Combining Latent Learning with Dynamic Programming in the Modular Anticipatory Classifier System

Pierre Gérard [*,**], Jean-Arcady Meyer [*], Olivier Sigaud [*]

[*] LIP6, AnimatLab
8, Rue du Capitaine Scott
75015 Paris

[**] Dassault Aviation, DGT/DPR/DESA
78, Quai Marcel Dassault
92552 St-Cloud Cedex

Abstract

Learning Classifier Systems (LCS) are rule based Reinforcement Learning (RL) systems which use a generalization capability. In this paper, we highlight the differences between two kinds of LCSs. Some are used to directly perform RL while others latently learn a model of the interactions between the agent and its environment. Such a model can be used to speed up the core RL process. Thus, these two kinds of learning processes are complementary. We show here how the notion of generalization differs depending on whether the system anticipates (like ACS, Anticipatory Classifier System and YACS, Yet Another Classifier System) or not (like XCS). Moreover, we show some limitations of the formalism common to ACS and YACS, and propose a new system, called MACS (Modular Anticipatory Classifier System), which allows the latent learning process to take advantage of new regularities. We describe how the model can be used to perform active exploration and how this exploration may be aggregated with the policy resulting from the reinforcement learning process. The different algorithms are validated experimentally and some limitations in presence of uncertainties are highlighted.

1 Introduction

The Reinforcement Learning (RL) framework [KLM96, SB98] considers adaptive agents involved in a sensori-motor loop (see figure 1). Such agents perceive situations through their sensors, and use these perceptions to select the action they will perform in the environment. As a result of their action, the agents receive a scalar reward from the environment and they perceive a new situation. The task of the agents is to learn the optimal policy – *i.e.* how to act in every situation in order to maximize the cumulative reward on the long run – in an unknown environment.

This classical Situated Artificial Intelligence problem is an optimization problem whose formal foundations are drawn from Dynamic Programming [Bel57] and which addresses several issues in the field of Operational Research. In particular, learning incrementally how to act according to perceptions is a particular classification problem which can be solved by local search algorithms like Genetic Algorithms (GAs) as evidenced, for instance, by the application of Learning Classifier Systems (LCSs) to Data Mining problems [BXM01]. Other connections between Situated Artificial Intelligence and Operational Research concern Multi Criteria Decision Problem [RV81, GSH99], when the agent has to select actions giving rise to different kinds of rewards.

The originality of Reinforcement Learning with respect to other Artificial Intelligence learning techniques is that the agent has to improve its behavior by drawing information from its interactions with the environment, without being explicitly taught what to do by an external teacher. In this framework, the learning process cannot rely on any tagged sample dataset. On the contrary, the agent must learn in an incremental way, by taking into account the information provided by its sensors along its actions. In particular, the necessity to adapt to a possibly changing environment prevents it from relying too much on the memory of its past experience.

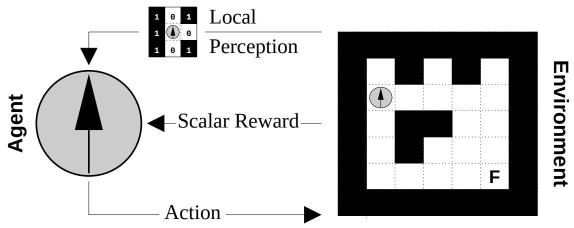

Figure 1: In Reinforcement Learning (RL) problems, the agent is situated in an *a priori* unknown environment. At every time step, it perceives its situation through its sensors and can perform an action thanks to its effectors before a new time step starts. The goals of the agent are defined by scalar rewards provided by the environment. In this example, the agent gets a reward as soon as it reaches the cell F that provides food. The task of the agent consists in learning the optimal behavior. Here, the agent must learn how to reach the food by performing as few successive actions as possible, starting from any cell. RL tackles problems which can be represented as finite-state diagrams. Maze problems are well suited to study such problems since they provide an intuitive view of such diagrams.

Trying to improve the policy does not impede extracting knowledge about the problem that the agent has to solve. Indeed, a way to speed up the policy learning process is to simultaneously learn a model of the dynamics of the interactions between the agent and its environment [SB98]. In this article, we will mainly focus on the problem of extracting such a model.

This idea reaches far back in psychology. In sharp contrast with behaviorist theories, Tolman [Tol32] proposed that learning is the process of discovering what leads to what – *i.e.* that animals develop a sort of internal representation of the world. Seward [Sew49] provided further empirical evidences of such *latent learning*, which is defined as learning without environmental reward or punishment. Such a representation of the environment can be used to anticipate the consequences of an action in a given situation.

Computational models of learning are also concerned by such evidences of latent learning. When an agent interacts with its environment, the consequence of an action does not only consist in a possible reward, but also in a resulting new situation. Thus the agent may learn latently what happens immediately after the execution of an action and may build a model of the transitions between situations perceived successively. This model of the transitions makes it possible to *anticipate* and this capacity can be used either for planning thanks to Dynamic Programming techniques, or for speeding up the RL process by simulating actions according to the model, as shown in [SB98]. In other words, learning latently a model of the dynamics of the interactions between the agent and its environment is independent from the reward but helps to improve the overall RL process.

