Designing Efficient Exploration with MACS: Modules and Function Approximation

Pierre Gérard and Olivier Sigaud

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

Abstract. MACS (Modular Anticipatory Classifier System) is a new Anticipatory Classifier System. With respect to its predecessors, ACS, ACS2 and YACS, the latent learning process in MACS is able to take advantage of new regularities. Instead of anticipating all attributes of the perceived situations in the same classifier, MACS only anticipates one attribute per classifier. In this paper we describe how the model of the environment represented by the classifiers can be used to perform active exploration and how this exploration policy is aggregated with the exploitation policy. The architecture is validated experimentally. Then we draw more general principles from the architectural choices giving rise to MACS. We show that building a model of the environment can be seen as a function approximation problem which can be solved with Anticipatory Classifier Systems such as MACS, but also with accuracy-based systems like XCS or XCSF, organized into a Dyna architecture.

1 Introduction

Research on Learning Classifier Systems (LCSs) has received increasing attention over the last few years. This surge of interest has concretized itself in two different directions.

First, a trend called "classical LCSs" hereafter comes from the simplification of Holland's initial framework [Hol85] by Wilson. The design of ZCS [Wil94] and then XCS [Wil95] resulted in a dramatic increase of LCS performance and applicability. The latter system, using the accuracy of the reward prediction as a fitness measure, has proven its effectiveness on different classes of problems such as adaptive behavior learning or data mining. XCS can be considered as the starting point of most new work along this first line of research.

Second, a new family of systems called Anticipatory Learning Classifier Systems (ALCSs) has emerged, showing the feasibility of using model-based Reinforcement Learning (RL) in the LCS framework. This second line of research is more inclined to use heuristics rather than genetic algorithms (GAs) in order to deal with the improvement of classifiers. Several systems (*e.g.* ACS [Sto98,Sto00], ACS2 [But02a] and YACS [GSS02]) have highlighted the interesting properties of this family of approaches.

The way ALCSs achieve model-based RL consists in a major shift in the classifiers representation. Instead of `[condition] [action]` classifiers, they use a `[condition] [action] [effect]` representation, where the `[effect]` part represent what would result from taking the action if the condition is verified. As a consequence of this representational shift, the ALCS framework is somewhat distinct from the classical LCS one.

In this paper, we want to show that one can benefit from the model-based properties of ALCSs while keeping the classical LCS framework as is, thanks to the design of a general architecture whose basic components can be either ALCSs or classical LCSs like XCS or XCSF [Wil01,Wil02], or even other kinds of function approximators.

More precisely, after presenting ALCSs in section 2, we will introduce in section 3 a new ALCS called MACS, whose generalization properties are different from those of previous ALCSs. Classifiers in MACS are only intended to predict one attribute of the next situation in the environment depending on a situation and an action, whereas classifiers in all previous systems try to predict the next situation as a whole. We will show that this new representation gives rise to more powerful generalization than previous ones, and can be realized with a modular approach. Then we will explain in section 4 how such an efficient model-based learning process can be integrated into a Dyna architecture combining exploration and exploitation criteria, giving empirical results in section 5.

Then, in section 6, we will generalize what we have learned from MACS in a wider perspective. Predicting the value of one attribute of the environment can be seen as a function approximation problem, and any system able to approximate a function can be used in the same architecture. In particular, since XCS and XCSF are such systems, we will conclude in section 7 that model-based reinforcement learning with generalization properties can be performed as well with XCS, XCSF or even other kinds of systems, provided that an adequate architecture is used.

2 Anticipatory Learning Classifier Systems

The usual formal representation of RL problems is a Markov Decision Process (MDP) which is defined by:

- a finite state space S;
- a finite set of actions A;
- a transition function $T : S \times A \rightarrow \Pi(S)$ where $\Pi(S)$ is a distribution of probabilities over the state space S;
- a reward function $R : S \times A \times S \rightarrow \mathbb{R}$ which associates an immediate reward to every possible transition.

