nombre insuffisant de règles permettant de proposer une action relative à la situation. Le *covering* peut également créer des règles lorsque les qualités des classeurs appariés sont

2. Le mécanisme de roue de la fortune est une métaphore où il faut imaginer une roulette de casino sur laquelle est placé chacun des éléments sélectionnables, la place accordée à chacun étant proportionnelle à sa sélectivité. Ensuite la bille est lancée et l'endroit où elle s'arrête indique l'élément sélectionné.

considérées insuffisantes. Dans les deux cas, de nouvelles règles sont créées avec une partie condition(s) adéquate(s) plus ou moins généralisante. Dans ce cas, la probabilité d'obtenir un # est un paramètre à fixer. La partie action est choisie aléatoirement et la *Force f* peut être la moyenne des forces de `[P]`. Imaginons par exemple que le message d'entrée provenant de `Ie` soit 0111. La population `[P]` ne possède pas de règle correspondante. Le *covering* crée une règle dont la condition peut être 0111 ou #111 ou 0##1. Cette méthode permet notamment d'initialiser le système avec une population `[P]` vide et évite la génération de règles ne correspondant à aucun état possible de l'environnement.

– Les *algorithmes génétiques* (Holland, 1975; Goldberg, 1989) sont utilisés comme `algorithme évolutionniste` pour générer de nouvelles règles à partir de celles existantes. La sélection des règles servant de base pour les opérations génétiques est généralement effectuée par un mécanisme de roue de la fortune. Les algorithmes génétiques peuvent intervenir sur les règles se situant dans `[P]` ou dans `[A]` selon les versions de système de classeurs (Wilson, 1994; Wilson, 1995). Ce mécanisme permet au système de s'adapter plus rapidement aux environnements dynamiques. La fréquence d'utilisation des algorithmes génétiques par rapport au cycle d'un système de classeurs est un facteur important pouvant influencer sensiblement les performances d'apprentissage. La sélection par algorithmes génétiques peut encore être raffinée. Deux gestions distinctes des algorithmes génétiques se distinguent au travers des types de classeurs dits « Michigan » (Holland *et al.*, 1978) ou « Pittsburgh » (Smith, 1980). La méthode « Michigan » applique les algorithmes génétiques en utilisant un unique système où chaque règle est un individu en compétition avec tous les autres alors que celle de « Pittsburgh » regroupe les règles en ensembles qui constituent alors les individus en compétition. Dans ce dernier, l'opérateur de croisement mêle les ensembles de règles. Une valeur sélective est attribuée à chaque ensemble. Les bases de règles sont mises en compétition. Elles doivent être évaluées avant de passer à une génération suivante. Par conséquent, le nombre d'évaluations est important et ne convient pas à un apprentissage incrémental d'un environnement dynamique. Néanmoins, les systèmes utilisant la méthode « Pittsburgh » présentent des qualités de stabilité et de convergence qui font souvent défaut à ceux utilisant la méthode « Michigan ».

Le mécanisme de suppression est simple puisqu'il s'agit d'éliminer les individus les « moins bons ». La qualité d'un individu reste sujette à débat. Elle dépend des types d'applications.

4. Différentes versions de systèmes de classeurs

La complexité des premiers systèmes, tels que le CS1, n'a pas donné de résultats concluants. En effet, le CS1 peut s'envoyer lui-même des messages à l'aide de la liste des messages, cela crée des cycles d'inférences internes ayant tendance à développer et à maintenir des règles parasites. De plus, d'autres problèmes ont été mis en évidence (prolifération de règles surgénéralisées, etc.). Contrairement à certains travaux qui essayent de corriger les déficiences de performance des systèmes originaux, Wil-

son choisit une approche qui consiste à les simplifier et propose le ZCS. Il réduit les systèmes originaux aux éléments essentiels (suppression de la liste des messages), tout en conservant la même architecture.

4.1. *ZCS*

Le ZCS (*Zeroth level Classifier System*) a été présenté par Wilson comme un système de classeurs de type « Michigan » (Wilson, 1994). Son fonctionnement est représenté en figure 9. Ce type de système de classeurs utilise des règles sous la forme classique du triplet $R = (C, A, f)$. Il précise les mécanismes suivants :

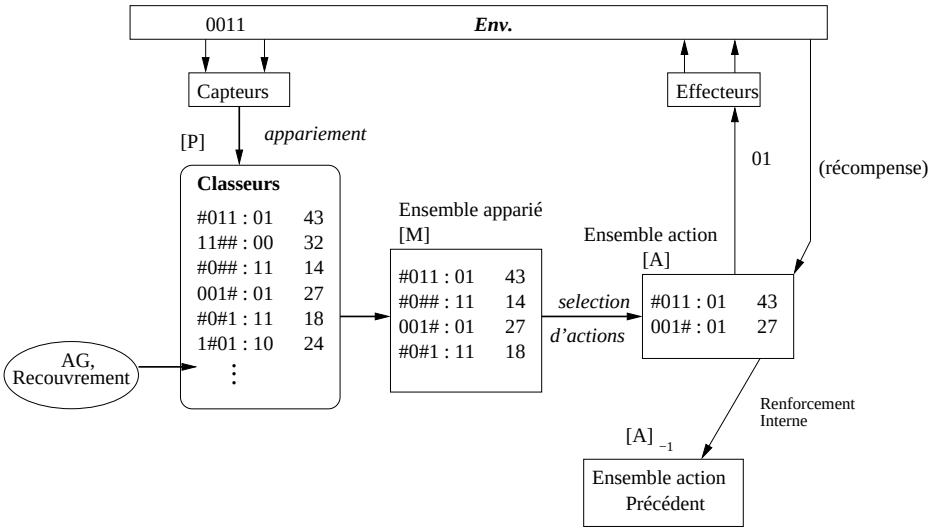


Figure 9. *Système de classeurs de type* ZCS *d'après (Wilson, 1994)*

4.1.1. *Sélection*

La `Sélection` des règles relatives à une situation s'effectue en tenant compte de la *force* des classeurs de `[M]`. Lors de la phase d'exploitation, elle est généralement soumise au principe de *roue de la fortune*.

