Adding a Generalization Mechanism to YACS

Pierre Gérard *,**
* AnimatLab (LIP6)
8, rue du Capitaine Scott
75015 PARIS

Olivier Sigaud **
** Dassault Aviation, DGT/DPR/ESA
78, Quai Marcel Dassault
92552 St-Cloud Cedex

Abstract

A new and original trend in the Learning Classifier System (LCS) framework is focussed on *latent learning*. These new LCSs call upon classifiers with a [*condition*], an [*action*] and an [*effect*] part. In the LCS framework, the *latent learning* process is in charge of discovering classifiers which are able to anticipate accurately the consequences of actions under some conditions. Accordingly, this process builds a model of the dynamics of the environment. This paper describes how YACS performs latent learning, and how it is enhanced by a dedicated generalization process which offers an alternative to Genetic Algorithms.

1 INTRODUCTION

Holland [Hol76] presented the first ideas about LCSs (Learning Classifier Systems). The capability of generalizing is the main advantage of LCSs with respect to other reinforcement learning systems like *Q-learning* [Wat89]. It allows to consider several perceived situations within a common description so that the representation of the problem gets smaller. The accuracy based approach in Wilson's XCS [Wil95] overcomes the problem in previous LCSs where especially deferred reward leads to over-generalization.

Another concern in the general reinforcement learning framework is to build an internal model of the dynamics of the environment. This model can be used to adapt the policy further and faster. In multi-step problems, an agent can learn to *anticipate* what happens immediately after the execution of an action. This learning process can take place even in the absence of reward. Such a model of the dynamics of the environment can be learned *latently* and allows lookahead mechanisms. In order to use LCSs to learn a model of the dynamics of the environment, Holland [Hol90] proposed an implicit approach based on tagged internal messages. Riolo [Rio91] implemented this idea in his CFSC2 and demonstrated its latent learning capability. A more explicit linkage is used in CXCS [TB00].

In contrast with all these approaches, ACS (Anticipatory Classifier System, [Sto98]) and YACS (Yet Another Classifier System, [GSS01]) both form $C-A-E$[1] classifiers. This formalism is similar to Sutton's DynaQ+ [Sut91] approach but draws benefits of the generalization capability of LCSs. ACS and YACS both take advantage of the information provided by the succession of situations in order to drive the classifier discovering process. Therefore, they use heuristics instead of Genetic Algorithms, which are general but not explicitly driven by experience. This way, YACS explores the solution space rationally, so as to be able to tackle large problems like the Sheep-dog problem described in [SG01].

In [GSS01], we showed how the latent learning process in YACS leads to near-optimal but not optimal representations of the dynamics of the environment. This paper focuses on the *latent learning* process of YACS and presents the generalization process which overcomes the near-optimality problem.

In section 2 we show how the formalism used in YACS allows generalization. In section 3 we briefly describe the heuristics used for the *latent learning* process in YACS. For further details or a comparison with ACS, please refer to [GSS01]. In section 4 we describe how we introduce a generalization process in YACS. In section 5 we show experimentally how this new process helps to overcome the over-specialization problems in YACS.

[1] C stands for [*condition*], A for [*action*] and E for [$effect$]

2 GENERALIZATION IN YACS

As ACS [Sto98], YACS deals with *C*-*A*-*E* classifiers [2]. *C* parts take advantage of generality and may match several perceived situations. An *A* part specifies a particular action possible in the environment.

A situation is divided into several features representing perceivable properties of the environment. A *C* part has the same structure but it may contain *don't care* symbols **"#"**. The *E* part stores for each perceived feature the expected changes in the environment when the action of the classifier is chosen and when the perceived situation matches its condition. The *E* part might contain *don't change* symbols **"#"**. A *don't change* symbol in the *E* part means *"the feature of the perceived situation corresponding to the* don't change *symbol remains unchanged"*.

This formalism allows the classifiers to represent regularities in the environment like for instance *"In a maze, when the agent perceives a wall on north, whatever the other features are, moving north will drive the agent to hit the wall, and no change will be perceived"*

In YACS, generalization is allowed by the joint use of *don't care* and *don't change* symbols. As ACS, YACS generalizes over the anticipation of an expected effect in terms of situations, and not over the prediction of a payoff, as in XCS [Wil95].