In this paper, we study how generalization capacities may expedite latent learning in a Learning Classifier System (LCS) devoted to anticipation. The problem to be solved is that of extracting knowledge about the dynamics of the interactions between the agent and its environment. It is a classification problem since the agent must learn a model that distinguishes situations which lead to different effects.

More specifically, we study how to acquire and to use anticipation capabilities in order to solve the action decision problem faced by the agent. In particular, we describe how to use Dynamic Programming techniques in order to build two different policies: one for active exploration and one for cumulative reward maximization. These two policies define two criteria that the system has to combine in order to solve the overall RL problem stated above.

In section 2, we briefly present the usual LCS approach to generalization as an extension of Q-learning [Wat89]. In section 3 we introduce the formalism used in the so-called Anticipatory

Classifier System ACS [Sto98, BGS00] and YACS[1] [GS01b, GSS02, GS01a] so as to combine generalization and latent learning. We also discuss a variety of regularities in the interactions between the agent and its environment that neither ACS nor YACS are able to consider. Then, we propose a new formalism to deal with that kind of regularities, thus making it possible to learn a more compact model. In section 4 we describe MACS[2]: a new LCS using this formalism to learn the model that is required to use iterative planning techniques from Dynamic Programming. In section 5 we show how MACS uses this model to build separate policies for active exploration and exploitation, and how these policies are combined. Section 6 provides experimental comparisons of MACS and YACS with respect to their latent learning abilities. The results demonstrate the capacity of MACS to use the model of the transitions to solve plain RL problems, by combining exploration and exploitation criteria. In section 7, we enlight some limitations of MACS, and we claim that the benefits of anticipation capabilities could also be obtained with non-specifically dedicated systems.

2 Generalization in Learning Classifier Systems

The main advantage of Learning Classifier Systems (LCS) with respect to other Reinforcement Learning (RL) techniques like *Q-learning* [Wat89] is to afford generalization capabilities. This makes it possible to aggregate several situations within a common description so that the representation of the RL problem gets smaller.

The first proposals for a LCS devoted to RL problems are presented in [Hol76]. The first implementation of an actual LCS, called CS1, can be found in [HR78]. Wilson [Wil95] introduced in LCSs a learning algorithm similar to Q-learning [Wat89] to replace the traditional *Bucket Brigade* algorithm [Hol85]. This work led to a revival of LCS research since the new accuracy-based approach in XCS overcomes the over-generalization problems found in previous LCSs [Wil89].

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 $r : S \times A \times S \rightarrow \Re$ 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 updates a Q-table representing a quality function $q : S \times A \rightarrow \Re$. Thus, the quality $q(s, a)$ represents the expected payoff when the agent performs the action a in the state s, and follows the best policy thereafter.

At time step t, the qualities are updated according to the following formula based on the Bellman equation used in Dynamic Programming:

$$q(s_{t-1}, a_{t-1}) \leftarrow (1-\alpha)q(s_{t-1}, a_{t-1}) + \alpha(r_t + \gamma \max_{a \in A} q(s_t, a)) \quad (1)$$

where s_t is the state resulting from taking the action a_{t-1} in the previous state s_{t-1} and r_t is the associated immediate reward. α is the learning rate of a Widrow-Hoff delta rule[3]. γ is the discount factor used in the Bellman equation [Bel57]. The effect of this equation is to assign low qualities to states that are "far" from a set of distant reward sources.

[1]Yet Another Classifier System

[2]Modular Anticipatory Classifier System

[3]The Widrow-Hoff delta rule uses a learning rate $\beta \in [0, 1]$. A scalar x is increased with such a rule with respect to the formula: $x \leftarrow (1-\beta)x + \beta$. It is decreased according to the formula: $x \leftarrow (1-\beta)x$