One of the most popular RL algorithm based on this representation is Q-learning [Wat89]. This algorithm directly and incrementally updates a Q-table representing a quality function $q : S \times A \rightarrow \mathbb{R}$, without using the transition and the immediate reward functions. The quality $q(s, a)$ represents the expected payoff when the agent performs the action a in the state s, and follows the greedy policy thereafter. Then, the qualities aggregate the immediate and future expected payoffs.

The main advantage of Learning Classifier Systems with respect to other RL techniques like tabular *Q-learning* relies in their generalization capabilities. In problems such that situations are composed of several attributes, generalization makes it possible to aggregate several situations within a common description so that the model of the quality function q becomes smaller.

In [Lan00], Lanzi shows how it is possible to shift from a tabular representation of a RL problem to a classifier-based representation. While tabular Q-learning considers triples $(s, a, q) \in S \times A \times \mathbb{R}$, LCSs like XCS consider `C-A-p` rules `[condition] [action] payoff` *classifiers*). During the learning process, the LCS learns appropriate general conditions and updates the payoff prediction.

Within the classical LCS framework, the use of *don't care* symbols `#` in the `C` part of the classifiers permits generalization, since *don't care* symbols make it possible to use a single description to describe several situations. Indeed, a *don't care* symbol *matches* any particular value of the considered attribute. Therefore, changing an attribute into a *don't care* symbol makes the corresponding condition more general (it matches more situations). The main issue with generalization in classical LCSs is to organize `C` and `A` parts so that the *don't care* symbols are well placed.

XCS offers a generalization capability but, as *Q-learning*, it can only update very few measures of classifier payoff prediction at each time step, corresponding to the immediate actual previous situation and action s_{t-1} and a_{t-1}. Indeed, the model of the expected payoff can only be updated when an actual transition occurs. Sutton [Sut90] proposed the Dyna architecture to endow the system with the ability to update many qualities at the same time, in order to significantly improve the learning speed of the quality function. The Dyna architecture illustrated in figure 1 uses a model of the environment to build hypothetical transitions independently from the current experience. These simulated actions are used to update the model of the quality function more than once per time step, with a *value iteration* algorithm inspired from *Dynamic Programming*. The model of the environment is learned *latently* – i.e. independently from the reward.

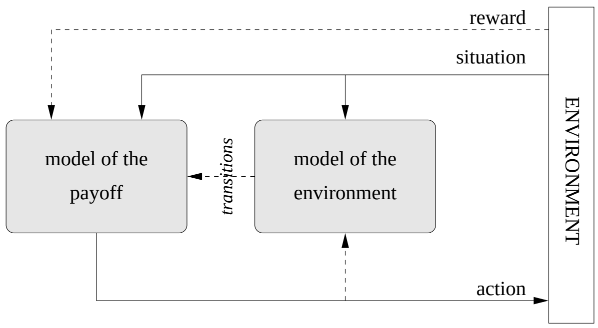

Fig. 1. A Dyna architecture to perform reinforcement learning. It combines a model of the payoff with a model of the environment. The model of the payoff may consist in an approximation of the immediate reward function $R : S \times A \rightarrow \mathbb{R}$, and in an approximation of the quality function $q : S \times A \rightarrow \mathbb{R}$, which produces a scalar quality for each $(situation, action)$ pair. To learn the function q incrementally, the system is given $(s_{t-1}, a_{t-1}, s_t, r_t)$ tuples. The model of the environment is an approximation of the transition function $T : S \times A \rightarrow S$. The transition function and the immediate reward functions provide hypothetical samples to the quality learning system, and subsequently speed up the reinforcement learning process.

Instead of directly learning a model of the quality function q as XCS does, ALCSs such as ACS [Sto98,BGS00], ACS2 [But02a] and YACS [GS01,GSS02] learn a model of the transition function T. They take advantage of generalization capabilities to learn a model of the transition function which is more compact than the exhaustive list of all the (s_t, a_t, s_{t+1}) transitions. The transition function provides a model of the dynamics of the interactions between the agent and its environment which takes place in a Dyna architecture to speed up the plain reinforcement learning process.