Thus, what we call *generalization* in YACS is not the same as the generalization studied in XCS by [Lan97] for instance. As a result, it does not make sense to store information about the expected payoff in the classifiers. The list of classifiers only models the transitions in the environment.

As we showed in [GSS01], so as to perform reinforcement learning, YACS must deal with information about specific situations. So, this system uses a set *P* of every perceived situation encountered during the lifetime of the agent. This set only contains one single instance of each already perceived situation. Each situation is valued by the expected payoff when reaching the considered situation.

This set only contains the actually perceived situations, not all the virtually possible situations resulting from the number of features and the number of values they can take. In a large problem like the multi-agent Sheep-dog problem described in [SG01] for instance, the number of actually encountered situations is 290 while the number of virtually possible situations is 8192.

[2] *C* stands for [*condition*], *A* for [*action*] and *E* for [*effect*]

A way to reduce the size of this set could be to provide to YACS with a dedicated generalization mechanism which relies on the expected payoff.

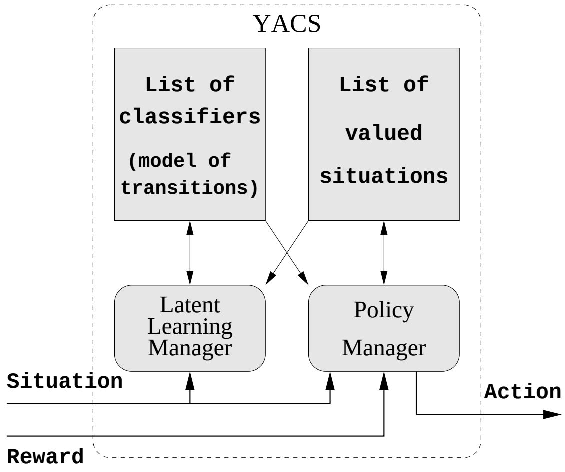

Figure 1: The YACS architecture

So, as shown in figure 1, YACS consists in several parts:

- a latent learning manager which updates the classifiers list;
- a policy manager which is in charge of updating a set of valued encountered situations. The policy manager is also in charge of selecting actions.

3 LATENT LEARNING IN YACS

The *latent learning* process is in charge of discovering $C - A - E$ classifiers with maximally general *C* parts that accurately model the dynamics of the environment. Unlike ACS, it learns *C* and *E* parts separately. So as to discover accurate *C* and *E* parts, YACS associates additional information to the classifiers[3]. As a result, a classifier in YACS needs more memory, but this information is used in order to reduce the complexity of the resulting model in terms of number of classifiers.

In the following sections, we briefly give the main mechanisms of the *latent learning* process as it were described in [GSS01]. For further details, please refer to this paper.

[3] two situations, a finite set of booleans markers and two sets estimates which are real numbers.

3.1 EFFECT COVERING

The effect covering mechanism is the part of the latent learning process is in charge of discovering accurate E parts (*i.e.* E parts representing actual effects of actions under some conditions). When the system learns accurate effects, it creates new classifiers with suitable E parts settled according to experience, by direct comparison of successive perceived situations.

This mechanism causes major problems in noisy environments. In such environments, it may create a lot of classifiers. Thus, we work on a new version of YACS without the effect covering mechanism.

During the *effect covering* process, YACS also updates a trace T of *good* and *bad* markers memorizing past anticipation mistakes and successes of each classifier. This trace works as a FIFO list with a finite length m.

3.2 SELECTION OF ACCURATE CLASSIFIERS

As YACS tries to build a set of classifiers that anticipate accurately, it has a deletion mechanism to remove inaccurate classifiers. The trace T of *good* and *bad* markers allows to check the anticipation abilities of a classifier.

If the trace T of a classifier is full and if it only contains *bad* markers, then YACS assumes that the classifier always anticipates incorrectly and removes it. If the trace is full and if it contains *good* and *bad* markers, we say that the classifier *oscillates* because its condition is too general. In this case, the condition must be further specialized.

3.3 SPECIALIZATION OF CONDITIONS

A C part should be as general as possible in order to represent regularities in the environment. But it must be specific enough so that the classifier does not oscillate. The specialization process incrementally specializes C parts so as to reach the right level of generality.