4.1.2. *Rétribution*

La `Rétribution` est soumise à un mécanisme proche du Q-Learning : le *Bucket Brigade*. La rétribution de l'environnement est l'origine de la chaîne de rétribution, qui passe par tous les classeurs qui ont participé aux déclenchements de l'action, et se termine par les classeurs qui sont à l'origine de toute l'action. L'algorithme (Holland, 1980; Holland, 1985) intègre un système économique dans la population de classeurs `[P]`. L'idée est d'assimiler les classeurs et l'environnement

à des agents financiers. La force d'une règle peut alors être vue comme le capital de l'agent. Le mécanisme de marché met en place une compétition entre les agents concernés par la proposition de l'environnement pour avoir le droit de participer aux enchères. Pour chaque classeur de `[P]`, trois paramètres sont introduits : une *Enchère Cbid*, une *Taxe Ctaxe* et une *Rétribution Cr*. Ils sont initialisés avec une valeur nulle à chaque cycle. Nous pouvons considérer un classeur sous la forme $R = (C, A, f, s, Cbid, Ctaxe, Cr)$. Le mécanisme est constitué de trois étapes :

1) *La vente aux enchères* : chaque agent concerné donne une enchère pour participer. Elle consiste en un calcul d'une valeur *Cbid* pour les classeurs étant dans `[M]`. L'enchère proposée par chaque classeur est proportionnelle à sa *Force f* et à sa *Spécificité s*. Elle est calculée en suivant : $Cbid = (r_{const} * s) * f$ avec $0 < r_{const} \leqslant 1$. La constante r_{const} peut être comparée au taux d'apprentissage d'un algorithme connexionniste.

2) *La compétition* : une compétition est mise en place pour déterminer les agents vainqueurs. La probabilité d'être victorieux est proportionnelle à l'enchère relative. Elle consiste à calculer la probabilité qu'un classeur de `[M]` puisse gagner (mécanisme de roue de la fortune). C'est le mécanisme de `Sélection`. Les classeurs non sélectionnés réinitialisent leur *Cbid* à 0.

3) *La répartition des rétributions* va modifier la force des classeurs de `[A]`, les transactions sont effacées et chaque agent :

- reçoit une *rétribution de l'environnement* s'il est un des vainqueurs (les classeurs de `[A]`) : la rétribution *Cr* dépend de l'environnement fournissant *r* (*r* est en général divisé entre tous les classeurs) ;

- doit *payer son enchère* s'il a été vainqueur (les classeurs de `[A]`) et reçoit une *rétribution des vainqueurs* s'il est à l'origine de la situation (précédent vainqueur : `[A]` au cycle précédent noté $[A]_{-1}$) : les classeurs de `[A]` vont payer leurs enchères et la somme sera distribuée aux classeurs à l'origine de la situation présente $[A]_{-1}$. Ce mécanisme permet de favoriser l'enchaînement de règles. Le mécanisme fonctionne comme une économie où des entreprises en compétition rémunèrent leurs sous-traitants, s'enrichissant ou s'appauvrissant selon que le contrat soit remporté ou non.

La mise à jour de la force du classeur peut être soumise à un mécanisme de taxe. Celui-ci diminue la force des classeurs qui ne sont jamais choisis afin d'éliminer les règles qui ne sont jamais invoquées. *Ctaxe* est prélevée proportionnellement à la force des classeurs de `[P]`. En résumé, la *Force f* de chaque classeur de `[P]` est mise à jour à chaque cycle de fonctionnement, suivant la formule :

$$f(t) = f(t-1) - Cbid * f(t-1) - Ctaxe * f(t-1) + Cr$$

(Dorigo *et al.*, 1994) montrent l'équivalence entre le *Bucket Brigade* et le Q-Learning et soulignent le fait que le mécanisme de taxes produit le même effet sur les forces que le facteur d'amortissement mis en place dans le Q-Learning.

4.1.3. *Génération*

La `Génération` s'effectue à l'aide du *covering* ou d'algorithmes génétiques s'appliquant au niveau de la population `[P]` en considérant la force des classeurs comme valeur sélective. Ces derniers agissent à fréquence constante. L'opérateur de mutation utilise des « parents » issus de la méthode de la roue de la fortune sur `[P]`. Les nouveaux classeurs remplacent les règles les plus faibles, afin de maintenir une population constante. La suppression sélectionne des règles en utilisant la méthode de la roue de la fortune sur `[P]` en considérant la valeur inverse de la force. La force des nouvelles règles est initialisée par la valeur moyenne des forces des « parents ». Dans le cas du « covering », elle est initialisée par la valeur moyenne des forces sur `[P]`.

4.1.4. *Évolutions*

Certains systèmes proposent d'étendre les capacités du ZCS :

– le ZCSM (Mémoire) (Cliff *et al.*, 1995) ajoute une mémoire temporaire au ZCS. Chacune des parties (condition et action) des règles est étendue avec une sous-chaîne. La partie condition incorpore des bits, appelés registres, et la partie action est étendue avec un effecteur interne capable de modifier ces registres. Cette méthode permet d'augmenter les capacités des ZCS à des environnements non markoviens ;

– le ZCCS (Corporation) (Tomlinson *et al.*, 1999b) définit des liens entre les règles. Chacune possède un lien vers la règle précédente et un lien vers la règle suivante. Une corporation est un ensemble de classeurs liés entre eux. La méthode de croisement utilise des règles sélectionnées de la corporation.

4.1.5. *Limites*

Comme le souligne (Gérard, 2002), le ZCS est sujet au problème de la maintenance des longues chaînes d'actions. En effet, le mécanisme d'enchère permet de rétribuer la chaîne des règles qui ont permis d'arriver à l'action proposée. Les classeurs appariés à des situations éloignées de toute récompense sont donc rétribués, mais d'une façon moindre aux classeurs qui sont en fin de chaîne menant à la rétribution. Le ZCS utilise la force des classeurs comme valeur sélective pour l'algorithme génétique. Les classeurs qui associent une action optimale à une condition peuvent avoir une force moindre que des classeurs associant une action non-optimale à des situations plus proches d'une source de rétribution et par conséquent ils peuvent être supprimés. Ce problème augmente avec la taille de l'environnement, qui a pour effet de modifier la taille moyenne de la chaîne d'actions permettant d'atteindre un but.

4.2. *XCS*

Le XCS (Wilson, 1995) est une version de système de classeurs basée sur le ZCS. Son fonctionnement est représenté en figure 10. Il a été introduit pour pallier au problème de la maintenance des longues chaînes d'actions. Wilson propose de ne plus

considérer la force d'un classeur comme sa valeur sélective. Le XCS propose de remplacer la force d'une règle, définie initialement par les gains qu'elle permet d'espérer, par la précision de ses prédictions de gain.