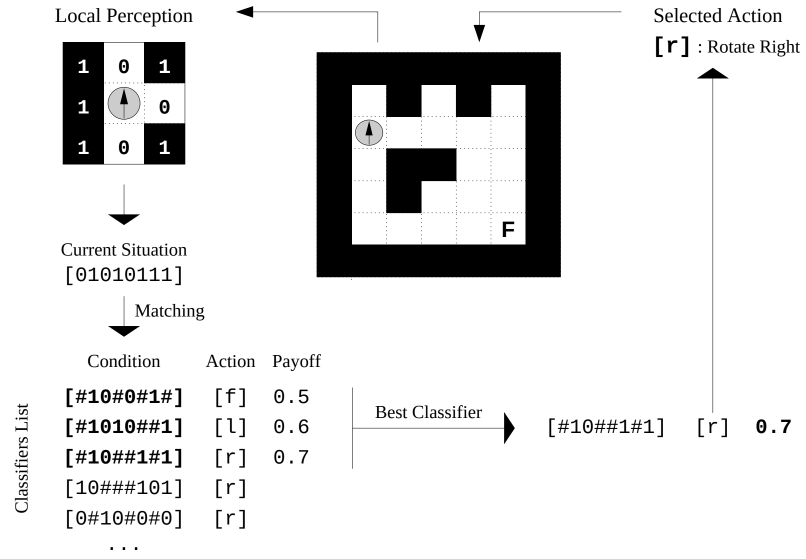

Figure 2: A Learning Classifier System is an agent characterized by a list of classifiers. It perceives a situation as an aggregation of several attributes. In this example, the agent is situated in a maze and perceives the presence or absence of a block in each of the eight surrounding cells. It perceives symbol 1 when there is a block in that cell and symbol 0 when there is not. The eight perceived attributes are considered clockwise, starting with the cell in front of the agent. The agent has to decide whether to rotate right [r] or left [l], or to move forward [f]. From the cell it is currently located in, and according to its orientation, the agent perceives [01010111]. Among the classifiers learned during its past experience – each characterized by a condition – an action and a payoff prediction, the LCS selects those whose conditions match the current situation. Among this matching set, the LCS selects stochastically a classifier with a high prediction of the long term payoff (here 0.7). The action proposed by this classifier is chosen and actually performed in the environment. In this example, the agent rotates right.

The problems tackled by LCSs are characterized by the fact that the so-called *states* in the MDP framework are defined by several attributes representing perceivable properties of an environment. For instance, one can define a grid world in which the agent perceives eight features, one for each adjacent cell (see figure 2). In such a world, a *situation* is an ordered set of several discrete values, one for each of the perceived attributes of the environment. Actions are characterized by a single attribute that represents different possible effectors.

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 $(s, a, q) \in S \times A \times \Re$, 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 LCS framework, the use of *don't care* symbols "#" in the condition parts 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). For instance, assuming that the attributes that characterize situations may only take values 0 or 1, the condition [#01] is general and matches the specialized situations [001] and [101]. Likewise, the condition [#0#] matches 4 situations: [000], [001], [100] and [101].

Thanks to the *don't care* symbols, it is possible to build a model of the expected payoff with a smaller number of classifiers than the number of possible triples in a tabular representation. Usually, the payoff predictions p of the classifiers are learned according to different propagation of delayed reward algorithms, like Bucket Brigade [Hol85] or Q-learning, as in XCS [Wil95].

The main issue with generalization is to to organize C parts (conditions) and A parts (actions) so that the *don't care* symbols are well placed. When a distinction should be specified by a specific attribute but is not, then the classifier is too general. It matches too many situations and its prediction cannot be accurate. Conversely, unuseful distinctions should be avoided to provide the optimal level of generalization.

To do so, LCSs usually call upon a *Genetic Algorithm* (GA) to evolve a population of classifiers. Each classifier is an individual which is evaluated through the interaction of the agent with the environment all along its life time. These algorithms use classical genetic operators like *crossover*[4] or *mutation*[5]. These operators serve to correctly position *don't care* symbols and specialized values in the C parts of the classifiers. Classical LCSs use a selection mechanism relying on *fitness* values and classifiers with a low fitness tend to be suppressed from the classifier list. Such GAs create classifiers randomly and evaluate them afterward. Alternatively, in section 4, we propose devoted estimates and heuristics to drive the creation of classifiers.

3 Formalisms for transitions modeling in Learning Classifier Systems

In multi-step problems, the agent needs to act more than once so as to solve the problem. In such conditions, in addition to a reward, the agent also receives a new situation as a result of its last action. Thus, it makes sense to use LCSs to learn a compact model of the dynamics between the agent and its environment.

3.1 Representing regularities with ACS and YACS

Early LCSs [Hol90] actions could deposit internal messages on a so-called message list instead of suggesting an actual action in the environment. With such internal messages, it was possible to use tags that specified if a current "action" posted to a message list directly suggested an action or participated in an internal reasoning process. Riolo [Rio91] implemented such capacities in CFSC2 and demonstrated how they can be used for latent learning.

In contrast with this approach, the ALP (Anticipatory Learning Process) used in ACS [Sto98, BGS00] is a development of the *Anticipatory Behavioral Control* theory introduced in psychology by Hoffmann [Hof93]. YACS [GS01b, GSS02, GS01a] is a similar approach and both systems call upon explicit `condition-action-effect` classifiers, noted `C-A-E`. This formalism is similar to Sutton's DynaQ+ [Sut91] approach or to Drescher's `context-action-result` rules [Dre91], but it affords generalization capabilities.

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 both ACS 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 (see section 2). 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.