The classifier discovery problem is usually solved by a Genetic Algorithm. But the genetic operators do not explicitly take advantage of the experience of the agent.

YACS starts without making any distinction between situations, and incrementally introduces experience driven specializations in C parts. It uses neither mutation nor crossover operators.

The specialization process of YACS uses the *mutspec* operator introduced by [Dor94]. This operator selects a general feature of the C part[4] of an *oscillating* classifier, and produces one new classifier for each possible specific value of the selected feature. YACS improves the selection of the features to specialize by using the *expected improvement by specialization* estimate i_s associated to each *don't care* symbol in the C part of each classifier. This value estimates how much the specialization of the token would help to split the situation set covered by the C part into several sub-sets of equal cardinality.

4 GENERALIZATION OF CONDITIONS

In section 3.3 we have presented how YACS specializes C parts so as to allow the E part to be accurate. But even if this process is cautious, it may produce classifiers with a C part at a sub-optimal level of generality, in particular when YACS specializes C parts while it did not experience many possible situations.

As the specialization process, the generalization uses heuristics in order to take advantage of experience to drive the process. Thus the YACS approach differs from ACS since its generalization process does not use Genetic Algorithms [BGS00a]. This process considers sets of classifiers with the same C and A parts and decides how to specialize C parts. The generalization process also uses estimates to drive the generalization process: the *expected improvement by generalization*.

4.1 THE *EXPECTED IMPROVEMENT BY GENERALIZATION* ESTIMATES

An *expected improvement by generalization* i_g estimate is associated to each specialized feature of a C part. It estimates if the E part of the classifier would remain accurate if the considered feature was general.

At each time step, YACS knows the current situation S_t resulting from the action A_{t-1} in the situation S_{t-1}. This information is used to compute the desired effect DE which is the E part of a classifier which could have been fired at the preceding time step, and whose E part accurately reflects the changes actually perceived in the environment.

In the effect covering process, every classifier whose C part matches S_{t-1} and whose A part matches A_{t-1} is checked. In order to compute the i_g estimates, YACS checks every classifier whose A part matches A_{t-1} and whose C part does *not* match S_{t-1}.

Considering such classifiers, for each specialized fea-

[4] a feature with a *don't care* symbol

ture of the C part, YACS checks if the C part of the classifier would match S_{t-1} if the considered feature were general. In this case, the considered i_g estimate is updated:

- If the E part of the classifier equals the desired effect DE, then a classifier with a more general C part would have an accurate E part and the considered i_g estimate is increased.
- If the E part of the classifier does not equal the desired effect DE, then a classifier with a more general C part would have an inaccurate E part and the corresponding i_g estimate is decreased.

The i_g estimates are increased and decreased according to a Widrow-Hoff delta rule. The initial values are 0.5. A general feature is given an i_g value of 0.5.

Up to that point, with this mechanism, YACS is able to check if a feature of a C part should be generalized or not.

4.2 THE GENERALIZATION PROCESS

The *expected improvement by generalization* estimates detailed above allow the classifier generalization mechanism to be driven by experience and are used in the C parts generalization process.

From one situation S_{t-1} to a new one S_t, the selected action A_{t-1} leads to some effects DE in the environment. Each time step, YACS checks if there is some possible generalization between the C parts of the classifiers such that their E part equals DE and their A part matches A_{t-1}. So, the generalization process considers sets of classifiers with the same A and E parts.

With such a set of classifiers, YACS builds a new set of classifiers which are more general or equal to the original ones. These new classifiers are such that they do not match situations already matched by classifiers with a different E part. The classifiers of the new set replace the original ones in the Classifier System.

If every estimate i_g of a classifier is lower than 0.5, it is not a good candidate for the generalization and it is added without modifications in the new set of classifiers.

In either case, a new classifier is created. A feature of the C part is generalized if its associated estimate i_g equals to the greatest among the estimates associated to the considered classifier.

The new C part may lead to a conflict with other classifiers with the same A part but a different E part. In this case, YACS does not add the new and general classifier to the new set, but the original one. A conflict is detected when two classifiers share the same A parts but have different E parts, and if at least one situation is matched by both C parts. YACS finds the possible situations in the set P of every perceived situation encountered during the lifetime of the agent (see section 2).

At this point YACS has computed a new set of classifiers such that each classifier is more general or equal to the original one, and such that none of them drives to a conflict with other classifiers in the system. The next step is to select the more general classifiers.