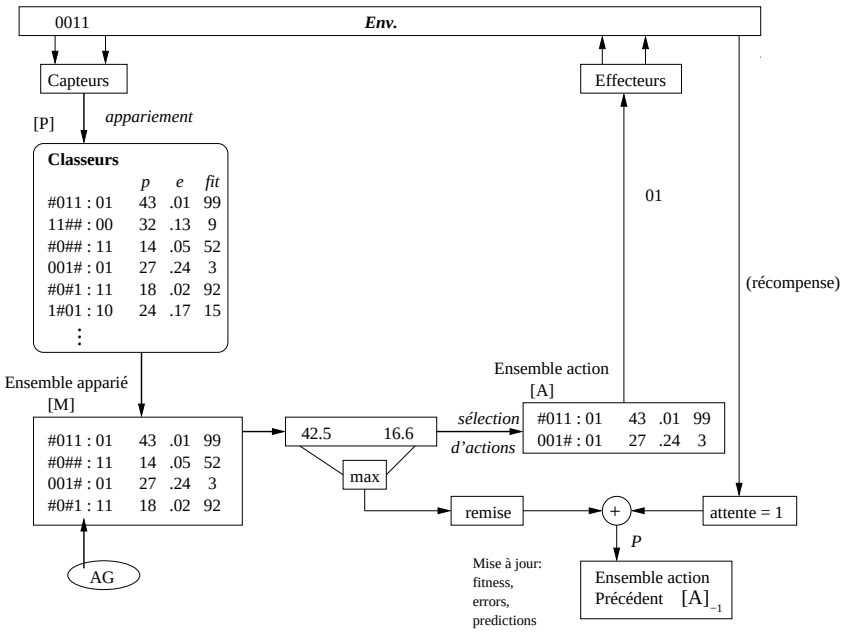


Figure 10. *Système de classeurs de type* XCS *d'après (Wilson, 1995)*

4.2.1. *Règles*

Les règles sont composées de trois paramètres venant compléter les parties condition(s) et action(s). L'ancienne *Force f* du ZCS est ainsi décomposée :

– la *Prédiction de paiement p* représente la récompense attendue en effectuant l'action A dans les situations appariées par C,

– l'*Erreur sur la prédiction e* représente l'écart entre la *Prédiction de paiement p* et le paiement réel,

– la *Fitness fit* est la fonction inverse de *e*, elle représente donc l'exactitude de la *prédiction de paiement p*.

Les classeurs peuvent être définis par $R = (C, A, p, e, fit)$ avec C et $A \in \{0, 1, \#\}^n$ et $p, e, fit \in \Re$.

4.2.2. *Sélection*

La `Sélection` se base, non plus sur la *Force f* du classeur du ZCS, mais sur la précision de la *Prédiction de paiement p*. Cette méthode permet de sélectionner les classeurs n'ayant pas seulement une action optimale, mais plutôt ceux qui permettent

de donner la qualité de l'action proposée avec précision. Elle nécessite un calcul des prédictions (*Prediction Array* PA). Pour chaque action a_i présente dans [M], une prédiction $p(a_i)$ est calculée de la manière suivante : $p(a_i) = \sum p_i.(fit)_i / \sum (fit)_i$. Dans l'exemple de la figure 10, quatre classeurs s'apparient à la situation de l'environnement et proposent les actions 01 et 11, la prédiction $p(01) = (43 * 99 + 27 * 3)/(99 + 3) = 42.5$ et $p(11) = (14 * 52 + 18 * 92)/(52 + 92) = 16.6$. (Wilson, 1996) propose de séparer explicitement les stratégies d'exploration et d'exploitation :

– pour l'exploitation, la sélection est déterministe et basée sur la plus haute prédiction ; les algorithmes génétiques sont inhibés ;

– pour l'exploration, la sélection est soumise à une probabilité permettant de sélectionner aléatoirement ou de sélectionner la plus haute prédiction.

4.2.3. *Rétribution*

La Rétribution n'est plus un *Bucket Brigade* suite à l'article de (Dorigo *et al.*, 1994). Elle est soumise à un mécanisme de « Q-Learning dérivé », la *Prédiction de paiement p* représente la fonction d'utilité $Q(s, a)$ du Q-Learning et la rétropropagation de la récompense utilise les équations de Bellman. Une fois la Sélection effectuée, l'ensemble des actions précédentes $[A]_{-1}$ (l'ensemble [A] lors du dernier cycle) est modifié en utilisant une combinaison de la dernière rétribution de l'environnement et la prédiction maximale de la PA actuelle (cf. figure 10). Le système ne cherche plus à trouver des classeurs adaptés au problème, mais à trouver une approximation de la fonction d'utilité $Q(s, a)$ sans se limiter aux classeurs proposant une action optimale.

4.2.4. *Génération*

La Génération est effectuée à l'aide du *covering*. Il permet d'initialiser [P] avec un ensemble vide ou très réduit. Le processus d'algorithme génétique ne se situe plus au niveau de la population [P] mais au niveau de $[A]_{-1}$, et se base sur l'erreur de prédiction. La *Fitness fit* est la valeur sélective. Si l'erreur est grande alors la valeur sélective est faible et inversement. Le détail des opérations est présenté dans (Butz *et al.*, 2001; Butz *et al.*, 2002) montrent que cette méthode favorise le maintien dans [P] des classeurs les plus généraux. Le classeur est conservé tant que son erreur est faible.

4.2.5. *Le problème de la maintenance de la longueur de chaîne d'actions*

Ce système résout le problème de la maintenance de la longueur de chaînes d'actions en ne cherchant pas directement à résoudre le problème de la maximisation de la récompense attendue. Le mécanisme sépare l'apprentissage de la sélection (cf. figure 11). Il apprend un modèle de la fonction de récompense en utilisant l'*Erreur sur la prédiction e* et sélectionne les classeurs en utilisant la *Prédiction de paiement p*.