In ALCSs, the classifiers are organized into `[condition] [action] [effect]` parts, noted `C-A-E`. In such classifiers, the `E` part represents the effects of action `A` in situations matched by condition `C`. It records the perceived changes in the environment. In ACS, ACS2 and YACS, a `C` part is a situation which may contain *don't care* symbols `#` or specific values (like 0 or 1), as in XCS. An `E` part is also divided into several attributes and may contain either specific values or *don't change* symbols `=`. Such a *don't change* symbol means that the attribute of the perceived situation it refers to remains unchanged when action `A` is performed. A specific value in the `E` part means that the value of the corresponding attribute *changes* to the value specified in that `E` part.

This formalism permits the representation of regularities in the interactions with the environment, like for instance *"In a grid world, when the agent perceives a wall in front of itself, whatever the other features of the current cell are, trying to move forward entails hitting the wall, and no change will be perceived in the cell's features"*.

The latent learning process is in charge of discovering `C-A-E` classifiers with general `C` parts that accurately model the dynamics of the environment. ACS and YACS generalize according to anticipated situations, and not according to the payoff, as in XCS. As a result, it does not make sense to store information about the expected payoff in the classifiers. Therefore, the list of classifiers only models environmental changes. The information concerning the payoffs must be stored separately.

3 Improved Latent Learning with MACS

3.1 Representing More Regularities with MACS

Generalization makes it possible to represent regularities in the interactions with the environment. However, while ACS and YACS are able to detect if a particular attribute is changing or not, their formalism cannot represent regularities across different attributes because it considers each situation as a whole. To make this point clear, let us consider an agent in a grid world such as those presented in figures 3, 4 and 5, where its perceptions are defined as a vector of boolean values depending on the presence or absence of walls in the eight surrounding cells. Turning right results in a two-positions left shift of the attributes. For instance, the agent may experience transitions like `[11001100]` [$\curvearrowright$] `[00110011]`.

In such a case, every attribute is changing. Thus, the formalism of ACS and YACS is unable to represent this regularity. Nevertheless, the shift in the perceived situation is actually a regularity of the dynamics of the interactions: whatever the situation is, when the agent turns clockwise, the value of the 1st attribute comes to the last value of the 3rd, the value of the 2nd becomes the 4th *etc.*

The particularity of such a regularity is that the new value of an attribute depends on the previous value of another one. Expressing generalization with *don't change* symbols forbids the representation of such regularities. In the ACS/YACS formalism, the new value of an attribute may only depend upon the previous value of the same attribute, a situation which is seldom encountered in practice. To overcome this problem, it is necessary to decorrelate the attributes in the `E` parts, whereas ACS and YACS classifiers anticipate all attributes at once.

To this end, our new system, MACS, describes the `E` parts with *don't know* symbols "`?`" rather than with *don't change* symbols. This way, the accurate classifier `[####1###]` [$\curvearrowright$] `[??1?????]` means that "*just after turning right, the agent always*

[11001100]		$\leftarrow$*Situation*
[1#######]	[$\curvearrowright$]	[??????1?]
[#1######]	[$\curvearrowright$]	[???????1]
[##0#####]	[$\curvearrowright$]	[0???????]
[###0####]	[$\curvearrowright$]	[?0??????]
[####1###]	[$\curvearrowright$]	[??1?????]
[#####1##]	[$\curvearrowright$]	[???1????]
[######0#]	[$\curvearrowright$]	[????0???]
[#######0]	[$\curvearrowright$]	[?????0??]
[#######0]	[$\curvearrowright$]	[?????1??]
Anticipations $\rightarrow$		[00110011]
	or	[00110111]

Table 1. During the integration process, the LCS proposed in section 3 scans the `E` parts and selects classifiers whose `A` parts match the action and whose `C` part match the situation. The integration process builds all the possible anticipated situations with respect to the possible values of every attribute. Here, the system anticipates that using `[11001100]` as a current situation should lead either to `[00110011]` or to `[00110111]`. In deterministic environments, if all the classifiers were accurate, this process would generate only one possible anticipation.

perceives a wall at its left when it perceived a wall behind itself, whatever the other attributes were". This classifier does not provide information about the new values of other attributes (as denoted by the `?` symbol). Thus, the overall system gains the opportunity to discover regularities involving different attributes in the `[condition]` and the `[effect]` parts.