To do so, YACS checks iteratively every possible pair of classifiers. When the C parts of two classifiers are matching, the classifier with the smallest number of general features is removed. So the classifiers of the resulting set are not redundant.

This process allows to replace several classifiers with a smaller or equal number of classifiers. The C part of the new classifiers cover the same situations. Thus they do not drive to a conflict with other classifiers in the system.

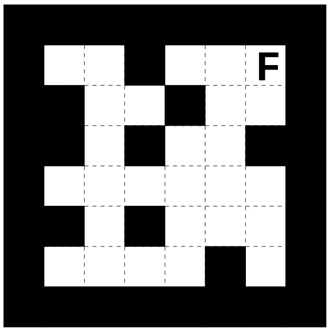

Figure 2: The Maze4 environment

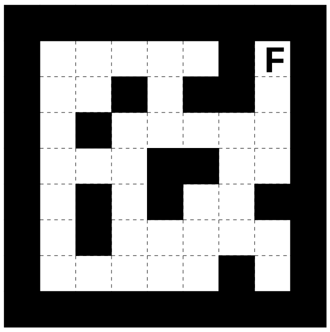

Figure 3: The Maze6 environment

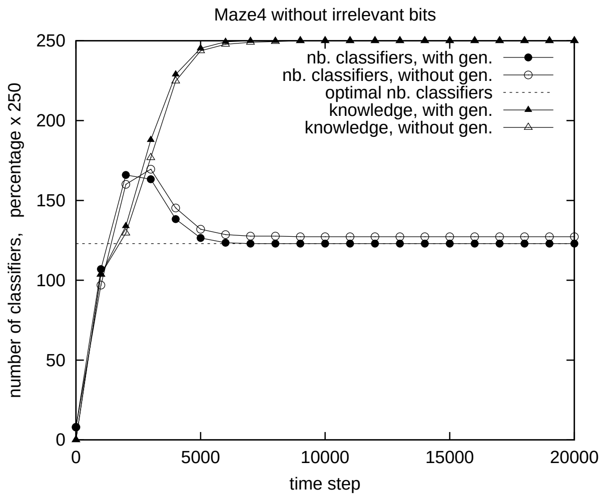

Figure 4: Evolution of the number of classifiers in Maze4

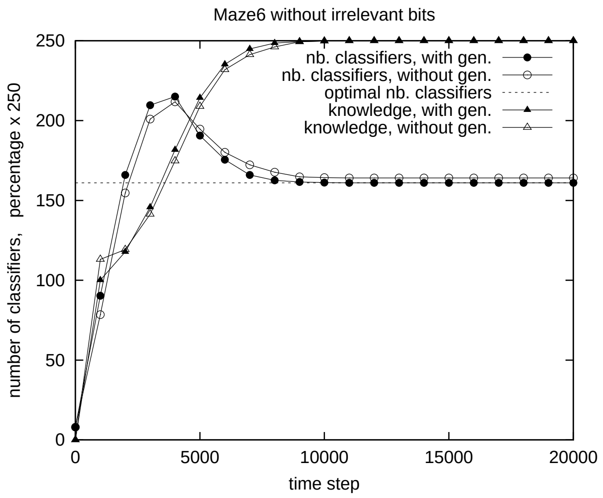

Figure 5: Evolution of the number of classifiers in Maze6

5 EXPERIMENTAL RESULTS

This section presents experimental results of YACS modeling *Wilson's woods* environments. The simulated woods environments are described in section 5.1. We show in section 5.2 how the generalization process helps the *latent learning* process of YACS to converge to the optimal number of accurate classifiers. Therefore, YACS did not learn a policy, but only a model of dynamics of the environment while moving randomly in the mazes.

5.1 THE MAZE4 AND MAZE6 WOODS ENVIRONMENTS

In woods environments, the agent is situated in a maze cell and perceives the eight adjacent cells. A cell can either be empty, or contain an obstacle ■ or food F. It can move towards any of these cells. If the agent moves towards an obstacle, it remains in the same cell.

Maze4 and Maze6 (see figures 2 and 3) have been earlier investigated by Lanzi. The experiments we present in this paper involve YACS interacting with these environments. The experiments are divided into trials. The agent starts a trial in a free cell chosen randomly. A trial ends when the agent reaches the cell with food. In that case the agent gets a reward, it gets a new perceived situation, and a new trial starts.