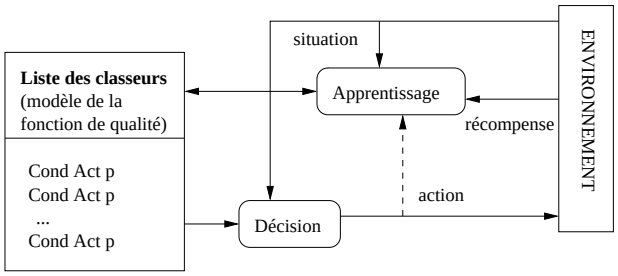


Figure 11. *Architecture de* XCS *d'après (Gérard, 2002)*

4.2.6. *Améliorations*

Des modifications ont été proposées afin d'améliorer le système dans des cas précis :

– une classification des règles (Kovacs, 1997) selon la capacité de généralisation (nombre de #) : surgénéralisée, généralisée au maximum et généralisée sous optimale. Cette classification oriente la généralisation des règles ;

– le POP-XCS (optimisé) (Kovacs, 1996) propose d'étendre la généralisation au maximum. La méthode permet de converger vers une population réduite pour des problèmes markoviens ;

– le XCSS (spécifié) (Lanzi, 1997) est utilisé pour des environnements qui permettent peu de généralisation. Il introduit un nouvel opérateur « specify » qui permet de remplacer certains symboles # par des valeurs binaires et supprime les oscillations dues à une surgénéralisation ;

– le XCSm (« messy ») (Lanzi *et al.*, 1999) propose une modification de la partie condition des règles permettant d'avoir une longueur variable. Il supprime la correspondance entre les bits de la partie condition et ceux du message d'entrée. La condition est remplacée par une séquence de gènes en deux parties (un gène est constitué du type de capteur et d'une valeur binaire) ;

– le XCSR (réel) (Wilson, 2000) permet la prise en compte des données réelles. Il modifie l'interface d'entrée, la mutation, le « covering » et l'algorithme génétique ;

– le CXCS (corporation) (Tomlinson *et al.*, 1999a) introduit le principe de corporation dans les XCS ;

– le XCSTS (« tournament selection ») (Butz *et al.*, 2003) propose de répondre au problème de performance dû à la sélection proportionnelle des parents de l'algorithme génétique. Les parents sont sélectionnés en considérant deux sous - populations choisies aléatoirement. Les vainqueurs sont ceux qui ont la plus grande *fitness*. Cette modification rend les XCS plus efficaces.

4.3. *Systèmes de classeurs à anticipation*

Le problème majeur des systèmes de classeurs vus jusqu'à présent, réside dans les temps de convergence de l'apprentissage. Il s'avère que l'exploration de l'environnement est sous-utilisée puisque c'est simplement lorsqu'une récompense est reçue qu'il y a un apprentissage. Or, durant cette exploration il est possible de faire un apprentissage latent (cf. figure 12). Le principe est de découvrir un modèle de l'environnement de façon découplée de l'apprentissage par renforcement et de rentabiliser ainsi les différentes explorations non exploitées en l'absence de renforcement avec des algorithmes tels que le Q-learning. Le modèle de l'environnement est une liste des probabilités de passage d'état à état (par le biais d'actions) apprise au fur et à mesure des évolutions du système par évaluation statistique. Les systèmes de classeurs à anticipation proposent une approche similaire, remplaçant l'énumération exhaustive des transitions entre états par une base de règles généralisantes, caractérisant les modifications apportées à l'environnement par les actions. Une telle généralisation permet de refléter en une seule règle des *régularités* de l'environnement. Par exemple, dans un environnement peuplé d'obstacles, une règle peut représenter le fait que, foncer sur l'un d'eux, ne permet pas de bouger. Les différentes versions de ce type de systèmes de classeurs se distinguent par les modèles de l'anticipation qui peuvent mettre en évidence différents types de régularités, ainsi que les règles de spécialisation et de généralisation qui en découlent (cf. figure 13).

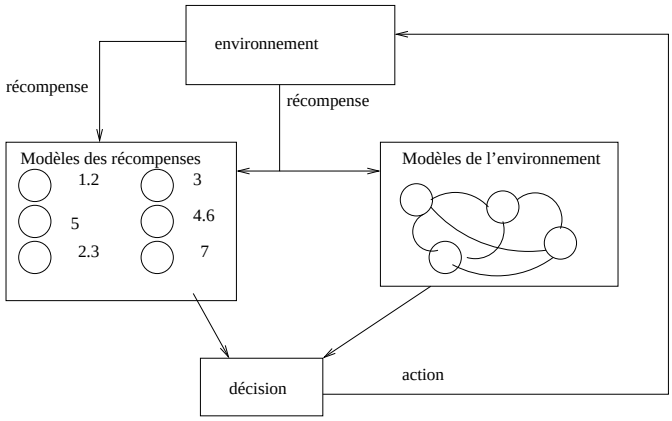


Figure 12. *Apprentissage artificiel latent*

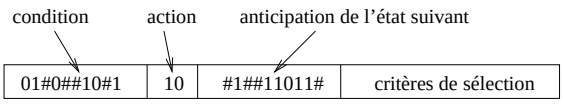


Figure 13. *Modèle d'une règle d'un système de classeur anticipant*

4.3.1. *ACS*

Les ACS (*Anticipatory Classifier System*) (Stolzmann, 1998) sont destinés à une utilisation dans une architecture de type *Dyna*. Pour introduire l'apprentissage latent, une partie *effet* est ajoutée à chaque règle ainsi qu'une *qualité d'anticipation* notée `fa`. C'est cette qualité d'anticipation qui sera utilisée pour sélectionner les règles (conjointement à la prévision de renforcement). Un effet est une représentation similaire à celle de la partie condition mais reflétant l'état espéré du système suite à l'application de cette règle. Pour cette partie effet, une sémantique différente est associée au symbole # : il représente l'absence d'un changement d'attribut et, par conséquent, n'est pas à proprement parlé, un symbole de généralisation. La figure 14 représente une règle d'un tel classeur et montre les états anticipés par celle-ci.