For instance, let us consider the classifier `[#0#1] [0] [=10=]`. It anticipates the effects of the action `[0]` in 4 possible situations (`[0001]`, `[0011]`, `[1001]` and `[1011]`) thanks to the *don't care* symbols in the C part. According to the E part, and if the classifier is accurate:

- the first attribute remains unchanged, whatever the initial value is (`0` or `1` because of the *don't change* symbol in the C part);

[4] A new classifier is a combination of different segments of its parents.

[5] Any attribute of a new created classifier may be randomly changed to any specific value or to a *don't care* symbol.

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

Table 1: Illustration of the anticipation mechanism in YACS

- the second attribute will change from `0` to `1`;
- the third attribute will be `1`, whatever the initial value is;
- the last value of the attribute remains `1`.

These cases are illustrated in table 1.

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 it, 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 maximally general C parts that accurately model the dynamics of the environment. A classifier is said to be *maximally general* if it cannot contain any other *don't care* symbol without becoming inaccurate. It is said to be *accurate* if, in every situation matched by its condition, effecting the corresponding action always leads to the same changes in the perceived situations.

Thus, *generalization* is not the same process in ACS or YACS than in XCS [Lan97]. Indeed, in ACS and YACS, generalization is afforded by the joint use of *don't care* and *don't change* symbols and makes it possible to represent regularities in the transitions between successive situations. Moreover, it provides the system with:

- a kind of *selective attention*, when some situations can be identified by paying attention to some attributes only;
- the ability of considering several situations defined by the same condition, thus reducing the size of the model that describes the dynamics of the environment.

As YACS does not generalize with respect to a payoff prediction, it is able to generalize over situations with different expected payoffs. As a result, it does not make sense to store information about the expected payoff in the corresponding classifiers. Therefore, the list of classifiers only serves to model environmental changes.

3.2 Representing more regularities

Generalization makes it possible to represent regularities in the dynamics of the interactions with the environment. However, if 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 an unsecable whole. To make this point clear, let us consider an agent in a grid world as in figure 2. 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 any 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 one *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 only forbids

[11001100]		←*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 →		[00110011]
		[00110111]

Table 2: During the integration process, the LCS proposed in section 3.2 scans the E parts and selects those 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 to [00110011] or to [00110111]. If all the classifiers were accurate, this process would generate only one possible anticipation.

to represent 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, we propose to describe the expected situations E, not with *don't change* symbols, but with new *don't know* symbols "?". This way, the accurate classifier [####1###] [$\curvearrowright$] [??1?????] means that "*just after turning right, the agent always 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, thanks to these new *don't know* symbols, a classifier may anticipate a few attributes only and the overall system gains the opportunity to discover new regularities.

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 no more the single classifier but the whole LCS. Given a situation and an action, a single classifier is not able to predict the next situation: it just describes a partial view of it, which is focused on a few attributes only. The system accordingly needs an additional mechanism which *integrates* these partial views and builds a whole anticipated situation, without any *don't know* symbol in its description, as shown in Table 2.

4 Latent Learning in MACS

As defined in section 3.2, an E part may contain several *don't know* and several specific attribute values. In the present work, we adopted a simplified point of view by allowing one and only one specific value in an effect part. Thus, every classifier is able to predict the value of a single attribute only.

In this section we detail the latent learning mechanisms of MACS, a new LCS designed to take advantage of the formalism just proposed.

4.1 Evaluation and selection of the accurate classifiers

This part of the system is in charge of evaluating the accuracy of each classifier and of suppressing some of them if necessary. Two integer values `g` and `b` are associated to each classifier:

- `g` for storing the number of *good* anticipations since the creation of the classifier;
- `b` for storing the number of *bad* anticipations since its creation.

MACS keeps a memory of the last perceived situation and the last performed action. Thus, it knows the current situation s_t resulting from the action a_{t-1} in the situation s_{t-1} at every time step.

With this information, YACS scans the classifier list and selects the classifiers whose C part matches s_{t-1} and whose A part matches a_{t-1}. For each such classifier:

- if its E part matches s_t[6], then the classifier anticipated well and its `g` value is increased by one unit;
- if its E part does not match s_t, then the classifier anticipated badly and its `b` value is increased by one unit;

A classifier which always anticipates badly during a given number of evaluations is considered inaccurate and is suppressed. This number of evaluations is a parameter of the system, noted e_r. A classifier is suppressed when `g`$= 0$ and `b`$= e_r$. Another parameter e_a of the system monitors how many evaluations are needed to assume that a classifier is *accurate*.

4.2 Specialization of conditions

As in YACS, a classifier which anticipates sometimes well, and sometimes not, is said to *oscillate*. Since its condition part is too general and matches too many situations, it must be specialized. Here again, this process is not driven by a GA but by heuristics which take advantage of specific estimates as described below.

4.2.1 The estimates used by the specialization process

An *expected improvement by specialization* estimate i_s is associated to each general attribute of the C part of each classifier – *i.e* to each *don't care* symbol. This variable estimates how much the specialization of the attribute would help splitting the situation set covered by the C part into several subsets of equal cardinality.