Again, this proposal for a new formalism leads to a new conception of generalization. As usual, a classifier is said to be *maximally general* if it could not contain any additional *don't care* symbol without becoming inaccurate. But it is now said to be *accurate* if, in every situation matched by its condition, taking the proposed action always sets the attributes to the values specified in the effect part, when such attributes are not *don't know* symbols.

As a result, the anticipating unit is not the single classifier anymore but the whole LCS. Given a situation and an action, a single classifier is not able to predict the whole next situation: it anticipates only one attribute. The system needs an additional mechanism which *integrates* these partial anticipations and builds a whole anticipated situation, without any *don't know* symbol in its description, as shown in Table 1.

Experimental results presented in [GMS03] demonstrated that the new formalism used by MACS actually affords more powerful generalization capacities than the formalism of YACS, without any cost in terms of learning speed.

The algorithms realizing the latent learning process in MACS will not be described in detail here, they are presented in [GMS03].

3.2 Modular Model of the Environment with MACS

In the previous section, we described how MACS represents its model of the dynamics of the environment with anticipating classifiers.

In all Dyna systems, this model consists of an exhaustive list of (s_{t-1}, a_{t-1}, s_t) triples, each specifying a whole transition, i.e. the expected value of a complete set of attributes. ACS and YACS both improve the model by adding generalization in the triples, but each classifier still specifies complete transitions. Conversely, in MACS, each classifier only provides a prediction concerning one attribute.

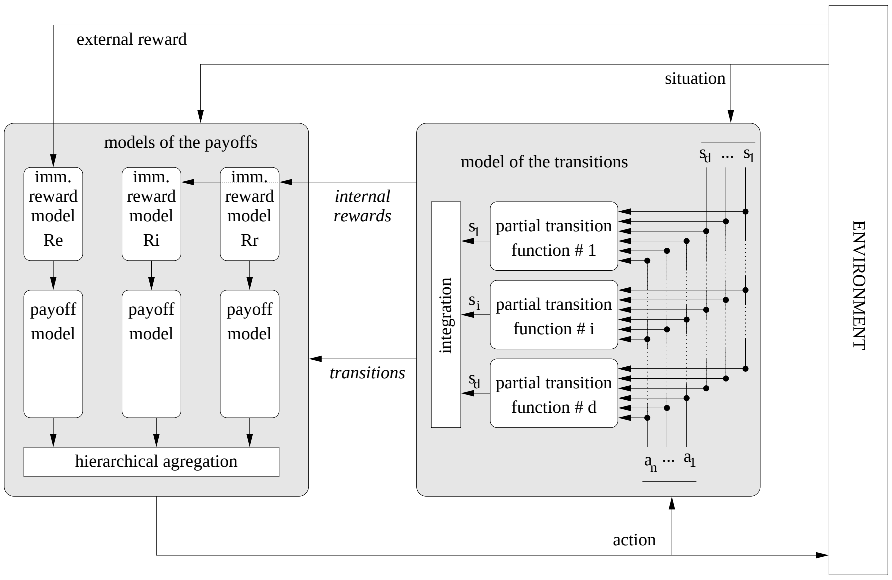

Fig. 2. MACS architecture. Each module is a simple function approximator.

Due to conditions, each classifier of YACS (or (s_{t-1}, a_{t-1}, s_t) triple of DynaQ+) is a subfunction of the global transition function $T\ : s_1 \times ... \times s_d \times a_1 \times ... \times a_e \rightarrow s_1 \times ... \times s_d$[1]. Conversely, in MACS, each classifier is a subfunction of a partial transition function $T_i\ : s_1 \times ... \times s_d \times a_1 \times ... \times a_e \rightarrow s_i$. Then, it is possible to consider groups of classifiers, each group anticipating one particular attribute. Each of these groups then models a partial transition function. The global transition function can be obtained by integrating the partial functions.

The MACS architecture illustrated in figure 2 shows how the latent learning part of MACS can be considered as a modular system, each module anticipating one attribute. Each of these modules provides an approximation of one partial transition function, each predicting one single value. As we will discuss in section 7, this architecture suggests that one could replace MACS modules by some other function approximation systems like neural networks or classical LCSs like XCS or XCSF.