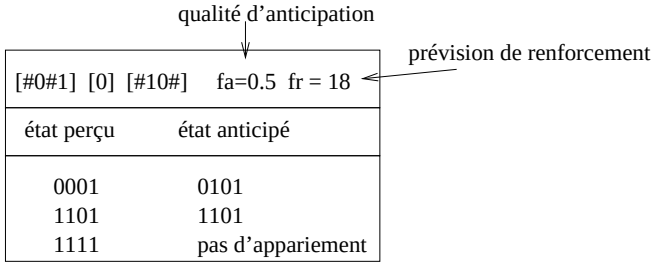


Figure 14. *Fonctionnement d'une règle dans un* ACS

Pour accélérer le processus d'apprentissage, l'ACS utilise des heuristiques de spécialisation plutôt que les algorithmes génétiques. Ces heuristiques (issues des théories de Hoffman sur l'apprentissage latent dans le contrôle du comportement animal par anticipation), consistent à modifier une *qualité d'anticipation* d'une règle en fonction du résultat de l'application de l'action qui lui est associée et éventuellement à spécialiser celle-ci : si le résultat prédit par la règle n'est pas celui obtenu, sa *qualité d'anticipation* est diminuée, mais si l'erreur avait pu être évitée en spécialisant la règle, l'algorithme effectue cette spécialisation en remplaçant les # par les symboles adéquats. Cette spécialisation est cependant très brutale car elle concerne en bloc tous les attributs mal évalués à la fois dans la partie condition et dans la partie action. La modélisation proposée ne permet pas de distinguer les attributs pertinents à spécialiser. Pour pallier cette sur-généralisation, un algorithme génétique classique s'emploie à générer de nouveaux classeurs. Les classeurs qui anticipent mal sont supprimés.

4.3.2. *YACS*

Le YACS (*Yet Another Classifier System*) (Gérard, 2002) utilise le même formalisme que le ACS avec des heuristiques différentes : il prend en compte un historique des erreurs de prédiction des classeurs ainsi qu'une mémorisation des valeurs des attributs lors du dernier échec et du dernier succès de prédiction. Ceci permet d'élaborer des heuristiques permettant de modifier indépendamment les parties condition et action des règles et d'obtenir des spécialisations et des généralisations beaucoup plus

fines, prenant en compte le *rôle* distinct de chaque attribut lors d'une erreur de prédiction. L'historique des erreurs de prédiction (sur un horizon fini) permet de détecter un classeur fiable que l'on préservera, d'un classeur oscillant, dont il faut spécialiser la partie condition. Enfin un classeur peu fiable, très souvent en échec, sera supprimé. La mémorisation des valeurs des attributs, lors des dernières situations d'échec et de succès, permet de définir une *estimation* de l'amélioration attendue en spécialisant chaque attribut généralisant (#) qui reflétera sa *chance* d'être modifié par généralisation : si un classeur a bien anticipé, on compare l'ancienne situation où il fut en échec de prédiction, à la situation qui vient de fonctionner : tous les attributs # qui permettent de distinguer ces deux situations, voient leur estimation augmenter (les autres la voient diminuer). A l'inverse, si l'anticipation n'a pas été correcte, on compare l'ancienne situation de succès à celle qui vient d'échouer et tous les attributs généralisant (#) qui distinguent ces situations, voient leur estimation augmenter. C'est l'*estimation* des attributs qui permet de choisir celui qui sera spécialisé. Notons également que, pour éviter des spécialisations abusives, un regroupement de classeurs similaires (possédant la même partie condition) est effectué, et que l'algorithme attend que ces classeurs soient tous oscillants pour les spécialiser. La généralisation est également effectuée à l'aide d'heuristiques similaires partant d'une estimation de l'amélioration attendue de la généralisation de chaque attribut spécialisé. Pour cela YACS s'intéresse aux classeurs qui n'ont pas été sélectionnés, à cause d'une partie condition trop spécialisée, alors qu'ils auraient fait une bonne anticipation (tout en ayant déclenché la bonne action). Comme pour la spécialisation, on attend d'avoir une bonne estimation de la fiabilité d'un classeur avant de le généraliser, et on opère par groupe de classeurs à effets similaires. Enfin, un mécanisme de couverture est introduit pour générer des règles s'appariant à une situation lorsqu'il n'en existe pas. YACS découple plus explicitement l'apprentissage par renforcement de l'apprentissage latent, puisqu'aucune force n'est associée aux règles : il y a une prévision de récompense apprise pour chaque état à l'aide d'un apprentissage par renforcement classique. Cette prévision est utilisée après les prédictions calculées par les règles pour choisir l'action à effectuer.

4.3.3. *MACS*

Le MACS (*Modular Anticipatory Classifier System*) (Gérard, 2002) est un formalisme pour l'apprentissage latent qui introduit les anticipations partielles : ses règles anticipent la valeur d'un seul attribut, pour les autres, on ne fait aucune hypothèse. Ce système offre de nouvelles possibilités de généralisation et d'apprentissage de régularités. En particulier, il permet d'apprendre des causalités entre attributs (la valeur de l'attribut n est répercutée sur celle de l'attribut $n + 1$ à chaque fois que l'on fait une action a et quelle que soit la valeur des autres attributs), très fréquentes dans le cas de perceptions partielles. La conséquence immédiate de cette approche est que plusieurs règles vont définir l'état, suivant l'exécution d'une action. Toutes les règles correspondant à l'exécution en cours (s'appariant avec l'état du système avant application de l'action), vont être évaluées : à chaque règle sont associées deux valeurs g et b comptant respectivement le nombre d'échecs et le nombre de succès de celle-ci. Ces nombres sont utilisés pour décider de la suppression de la règle. Un mécanisme

similaire à YACS permet de définir les attributs devant être généralisés et ceux devant être spécialisés : on spécialisera l'attribut d'un classeur oscillant discriminant au maximum les échecs des succès, et on généralisera l'attribut d'un classeur fiable ne discriminant pas les échecs des succès.

Au premier abord, toutes les améliorations effectuées sur ces systèmes améliorent les performances, mais il est toujours possible de trouver des contre-exemples. Par exemple, si l'environnement est stochastique, la plupart des systèmes ayant remplacés les algorithmes génétiques par des heuristiques sont pénalisant car ils peuvent tomber dans des minima locaux, dont le facteur aléatoire des algorithmes génétiques permet de sortir.

4.4. *Abstraction et hétérogénéité des données*

Bien que l'utilisation des systèmes de classeurs ait produit des résultats intéressants, les modèles présentés précédemment souffrent d'un problème de granularité de représentation, lorsqu'il s'agit d'établir des connaissances mêlant des informations de portée sémantique différente (Barry, 1993). En effet, prenons l'exemple d'un système pour lequel certains attributs représentent la position d'une entité permettant de définir sa vitesse pour atteindre un but, et d'autres sa stratégie de défense à long terme face à une attaque. Cette hétérogénéité des données implique des niveaux de décisions différents avec éventuellement des échelles de temps différents. Si elle est *identifiable*, l'introduction de structures dans les systèmes de classeurs permet d'abstraire (tâches simples et tâches complexes), de décomposer (décomposer un problème complexe pour le résoudre plus facilement) et de réutiliser (solutions des sous-problèmes) les connaissances (Barry, 1996), pour améliorer leurs performances. C'est ce que proposent les systèmes hiérarchiques et les systèmes de classeurs hétérogènes.