Let us consider a classifier which tries to anticipate the consequences of an action in several situations. If the value of a particular attribute of the situation when the classifier anticipates well is always different from the value of that attribute when the classifier anticipates badly, then this attribute is very relevant for distinguishing the situations covered by the C part. Thus, the C part must be specialized according to this particular attribute, and the corresponding estimate i_s should get a high value.

In order to compute the estimates i_s, each classifier memorizes the situation s_b preceding the last anticipation mistake, together with the situation s_g preceding the last anticipation success. At every time a classifier is such that its C part matches s_{t-1} and its A part matches a_{t-1}, its accuracy is checked:

- if the classifier anticipates well, for each attribute:
  - if a particular attribute of s_b equals the corresponding attribute of s_{t-1}, then the corresponding estimate i_s is *decreased*;
  - if a particular attribute of s_b differs from the corresponding attribute of s_{t-1}, then the corresponding estimate i_s is *increased*;

[6] a *don't know* symbol matches any value.

- if the classifier does not anticipate well, for each attribute:
  - if a particular attribute of s_g equals the corresponding attribute of s_{t-1}, then the corresponding estimate i_s is *decreased*;
  - if a particular attribute of s_g differs from the corresponding attribute of s_{t-1}, then the corresponding estimate i_s is *increased*.

The i_s estimates are increased and decreased according to a Widrow-Hoff delta rule. They are increased according to the formula:

$$i_s \leftarrow (1-\beta)i_s + \beta$$

and decreased according to the formula

$$i_s \leftarrow (1-\beta)i_s$$

The initial values are 0.5. A specialized attribute is given the same default i_s value of 0.5.

4.2.2 The specialization process

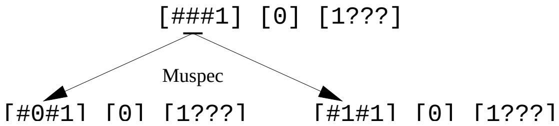

Figure 3: In this example, the *mutspec* operator specializes the C part of a classifier according to the second attribute. The original classifier is replaced by two new specialized versions.

A classifier is said to *oscillate* when g+b> e_o and g×b> 0, where e_o is a parameter of the system that represents the number of evaluations necessary to detect that a classifier oscillates.

As soon as such a classifier is identified, the *mutspec* operator [Dor94] is applied (see figure 3). This operator replaces the oscillating classifier by several more specialized versions. Some of the classifiers thus produced by the *mutspec* operator will always anticipate badly, but they will be eliminated by the *selection of accurate classifiers* process. Some of the classifiers created by the *mutspec* operator will still oscillate and will be specialized again. Finally, a new classifier which does not match any of the already perceived situations is not added in the set. This property can be checked thanks to the 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[7].

In contrary to usual *mutspec* practice [Dor94], the attribute to specialize in MACS is not chosen randomly, but thanks to the i_s estimates. The specialized attribute is the one with the highest i_s value, assuming such change is the most likely to improve the system.

4.3 Generalization of conditions

The specialization process may produce classifiers with a C part at a sub-optimal level of generality, especially in the case of *local exploration*, when the agent only experienced a part of the environment. Thus, MACS needs a generalization process which is in charge of reconsidering early sub-optimal specializations. As it is the case with the specialization process, estimates i_g and dedicated heuristics are used in order to take advantage of experience for driving the generalization process.

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

4.3.1 The estimates used by the generalization process

In order to compute the i_g estimates, MACS selects at every time step each classifier whose A part matches a_{t-1} and whose C part does *not* match s_{t-1}.

Considering such classifiers, for each specialized attribute in the C part, MACS checks if the C part of the classifier would match s_{t-1} if the considered attribute were general. In this case, the considered i_g estimate is updated:

- if the E part of the classifier matches s_t, 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 match s_t, 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 attribute is given a default i_g value of 0.5.

Up to that point, according to such a mechanism, MACS is able to check if an attribute of a C part should be generalized or not.

4.3.2 The generalization process

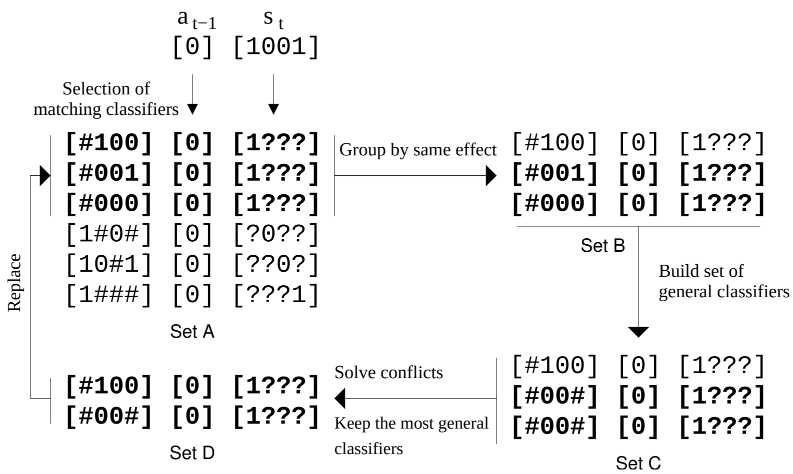

Figure 4: The generalization process in MACS. See text for explanations.