4 Combining Active Exploration and Exploitation

4.1 Hierarchical Aggregation of Different Criteria

The aim of active exploration is to provide the agent with a policy that maximizes the information provided by the sensori-motor loop. The agent selects actions that help improving the model according to this criterion.

[1] where e is the number of effectors, and d is the number of perceived attributes

As illustrated by figure 2, in order to combine active exploration with exploitation, we designed an architecture resulting from the hierarchical combination of three dynamic programming modules, each trying to maximize the reward from a different source. Indeed, we distinguish the internal reward, corresponding to a gain in information about the model of the environment, the rehearsal reward, corresponding to a measure of the time elapsed after the last visit of each transition, and the external reward, corresponding to a gain of food or whatever actual reward in the environment of the agent.

The precise definition of these immediate rewards are the following

– The general idea of defining an immediate information reward consists in measuring whether the model of the transitions can be improved or not thanks to the activation of a particular action.
  More precisely, we define an estimator $El(c)$ associated with each classifier c. $El(c)$ measures the evaluation level of the classifier. Thus $El(c)$ must be equal to 0 if the classifier has not been evaluated yet, and must be equal to 1 if the classifier has been tested enough.
  Thus we define $El(c) = min((b+g)/\theta_e, 1)$, where θ_e is the number of evaluations needed to classify a classifier as accurate, inaccurate or oscillating, and b and g are respectively the number of anticipation mistakes and successes already encountered by the classifier in previous evaluations.
  Each estimator $El(c)$ is bound to one classifier, not to one situation. In order to compute the information gain bound to one situation $R_i(s_0)$, the process is the following.
  The classifiers that match s_0 are grouped by action. For each possible action a, MACS computes the set $S_{s_0,a}$ of the possible anticipated situations as described in section 3[2]. Each triple (s_0, a, s_1), where $s_1 \in S_{s_0,a}$, is one of the possible transitions that would be experienced if action a were performed in situation s_0. We define the evaluation level $El(s_0, a, s_1)$ associated with this transition as the product of the evaluation levels $El(c)$ of all the classifiers c involved in this anticipation:

$$El(s_0, a, s_1) = \prod_{c \approx (s_0, a, s_1)} El(c)$$

  The classifiers c matching (s_0, a, s_1) are such that their C part matches s_0, their A part matches a, and their E part matches s_1. The less a transition has been evaluated, the greater the immediate information gain. Thus, if the transition occurs, the associated immediate information gain is:

$$R_i(s_0, a, s_1) = 1 - El(s_0, a, s_1)$$

  We define the immediate information gain associated with a situation and an action as the maximum information gain over the possible associated anticipations:

$$R_i(s_0, a) = \max_{s1 \in S_{s_0,a}} R_i(s_0, a, s_1)$$

  If the model does not provide MACS with at least one anticipated situation s_1, because of incompleteness, then $R_i(s_0, a)$ is given the default value 1, which is the maximum immediate information gain.

[2] There may be several possible anticipated situations in the case where the classifiers are not accurate.

Finally,

$$R_i(s_0) = \max_{a \in A} R_i(s_0, a)$$

- The external reward is the usual source of reward found in any reinforcement learning framework. We define it as $Re(s_0)$ corresponding to the immediate reward obtained for visiting the situation s_0.
- The immediate rehearsal reward leads to a policy which is similar to the exploration policy described in [Sut91] or [But02b][3]. Then it grows until the situation is visited again and then drops to 0.
  In order to update $R_r(s_0)$, we use the classical Widrow-Hoff equation:

$$R_r(s_0) = (1 - \beta_r)R_r(s_0) + \beta_r$$

Then, from each immediate reward, whether internal, rehearsal or external, a long term expected payoff is computed separately thanks to the *Value Iteration* algorithm (see [SB98] for a presentation). The latent learning process provides the system with the model of the transitions which is necessary for an offline computing of the qualities associated to each action, given a situation. By doing so, the values are updated independently from the actual experience of the agent, and many updates can be computed at each time step, as usual in a *Dyna* architecture.