4.4.1. *ALECSYS*

Le ALECSYS (*A LEarning Classifier SYStem*) (Dorigo, 1995) est une structure hiérarchique permettant de décomposer une tâche complexe en un ensemble de tâches simples. Un système de classeurs joue le rôle d'arbitre dans le choix de l'activation d'un des autres systèmes qui apprennent les comportements de bases. Le simulateur AutonoMouse permet d'évaluer le modèle. Une souris doit suivre une source lumineuse mobile, et doit se rendre dans son terrier à l'apparition d'un prédateur. L'architecture est sur deux niveaux. A la base, deux systèmes de classeurs apprenant les deux comportements de base en parallèle : suivre la lumière et éviter le prédateur. Au sommet, un système de classeurs choisit le comportement qui va être activé.

4.4.2. *OCS*

Le OCS (*Organizational Classifier System*) (Takadama *et al.*, 1999) est composé de différents agents possédant chacun un système de classeurs devant satisfaire une tâche commune. Les agents agissent collectivement pour résoudre un problème. Le mécanisme de spécialisation et la communication entre agents, permettent d'augmen-

ter l'efficacité du système. Ce système a été appliqué à la conception de circuit électronique (Takadama *et al.*, 2001). Chaque système de classeurs représente un composant électronique.

4.4.3. *MHICS*

Le MHICS (*Modular HIerarchical Classifier Systems*) (Robert *et al.*, 2003) est une architecture qui assemble des modules distribués sur plusieurs niveaux. Le premier niveau contrôle les motivations de l'entité (agressivité, faim, etc.). Il calcule une intensité pour chaque motivation en considérant une force relative à l'entité (PP) et une valeur de motivation (VM). La première correspond à la personnalité de l'entité, et la seconde évolue en fonction de la satisfaction obtenue préalablement. Un système de classeurs est associé à chaque motivation. L'objectif de chaque système est de satisfaire la motivation qui l'active. Les actions des règles des différents systèmes de classeurs du niveau I activent les classeurs du second niveau (pas dans l'implémentation actuelle) ou détermine directement des actions du niveau III (se déplacer vers la cible, tirer sur la cible, etc.). Le troisième niveau sélectionne les priorités entre les actions exécutables dépendantes des valeurs de motivations des niveaux I et II. Le niveau quatre fournit les ressources pour l'exécution des actions (angle, déplacement vers l'avant, etc.). L'action ayant la plus haute intensité (valeur provenant du niveau III) a la priorité pour utiliser les ressources. Cette architecture est utilisée pour contrôler des entités virtuelles pour des jeux en ligne massivement multi-utilisateurs.

4.4.4. *HGCS*

Les HGCS (Systèmes de Classeurs hétérogènes généralisés) (Sanchez, 2004) abordent l'hétérogénéité des données sous un autre angle : il n'y a pas une hiérarchie de systèmes de classeurs, mais une classification des données en types distincts. Par exemple, il existe un type de données correspondant à un intervalle de valeurs possibles d'une entrée entière. La structure d'un HGCS est donc très proche de celle d'un ZCS ou d'un XCS (il existe d'ailleurs des GHZCS et des GHXSC), ce qui l'en distingue, c'est l'utilisation d'algorithmes génétiques hétérogènes pour sélectionner, généraliser ou spécialiser les règles. Ces algorithmes utilisent des opérateurs de mutation et de croisement, spécifiques à chaque ensemble de données. Ces opérateurs sont définis en fonction de la nature des données, de sorte que les *chromosomes* générés ne dépendent pas d'une loi aléatoire quelconque, mais introduisent un biais représentatif améliorant les performances du système.

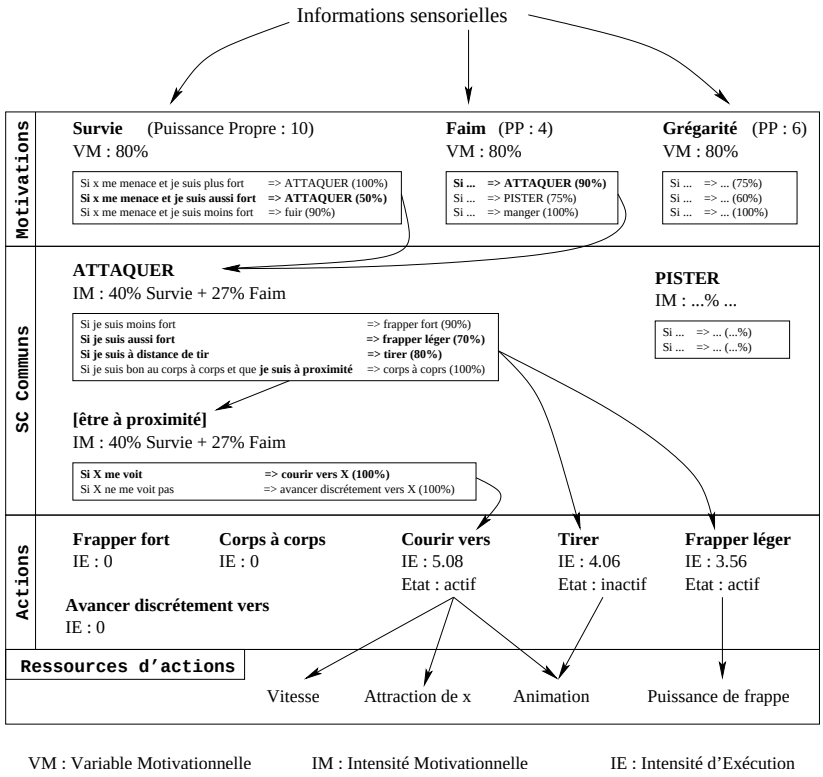


Figure 15. *La sélection d'action à travers les niveaux dans* MHICS

5. Conclusion

Cet article a abordé une présentation globale des systèmes de classeurs. Nous avons pu voir, au travers des différentes versions existantes, les problèmes qui ont été rencontrés ainsi que les solutions proposées. Les systèmes que nous avons présentés permettent de trouver des solutions optimales dans des environnements markoviens et, pour certain, non markoviens.