In these environments, it is possible to generalize the transitions which do not lead to any change. This is the case when an action leads the agent to hit an obstacle. There are respectively 93 and 135 transitions of that kind in Maze4 and Maze6 By taking advantage of generality, the transitions resulting from such actions can be modeled with 8 classifiers: one classifier for each possible action, by paying attention to the presence of a block in the direction corresponding to the action. There are no other useful regularities in Maze4 and Maze6. Since the total number of possible transitions is 208 in Maze4 and 288 in Maze6, the optimal numbers of classifiers YACS should reach are respectively 123 $(208-93+8)$ and 161 $(288-135+8)$ for Maze4 and Maze6.

Moreover, so as to provide YACS more occasions to take advantage of generalization, we add *irrelevant bits* to the perceived situations. These attributes are randomly set between 0 and 1 when a new trial starts and keep the same value during the whole trial. For a system without any generalization capability this would result in new perceived situations. But as the added perceived features are irrelevant to distinguish between situations, the optimal number of classifiers remains the same when irrelevant bits are added.

5.2 EXPERIMENTS IN MAZE4 AND MAZE6

In order to estimate the evolution of the accuracy of the model over successive time steps, we use a measure of the *percentage of knowledge* provided by the model. For each possible transition in the environment, we check if the classifier system is able to model accurately the transition. The percentage of knowledge is the ratio of possible transitions covered by *reliable* classifiers only. The memory size m is set to 5 and

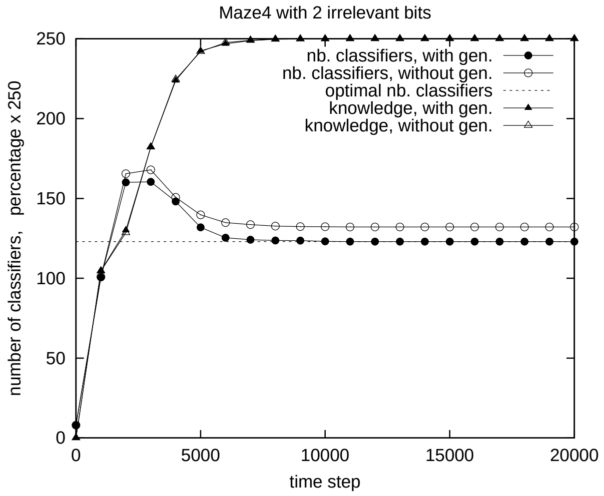

Figure 6: Evolution of the number of classifiers in Maze4 with 2 irrelevant bits

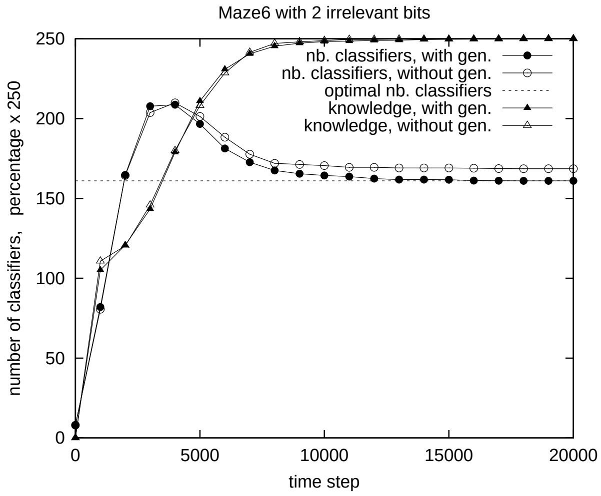

Figure 7: Evolution of the number of classifiers in Maze6 with 2 irrelevant bits

the learning rates are set to 0.1. All the results are averaged over 10 experiments.

Figure 4 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments with 0 irrelevant features. The experimental results are shown for YACS running with and without the generalization process. Figure 5 shows the same information for the experiments with the Maze6 environment. Without generalization, the average number of classifiers discovered by YACS converges towards 127.3 (4.3 more than optimum) for Maze4 and 164.1 (3.1 more than optimum) for Maze6. This number is only near-optimal and with the generalization process enabled, the number of classifiers converges to the optimum (123 for Maze4 and 161 for Maze6, see section 5.1). Even if in Maze6, there are around 40% more transitions to model than in Maze4, these figures show that YACS does not need much more time to converge towards an optimal model of the dynamics of the environment.