The last component of the architecture consists of a hierarchical combination of these three criteria. Since an inaccurate model may result in an inaccurate estimation of the expected external payoff, maximizing the information gain is given the priority against the external payoff maximization. Thus, if there is a better action with respect to the information gain criterion, this action is chosen. Else, if at least two actions provide the maximal expected information gain, then the one which maximizes the external payoff is chosen. In particular, if the information about the problem is perfect, which means that no action provide any gain in information anymore, then the policy will be completely driven by the external payoff and will converge to optimal exploitation. The last criterion, rehearsal reward, is the least often used. It is only chosen if at least two actions are equally likely to be fired with respect to both previous criteria.

5 Experimental Results: Moving Sources of Reward

In order to illustrate the gain resulting from latent learning in reinforcement learning problems, we present in this paper new experiments where we tried to test MACS on problems where the sources of reward are moved after the agent succeeds in reaching them.

This problem gives the opportunity to highlight two key properties of MACS. First, the necessity to have an internal model and to be able to perform *Value Iteration* on that model as an offline mental rehearsal arises when the agent discovers that the source of reward has moved: it must forget all the payoff estimates corresponding to its previous model, and this forgetting process would be very slow without such a model. Second, the necessity of active exploration arises once the model is forgotten: the agent must re-explore the whole environment to find the new location of the source of reward. This search is much more efficient thanks to active exploration.

[3] The latter also uses a sort of "internal reward" biasing action selection according to the accuracy or "quality" of the predicted effects.

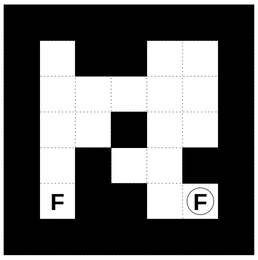

Fig. 3. Maze216C

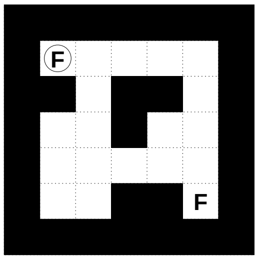

Fig. 4. Maze228C

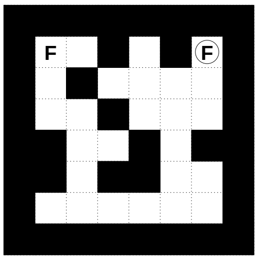

Fig. 5. Maze312C

As a benchmark problem, we tested MACS on three different environments, namely Maze216C, Maze228C and Maze312C, respectively shown on figure 3, 4 and 5. These "maze-like" or "woods" problems are standard benchmarks in LCS research, they could be replaced by any finite state automaton but they provide a more intuitive view.

At the beginning of the experiments, the food is in the cell marked with a circled F. Once the agent has reached it twenty times, the food is moved to the cell marked with a plain F. This is done repeatedly, the food being moved again each time the agent has found it twenty times.

Figure 6 illustrates the number of time steps MACS needs to solve Maze216C, Maze228C and Maze312C during 250 successive trials. The results are averaged over 100 experiments. The learning rates are set to 0.1 and the memory size and number of evaluations necessary to take a specialization/generalization decision are all set to 5. The discount factor γ is set to 0.9.

The results show that, though it only performs one *Value Iteration* step per time step, MACS is able to re-adapt the policy to the new source location very fast. Each twenty trials, the longer trial corresponds to the case where the agent must find the new location of the source of reward. It is found fast thanks to active exploration and, as soon as it is found, the time necessary to reach it again converges during the next trial. The slight variations are due to the fact that the agent always starts from a random cell and the results are averaged over 100 trials only.

Note that MACS is tested here in a deterministic environment and would not perform as efficiently in a probabilistic environment, but the lessons that we will draw in the following discussion and conclusion still apply in the probabilistic case.

6 Discussion

In this paper, we have shown how a latent learning process could be combined with several dynamical programming processes into a Dyna-like architecture. As in Dyna architectures, one module learns a model of the environment and the other modules are in charge of maximizing the payoff expectation thanks to offline Dynamic Programming techniques. By using several immediate rewards, we have shown how three different policies could be obtained and how a hierarchical combination of these policies resulted in improved performance in classical problems considered difficult in the reinforcement learning framework.