Au-delà des systèmes classiques, les systèmes de classeurs ont été combinés à d'autres techniques d'intelligence artificielle. Par exemple, certains systèmes remplacent les règles par des réseaux de neurones (Bull, 2002), intègrent des notions de logique floue (Valenzuela-Rendon, 1991), ou combinent les deux techniques (Bull *et al.*, 2002). Ainsi, les modèles utilisant ou se basant sur les systèmes de classeurs classiques fleurissent. Si de nombreux modèles sont proposés, peu de travaux mettent en évidence les critères qui pourraient permettre de choisir parmi les systèmes de classeurs classiques pour traiter un problème.

La nature heuristique des algorithmes sous-jacents aux systèmes de classeurs, implique qu'il n'y ait pas de système *idéal*, s'adaptant à tout problème. Comme le fait remarquer (Sanza, 2001) les systèmes sont adaptés à des cas précis, et aucun de ces modèles n'a vocation à apporter une solution globale pour tous les problèmes. Par exemple, il est difficile de définir si un MACS est adapté à des systèmes possédant des chaînes d'actions longues. Si la perception de l'environnement est non markovienne, il faudra introduire des mémoires internes (XCSM au lieu de XCS), si les récompenses sont peu fréquentes, les chaînes d'actions sont donc longues et il serait préférable de privilégier un système dont la sélectivité est basée sur la précision de prédiction que sur la valeur des renforcements (XCS plutôt que ZCS). Le XCS n'est efficace que si les rétributions sont discrètes et en nombre fixe, le ACS n'est utile que si chaque action engendre une modification dans la perception du monde. *A priori*, les systèmes à anticipation apprennent plus rapidement en termes de nombre de pas de simulation à effectuer. Cependant, le temps de calcul de ces pas peut être très important, car, à chacun d'eux, on réévalue le modèle de l'environnement. Si l'on opte pour un apprentissage latent, l'étude des régularités induite par le couple (perception-environnement) orientera le choix. Enfin, l'hétérogénéité des données et la définition de niveaux d'abstraction de ces données favorisent une approche hiérarchique.

Ces remarques appellent à une mise en œuvre formelle de critères de sélection de systèmes de classeurs. Cependant, celle-ci nous semble difficile par le fait des liens entre ces critères potentiels : le choix ne peut se résoudre à un ensemble de règles (si-alors). L'écueil possible est alors d'obtenir des mécanismes de formalisation qui demandent un effort d'analyse dont on cherche justement à se passer lorsque l'on utilise des mécanismes d'apprentissage artificiel.

6. Bibliographie

Barry A., « The Emergence of High Level Structure in Classifier Systems - A Proposal », *Irish Journal of Psychology*, vol. 14, n° 3, p. 480-498, 1993.

Barry A., « Hierarchy Formulation Within Classifiers System – A Review », *in* E. G. Goodman, V. L. Uskov, W. F. Punch (eds), *Evolutionary Computation and Its Applications (EVCA'96)*, The Presidium of the Russian Academy of Sciences, Moscow, p. 195-211, 1996.

Bellman R., *Dynamic Programming*, Princeton University Press, 1957.

Bonelli P., Parodi A., Sen S., Wilson S., « NEWBOOLE : A Fast GBML System », *International Conference on Machine Learning*, Morgan Kaufmann, San Mateo, California, p. 153-159, 1990.

Bull L., « On Using Constructivism in Neural Classifier Systems », *Parallel Problem Solving from Nature (PPSN 2002)*, Granada, Spain, p. 558-567, 2002.

Bull L., O'Hara T., « Accuracy-based Neuro and Neuro-Fuzzy Classifier Systems », *in* W. Langdon, E.Cantu-Paz, K.Mathias, R. Roy, D.Davis, R. Poli, K.Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. Schultz, J. F. Miller, E. Burke, N.Jonoska (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, p. 905-911, 2002.

Butz M., Pelikan M., « Analyzing the Evolutionary Pressures in XCS », *Genetic and Evolutionary Computation Conference (GECCO'01)*, Morgan Kaufmann, San Francisco, California, USA, p. 935-942, 7-11 July, 2001.

Butz M., Sastry K., Goldberg D., « Tournament selection : stable fitness pressure in XCS », *Genetic and Evolutionary Computation Conference (GECCO'03)*, Springer, p. 1857-1869, 2003.

Butz M., Wilson W. S., « An Algorithmic Description of XCS », *Journal of Soft Computing*, vol. 6, n° 3–4, p. 144-153, 2002.

Cliff D., Ross S., Adding Temporary Memory to ZCS, Technical Report n° CSRP347, School of Cognitive and Computing Sciences, University of Sussex, 1995.

Dorigo M., « Alecsys and the AutonoMouse : Learning to Control a Real Robot by Distributed Classifier Systems », *Machine Learning*, vol. 19, p. 209-240, 1995.

Dorigo M., Bersini H., « A Comparison of Q-Learning and Classifier Systems », *From Animals to Animats 3 : Third International Conference on Simulation of Adaptive Behavior (SAB'94)*, p. 248-255, 1994.

Gérard P., Systèmes de classeurs : étude de l'apprentissage latent, PhD thesis, Université Paris VI, 2002.

Girard B., Robert G., Guillot A., « Jeu Vidéo et Intelligence Artificielle Située », *In Cognito*, vol. 22, p. 57-72, 2001.

Goldberg D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

Heguy O., Sanza C., Berro A., Duthen Y., « GXCS : A generic Classifier System and its application in a Real Time Cooperative Behavior Simulations », *International Symposium and School on Advanced Distribued System (ISADS'02)*, Guadalajara, Mexique, 2002.

Holland J., « Adaptation », *in* R. Rosen, F. M. Snell (eds), *Progress in theoretical biology*, New York : Plenum, 1976.

Holland J. H., *Adaptation in Natural and Artifcial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

Holland J. H., « Adaptive algorithms for discovering and using general patterns in growing knowledge bases », *International Journal of Policy Analysis and Information Systems*, vol. 4, p. 245-268, 1980.

Holland J. H., « Properties of the Bucket Brigade Algorithm », *International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, p. 1-7, 1985.

Holland J., Reitman J., « Cognitive systems based on adaptive algorithms », *in* D. A. Waterman, F. Hayes-Roth (eds), *Pattern-directed inference systems*, New York : Academic Press, 1978. Reprinted in : Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998. ISBN : 0-7803-3481-7.

Jong K. A. D., An analysis of the behavior of a class of genetic adaptive systems., PhD thesis, George Mason University, Fairfax, VA, 1975.