During the first part of the learning process, YACS mostly creates new classifiers and their number is growing. During the second part, because the actions are selected at random, YACS may take time to experiment every possible transition as many times as necessary to remove inaccurate classifiers. As the memory size m is reduced, YACS converges faster because it takes less time to remove inaccurate classifiers, but the maximum number of classifiers during the learning process gets higher. So as to speed up the convergence towards an optimal model, we could use exploration bonuses as in [SB98]. The benefits drawn would be larger as the environment is more complex.

Figure 6 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments when 2 irrelevant bits are added. Figure 7 shows the same in the Maze6 environment.

Without generalization, the average number of classifiers discovered by YACS converges towards 132.1 for Maze4 (9.1 more than optimum) and 168.6 for Maze6 (7.6 more than optimum). With the generalization process enabled, the number of classifiers converges to the optimum. Without generalization, the difference between the number of discovered classifiers and the optimum is greater with irrelevant bits. In this case, YACS sometimes specializes according to these bits because the estimates are not absolutely reliable, especially in the case of partial exploration of the situation space.

The results with generalization show that this process is able to reconsider early specialization mistakes without modifying a lot the learning speed. This way, YACS models the environment with a smaller number of classifiers than ACS [BGS00b] does.

6 CONCLUSION AND FUTURE WORK

The latent learning process builds a model of the dynamics of the environment even in the absence of rewards. It models how the actions modify the perceived situations. This modeling process uses information about successive perceived situations. The information used is available at each time step. So, latent learning systems make an intensive use of the perceptual feedback offered by the sensori-motor loop. Thus, they can quickly identify relevant and general classifiers without using Genetic Algorithms.

In this paper, we briefly described the main mechanisms of the *latent learning* in YACS and we proposed a new way for performing generalization. We have shown experimentally that this additional process is able to overcome the over-specialization problems occurring in previous versions of YACS.

However, YACS is still bounded to deterministic Markov problems. In a middle term, YACS should be enhanced to tackle non-Markov problems. Moreover, we will explore in a short term a new formalism which allows to express more regularities of the environment.

References

[BGS00a] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part i: Theoretical approach. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, 2000.

[BGS00b] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Introducing a genetic generalization pressure to the Anticipatory Classifier System part ii: Experimental results. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference (GECCO 2000)*, 2000.

[Dor94] M. Dorigo. Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation*, 1(2):151–164, 1994.

[GSS01] P. Gérard, W. Stolzmann, and O. Sigaud. YACS : a new Learning Classifier System using Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, (to appear) 2001.

[Hol76] J.H. Holland. Adaptation. *Progress in theorical biology*, 1976.

[Hol90] J.H. Holland. Concerning the emergence of tag mediated lookahead in Classifier Systems. *Special Issue of Physica D*, 42:188–201, 1990.

[Lan97] P. L. Lanzi. A study of the generalization capabilities of XSC. In T. Baeck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 418–425, San Franciso, California, 1997. Morgan Kaufmann.

[Rio91] R. L. Riolo. Lookahead planning and latent learning in a Classifier System. In J.-A. Meyer and S. W. Wilson, editors, *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior*, pages 316–326. MIT Press, 1991.

[SB98] R. S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[SG01] O. Sigaud and P. Gérard. Being Reactive by Exchanging Roles: an Empirical Study. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *LNCS : Balancing reactivity and Social Deliberation in Multiagent Systems.* Springer-Verlag, (to appear) 2001.

[Sto98] W. Stolzmann. Anticipatory Classifier Systems. In J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[Sut91] R.S. Sutton. Reinforcement learning architectures for animats. In J.-A. Meyer and S. W. Wilson, editors, *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptative Behavior*, Cambridge, MA, 1991. MIT Press.

[TB00] A. Tomlinson and L. Bull. CXCS. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 194–208. Springer-Verlag, Heidelberg, 2000.

[Wat89] C. J. Watkins. *Learning with delayed rewards.* PhD thesis, Psychology Department, University of Cambridge, England, 1989.

[Wil95] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.