A finer decomposition into modules can be obtained if one considers that the latent learning process in MACS can itself be split into one module per anticipated attribute. The global architecture of our system is shown in figure 2.

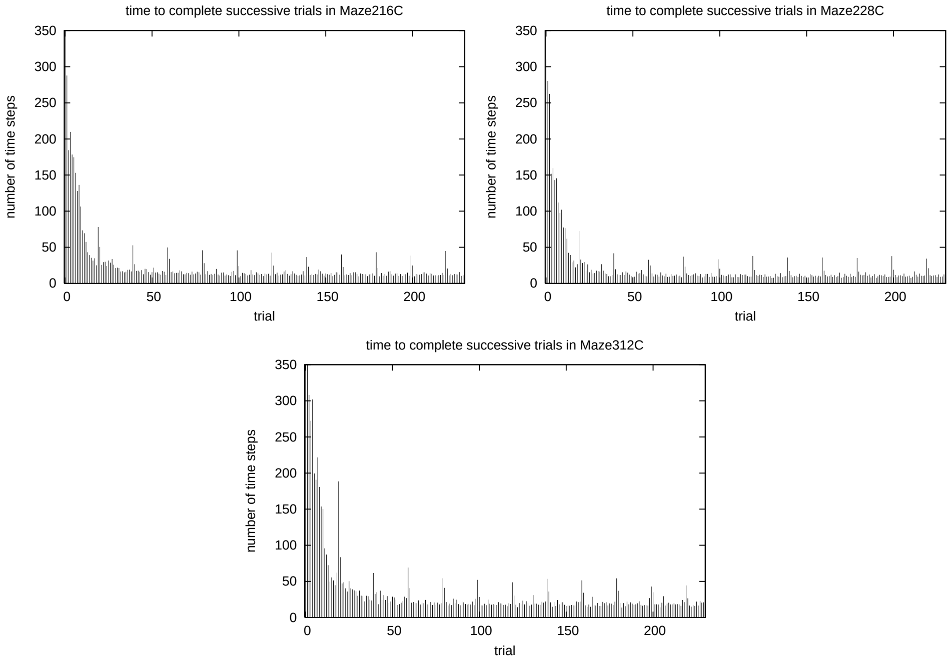

Fig. 6. MACS combining exploration and exploitation: number of time steps to reach the source of reward in successive trials in Maze216C, Maze228C and Maze312C.

As in DynaQ+, one policy relies on the external reward and a second one on the time elapsed after the last visit of each transition. But the third policy in MACS, relying on the information which can be gained by firing each classifier, is more original. It provides the agent with a systematic exploration capability which allows it to gain a perfect knowledge of its environment much faster than with a random exploration, as shown in [GMS03].

The hierarchical combination of these policies is also a distinctive feature of MACS with respect to DynaQ+. In DynaQ+, a weighted sum of the different criteria results in a compromise between the different policies. First, designing a weighted sum of different criteria while the magnitude of these criteria is not known in advance (it depends on the amount of external reward, which may vary from an environment to another) is very difficult. Thus it is likely that the weights have to be tuned for each experiment. In MACS, on the contrary, no weight parameter has to be tuned in order to deal with different levels of reward, since such weights do not exist in the hierarchical agregation.

Furthermore, since the criterion corresponding to the tendency to explore is never equal to zero, the overall behavior is always under-optimal with respect to payoff. In MACS, by contrast, giving the priority to building a correct model permits the construction of a complete model, and then the payoff maximization process naturally takes the control of the agent and gives rise to an optimal behavior.

7 Future Work and Conclusion

In the MACS architecture for latent learning presented on figure 2, the functional role of each module is to guess the value of one attribute at the next time step given the value of several attributes (conditions and actions) at the current time step. That is, each module tries to approximate a *many-to-one function*, whereas classifiers in ACS and YACS were trying to approximate a *many-to-many mapping*.

The effectiveness of this approximation results from the fact that all classifiers receive at the next time step the actual situation or attribute that they should have anticipated. I.e., though there is no supervisor, the learning process can benefit from informations on its errors as if it were supervised.