Klopf A. H., Brain function and adaptive systems - A heterostativ theory, Technical Report n° AFCRL72 -0164, Air Force Cambridge Research Laboratories, 1972.

Kovacs T., Evolving Optimal Populations with XCS Classifier Systems, Technical report, October, 1996.

Kovacs T., XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions, Technical Report n° CSRP-97-19, School of Computer Science, University of Birmingham, Birmingham, U.K., 1997.

Lanzi P. L., « A Study of the Generalization Capabilities of XCS », *in* T. Bäck (ed.), *the Seventh International Conference on Genetic Algorithms*, Morgan Kaufmann, San Francisco, CA, p. 418-425, 1997.

Lanzi P. L., Colombetti M., « An Extension to the XCS Classifier System for Stochastic Environments », *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, R. E. Smith (eds), *Genetic and Evolutionary Computation Conference (GECCO'99)*, Morgan Kaufmann, Orlando, Florida, USA, p. 353-360, 13-17 July, 1999.

Matteucci M., Fuzzy Learning Classifier System : Issues and Architecture, Technical report, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999.

Pavlov I. P., *Conditionned Reflexes*, Oxford University Press, New York, 1927.

Robert G., Guillot A., « MHiCS, A Modular And Hierarchical Classifier Systems Architecture For Bots », *in* Q. Mehdi, N. Gough, S. Natkin (eds), *4^{th} International Conference on Intelligent Games and Simulation (GAME-ON'03)*, London, United Kingdom, p. 140-144, 2003.

Robert G., Portier P., Guillot A., « Classifier systems as 'Animat' architectures for action selection in MMORPG », *in* Q. Medhi, N. Gough, M. Cavazza (eds), *3^{rd} International Conference on Intelligent Games and Simulation (GAME-ON'02)*, SCS, London, United Kingdom, p. 121-125, november, 2002.

Rust J., *Handbook of Computational Economic*, vol. 1, Elsevier, North-Holland, chapter Numerical dynamic programming in economomics, p. 614-722, 1996.

Sanchez S., Mecanismes evolutionnistes pour la simulation comportementale d'acteurs virtuels, PhD thesis, Université des Sciences Sociales Toulouse I, Décembre, 2004.

Sanza C., Evolution d'Entités Virtuelles Coopératives par Système de Classifieurs, PhD thesis, Université Paul Sabatier, 2001.

Schulenburg S., Ross P., « Strength and Money : An LCS Approach to Increasing Returns », *Third International Workshop on Advances in Learning Classifier Systems (IWLCS'00)*, Springer-Verlag, London, UK, p. 114-137, 2001.

Skinner B. F., *The behavior of organims*, Appleton-Century-Crofts, New York, 1938.

Smith J., Vavak F., « Replacement Strategies in Steady State Genetic Algorithms : Static Environments. », *FOGA*, p. 219-234, 1998.

Smith S. F., A Learning System Based on Genetic Adaptive Algorithms, PhD thesis, University of Pittsburgh, 1980.

Stolzmann W., « Anticipatory classifier systems », *Third Annual Genetic Programming Conference*, Morgan Kaufmann, p. 658-664, 1998.

Sutton R. S., Temporal Credit Assignement in Reinforcement Learning, PhD thesis, University of Massachusetts, 1984.

Sutton R. S., « Learning to Predict by the Methods of Temporal Differences », *Machine Learning*, vol. 3, p. 9-44, 1988.

Sutton R. S., « Reinforcement learning architectures for animats », *in* J.-A. Meyer, W. S. Wilson (eds), *From Animals to Animats : First International Conference on Simulation of Adaptive Behavior (SAB'91)*, p. 288-296, 1991.

Sutton R. S., « Reinforcement Learning Architectures for Animats », *in* J. Meyer, H. Roitblat, W. S. Wilson (eds), *From Animals to Animats 2. The second International Conference on Simulation of Adaptive Behavior*, MIT Press, 1992.

Takadama K., Terano T., Shimohara K., « Learning Classifier Systems Meet Multiagent Environments », *Third International Workshop on Advances in Learning Classifier Systems (IWLCS '00)*, Springer-Verlag, p. 192-212, 2001.

Takadama K., Terano T., Shimohara K., Hori K., Nakasuka S., « Can Multiagents Learn in Organization ? Analyzing Organizational-Learning Oriented Classifier Systems », *Agents Learning about, from and with other Agents*, 1999.

Tolman E., Honzik C., « Insight in Rats », *University of California Publications in Psychology*, vol. 4, p. 215-232, 1930.

Tomlinson A., Bull L., « A Corporate XCS », *Proceedings of International Workshop on Learning Classifier Systems*, p. 298-305, 1999a.

Tomlinson A., Bull L., « A zeroth level corporate classifier system », *Second International Workshop on Learning Classifier Systems (IWLCS'99)*, p. 306-313, 1999b.

Valenzuela-Rendon M., « The Fuzzy Classifier System : a Classifier System for Continuously Varying Variables », *in* L. Booker, R. Belew (eds), *Proceedings of the Fourth International Conference on Genetic Algorithms*, p. 346-353, 1991.

Watkins C. J. C. H., Dayan P., « Q-Learning », *Machine Learning*, vol. 8, n° 3-4, p. 279-292, 1992.

Wilson W. S., « ZCS : A zeroth level classifier system », *Evolutionary Computation*, vol. 2, n° 1, p. 1-18, 1994.

Wilson W. S., « Classifier Fitness Based on Accuracy », *Evolutionary Computation*, vol. 3, n° 2, p. 149-175, 1995.

Wilson W. S., « Explore/exploit strategies in autonomy », *From Animals to Animats 4 : the Fourth International Conference on Simulation of Adaptive Behavior (SAB'96)*, p. 325-332, 1996.

Wilson W. S., « Get Real ! XCS with Continuous-Valued Inputs », *Learning Classifier Systems, From Foundations to Applications*, p. 209-222, 2000.

Article reçu le 17 mars 2005
Accepté après révision le 20 février 2006

Cédric Buche *est post-doctorant au Centre Européen de Réalité Virtuelle (CERV). Ses travaux concernent les environnements virtuels de formation et les techniques d'apprentissage artificiel.*

Cyril Septseault *est doctorant au CERV. Sa recherche porte sur la simulation de comportements humains crédibles et de leur adaptation dans des environnements virtuels incertains.*

Pierre De Loor *est Maître de Conférences au CERV. Sa recherche concerne l'autonomisation de modèles pour la résolution de problème et la simulation de comportements.*