At every time step, according to the way they are updated, the i_g estimates only increased for the classifiers whose A part matches a_{t-1} and whose E part matches s_t. They are selected by MACS in the list of all classifiers. In the example of figure 4, these selected matching classifiers are identified as Set A.

The classifiers of Set A are grouped by similar effects in Set B. In our example, we only consider one of the resulting Set B, but the following process is repeated for each Set B that corresponds to similar E parts and similar A parts. Set B is only processed to build the Set C of general classifiers if all the classifiers of Set B are *accurate*. This way, only accurate classifiers are generalized.

With Set B, MACS builds a new Set C of classifiers which are either more general or mere replicates of the original ones. For each classifier of Set B:

- if every estimate i_g of the classifier is lower than 0.5, then it is not a good candidate for the generalization and it is added without modifications in Set C;
- otherwise, a new classifier is added to Set C. The attribute of the C part with the highest estimate i_g is generalized.

In our example, despite that the estimates do not appear on figure 4, they are used to decide that the C parts of the second and third classifiers should be generalized according to the fourth attribute. Indeed, this attribute is not relevant since both classifiers are accurate and anticipate the same value 1 for the fourth attribute. The first classifier remains unchanged and the two other ones are generalized according to the fourth attribute.

At this point, each classifier of Set C is checked for *conflicts* with other classifiers of the global list. Two classifiers are in conflict if they anticipate a different value for the same attribute, given an initial matching situation and action. If a classifier of Set C is involved into a conflict, the corresponding original classifier of set A is added to the new Set D. Otherwise, the classifier of Set C is added. Two classifiers are conflicting if their C and A parts are compatible, but if their E parts are not. Two E parts are *incompatible* if they do not match – *i.e* if the values of the symbols that are not *don't know* are different. Two C parts are compatible if they match and if at least one possible situation is matched by both C parts. MACS finds the possible situations in the set P of every perceived situation encountered during the lifetime of the agent (see section 4.2.2).

In order to only keep the most general classifiers in set D, MACS checks iteratively every possible pair of classifiers in that set. When the C part of a classifier is more general than that of another classifier, the former classifier is kept and the latter is suppressed. In our example, MACS only keeps one of the two last classifiers.

Up to that point, MACS has built a new Set D of classifiers which are equal or more general than the original ones in Set A. The classifiers of Set A are replaced in the list of classifiers of the system by the classifiers of Set D.

This process make it possible 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 conflict with other classifiers in the system (classifiers with incompatible E parts are not overlapping).

4.4 Transition covering

To fully describe a given environment, a model needs to cover every encountered transition in this environment. This may not be the case in the following circumstances:

- the system is initialized with an empty list of classifiers;
- the *selection of accurate classifiers* may eliminate inaccurate or oscillating classifiers because of local exploration, when the agent experienced only a part of the environment;
- the *condition specialization* process creates a specialized classifier only when such classifier matches at least one already perceived situation. As long as the agent does not experience every possible situation, relevant classifiers may not be added to the classifier list.

To summarize, at every time step, the system covers the transitions defined by s_{t-1}, a_{t-1} and s_t. For each attribute f of s_t, it considers an hypothetical part E_f of E such that its single specific attribute (which is not a *don't know* symbol) is set to its value in s_t. Among the classifiers with an A part corresponding to a_{t-1}, the system checks whether there is at least one such classifier whose C part matches s_{t-1} and whose E part equals to E_f. If it is not the case, the system *covers* the transition by adding a new classifier in the classifier list.

The A part of this covering classifier is set to a_{t-1}, its E part is set to E_f, whereas its C part is set as general as possible with respect to the following constraint: its C part does not match each of the C parts of the classifiers with the same E and A parts, but its C part matches s_{t-1}.