Therefore, any system relying on the same property can be used to approximate functions with the same effectiveness. In particular, XCS is such a system, since the prediction of the accuracy of each eligible classifier can be corrected after each time step.

Thus, rather than using a specialized ALCS such as MACS to solve reinforcement learning problems with a model-based approach, it is possible to use a more general system like XCS as a building block to implement the basic modules in the architecture presented in figure 2.

Using XCS in such an architecture would let the system solve discrete state and time problems verifying the Markov property as MACS does. Another possibility would be to use XCSF [Wil01,Wil02], an XCS extension devoted to the piecewise linear approximation of continuous functions, so as to address continuous state problems.

Our future work will consist in the investigation of that line of research : using XCSF modules to learn both a policy and a model of the environment. It must be mentioned that, where XCSF uses the Widrow-Hoff delta rule so as to linearly approximate a function, we envision to rather use a more powerful and efficient method coming from the field of incremental statistical estimations.

As a conclusion, we described in this paper several LCSs, casting a new light on the concept of generalization in the LCS framework. We presented MACS, a new LCS using a different formalism able to use additional regularities in its latent learning process. MACS formalism for latent learning does not consider situations as an unsecable whole, but decorrelates the attributes, making it possible to represent regularities across attributes. We have illustrated the effectiveness of our approach with MACS in the context of an active exploration problem, using a hierarchical aggregation criterion in order to tackle the exploration/exploitation tradeoff.

Beyond the presentation of MACS, we hope to have convinced the reader that solving reinforcement learning problems with a model-based approach can be seen as a conjunction of several function approximation problems, and that solving these problems can be achieved efficiently with different systems as a building block, given that they rely on a prediction process.

References

[BGS00] 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)*, pages 34–41, 2000.

[But02a] M. V. Butz. An Algorithmic Description of ACS2. In Lanzi et al. [LSW02], pages 211–229.

[But02b] M. V. Butz. Biasing Exploration in an Anticipatory Learning Classifier System. In Lanzi et al. [LSW02], pages 3–22.

[GMS03] P. Gérard, J.-A. Meyer, and O. Sigaud. Combining latent learning with dynamic programming. *European Journal of Operation Research*, to appear, 2003.

[GS01] P. Gérard and O. Sigaud. YACS : Combining Anticipation and Dynamic Programming in Classifier Systems. In P. L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 52–69. Springer-Verlag, Berlin, 2001.

[GSS02] P. Gérard, W. Stolzmann, and O. Sigaud. YACS: a new Learning Classifier System with Anticipation. *Journal of Soft Computing : Special Issue on Learning Classifier Systems*, 6(3-4):216–228, 2002.

[Hol85] J.H. Holland. Properties of the bucket brigade algorithm. In J.J. Grefenstette, editor, *Proceedings of the 1st international Conference on Genetic Algorithms and their applications (ICGA85)*, pages 1–7. L.E. Associates, july 1985.

[Lan00] P. L. Lanzi. Learning Classifier Systems from a reinforcement learning perspective. Technical Report 00-03, Dip. di Elettronica e Informazione, Politecnico di Milano, 2000.

[LSW02] P. L. Lanzi, W. Stolzmann, and S.W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 2321 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, 2002.

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

[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*, pages 658–664. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1998.

[Sto00] W. Stolzmann. An introduction to Anticipatory Classifier Systems. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 175–194. Springer-Verlag, Heidelberg, 2000.

[Sut90] R. S. Sutton. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning ICML'90*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

[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*, pages 288–296, Cambridge, MA, 1991. MIT Press.

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

[Wil94] S. W. Wilson. ZCS, a zeroth level Classifier System. *Evolutionary Computation*, 2(1):1–18, 1994.

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

[Wil01] S. W. Wilson. Function approximation with a classifier system. In L. Spector, Goodman E. D., A. Wu, W. B. Langdon, H. M. Voigt, and M. Gen, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO01)*, pages 974–981. Morgan Kaufmann, 2001.

[Wil02] S. W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1(2-3):211–234, 2002.