5 Combining Latent Learning and Dynamic Programming

In section 4, we described how MACS learns a model of the dynamics of the environment with anticipating classifiers. In this section, we show how this model is used in a Dyna architecture [Sut91] to define a policy. In such architectures, as illustrated in figure 5, the latent learning process takes place independently from the reward, and permits the construction of a model of

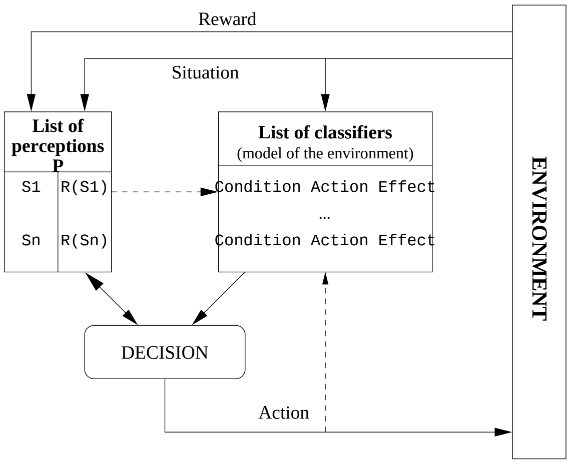

Figure 5: MACS uses a Dyna architecture [Sut91] to perform reinforcement learning. In this kind of architecture, the model of the environment is learned latently, *i.e. independently* from the reward. The informations about the rewards are stored apart, here in the list of perceptions P, and a decision module takes advantage of both informations to build a policy.

the environment. This model is then used to improve the learning speed of a policy, thanks to methods inspired by Dynamic Programming.

In the first part of this section, a description of how MACS uses a partial and inaccurate model of the transitions to drive the exploration process is given. The second part is devoted to the process of learning a policy to maximize the cumulative reward provided by the environment. We also show how MACS combines active exploration and exploitation.

5.1 Active Exploration

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

5.1.1 The internal immediate reward

In order to be able to drive the behavior of the agent, we define an internal reward function $i : S \times A \to \Re$ which estimates the immediate gain in information, given a situation and an action. With this function, MACS is able to choose the action which maximizes the information, in a given situation.

In a situation s_0, when the system has to choose an action in order to maximize the immediate information gain, MACS selects every classifier such that its condition matches s_0. If the action suggested by any such classifier was chosen by the system, the classifier will be evaluated at the next time step: if the number of good evaluations g does not increase, the number of bad evaluations b will, and conversely (see section 4.1).

For each of these classifiers that match s_0, MACS computes an evaluation level $l \in [0, 1]$. This level depends upon the number of good evaluations g, the number of bad evaluations b, and the number of evaluations needed to classify a classifier as inaccurate, accurate or oscillating:

- if $b > 0$ and $g > 0$, then the classifier needs to be evaluated further to gain information about the best way to specialize it and $l = \min((b + g)/e_o, 1)$;
- if $b = 0$ and $g > 0$, then the classifier c may be accurate but further evaluations are needed to check that point and $l = \min(g/e_a, 1)$;

- if $b > 0$ and $g = 0$, then the classifier c never anticipated well but requires further evaluations to be suppressed and $l = \min(b/e_r, 1)$;
- if $b = g = 0$, then the classifier has never been evaluated and $l = 0$.

Thus the level l is equal to 1 if the classifier does not need to be evaluated anymore before being suppressed, specialized or generalized. It is equal to 0 when the classifier needs additional evaluations.

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[8]. Only the anticipated situations which belong to the set P of already encountered situations (see section 4.2.2) are considered in $S_{s_0,a}$. 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 $l(s_0, a, s_1)$ associated to this transition as the product of the evaluation levels l_c of all the classifiers c involved in this anticipation:

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

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. If the transition occurs, the associated immediate information gain is:

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

We define the immediate information gain associated to 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 $i(s_0, a)$ is given the default value 1, which is the maximum immediate information gain.

This value $R_i(s_0, a)$ is used as an immediate reward in order to compute a policy for active exploration. It is computed by analyzing the model and does not rely on the environment. Thus, we call it an internal reward.

5.1.2 Planning to maximize the information gain on the long run

To perform active exploration, MACS has to maximize the cumulative immediate information gain on the long run. Therefore, the system must perform lookahead planning to be able to act in order to get information in the future, even if it currently perceives a situation such that no immediate internal reward is available.

The planning process relies on the immediate internal rewards and on the current model of the dynamics of the interactions with the environment. But, during the learning process, this model of the transitions is not reliable and may be misleading. For instance, an agent may plan on the basis of a transition which cannot be actually experienced. In that case, it may happen that the policy resulting from the model leads the agent into an infinite loop, with no chance to reconsider the misleading transition. Thus, the planning process must be cautious in order to avoid taking decisions relying on inaccurate information.

During the learning process, the model of the transitions improves, and the immediate internal rewards are changing a lot. Thus, the exploration policy of the agent is not stable at all over successive time steps.

Nevertheless, because we want to keep the agent reactive, it is not suitable to compute a whole plan at every time step. Therefore, we use an iterative planning approach similar to that of Value Iteration [SB98], and inspired from the Dynamic Programming approach. At every time step,

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

MACS only updates once the values associated to situations and the policy improves over several time steps and keeps near-optimal in most situations. However, since the model is not perfectly accurate, finding an optimal policy with respect to this model is not necessary. We only want that the resulting behavior makes the agent learn a model more quickly than with a random policy.

Each situation of the set P of already encountered situation is valued by the discounted information gain which can be expected from this point. Thus, the valued perception set P serves to store the values of the situations while transitions are computed according to the classifier list (see section 3.2), and immediate internal rewards are associated to each transition.

At every time step, MACS performs one simulated action for each situation s_0 of the perception set P. For each action a, the immediate reward associated to s_0 and a is $R_i(s_0, a)$ (see section 5.1.1). The expected future reward associated to a transition (s_0, a, s_1) is the discounted value $V_i(s_1)$ associated to s_1 in the set P.

Planning thanks to an inaccurate model can result in a sub-optimal policy and even to and endlessly cycling behavior. In order to avoid this kind of problems, MACS is cautious with respect to the expected discounted reward. Thus, given a situation s_0 and an action a, we define the expected information gain as:

$$E_i(s_0, a) = \min_{s_1 \in S_{s_0,a}} V_i(s_1)$$

The min in this equation reflects the cautiousness of MACS. Indeed, for each action, the system considers the minimum internal reward he should get, with respect to the model. With this information, MACS computes the quality associated to the situation s_0 and the action a:

$$Q_i(s_0, a) = R_i(s_0, a) + \gamma E_i(s_0, a)$$

Then, MACS updates the new value of s_0:

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

The γ factor is the discount factor. It plays the same role as in equation 1. This way, at every time step, MACS updates several values in the perception set and the policy improves. During the action selection process, when the perceived situation is s_t, MACS chooses the action that maximizes $Q_i(s_t, a)$.

5.2 Reinforcement Learning in MACS

In section 5.1, we defined how MACS performs active exploration thanks to iterative planning techniques. In this section, we show how MACS uses the model of the transitions to build a policy that maximizes the environmental payoff on the long run.

5.2.1 Learning a policy for the environmental reward

At every time step, MACS receives a scalar reward r_t and a new situation s_t from the environment, as the result of taking action a_{t-1} in situation s_{t-1}. This immediate environmental reward is associated to s_t in the perception set P. We note it $R_p(s_t)$. This function R_p represents the goals defined by the environmental rewards of the system. Here again, we design an iterative planning mechanism which permits MACS to take advantage of its model to reach the goals.

As in section 5.1.2, MACS simulates several successive actions at every time step. When MACS simulates an action with the situation s_0 as a starting point, it uses the model of transitions provided by the list of classifiers and the integration mechanism (see section 3.2) to compute for each action a the set of possible anticipations s_1. A payoff value $V_p(s)$ is associated to each situation s in the perception set P. This value represents the desirability of the corresponding situation. The reinforcement learning process updates these values iteratively thanks to the model of the transitions and to the immediate environmental rewards. Here again, the learning process is cautious because along the latent learning process, the model may be inaccurate. First, given all the possible transitions from s_0, MACS computes the qualities associated to the actions:

$$Q_p(s_0, a) = \min_{s_1 \in S_{s_0,a}} [R(s_1) + \gamma V_p(s_1))]$$

The γ factor is the discount factor. It plays the same role as in equation 1. MACS computes and updates the new value of s_0 with this information:

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

During the action selection process, when the perceived situation is s_t, MACS chooses the action to maximize $Q_p(s_0, a)$. This way, MACS builds an exploitation policy that enables it to decide which action to take in each situation to reach the goals defined by its environmental reward.

5.2.2 Combining the exploration and the exploitation policy

Section 5.1 and 5.2.1 respectively described how MACS computes a policy devoted to active exploration and how it computes a policy for reaching the goals defined by its environmental reward. We now address the issue of combining these policies in order to generate a behavior combining exploration and exploitation.

This combination takes place during the action decision process. When MACS perceives the situation s_t, it computes for each action a the qualities $Q_i(s_0, a)$ and $Q_p(s_0, a)$ respectively associated to the exploration and the exploitation, before aggregating them. In this case, it does not make sense to aggregate these criteria by computing a weighted sum because none of both qualities are bounded. Indeed :

- if there are many immediate gains of information, even if they all belong to $[0, 1]$, the discounted sums may be high;
- the level of the environmental rewards cannot be known by the system before the learning process is actuated since the environment is unknown.

Thus, we cannot select adequate weights in advance. However, we can still define a hierarchy between both criteria. As the optimality of the exploitation policy relies on the reliability of the model, seeking information that helps improving the policy is given the priority against the payoff maximization. The selection of an action takes place the following way (we note $A_{\setminus\{a_0\}}$ the set A, excluding a_0) :

- If $\exists a_0 \in A$ *tq.* $\exists a \in A_{\setminus\{a_0\}}$ *tq.* $Q_i(s_t, a_0) > Q_i(s_t, a)$ then the chosen action is a_0
- Otherwise, the chosen action a_1 is such that $Q_x(s_t, a_1) = \max_{a \in A} Q_x(s_t, a)$

This way, if the actions are equivalent with respect to the information gain, then MACS chooses the action according to the exploitation policy.

6 Experimental study

Up to that point, we described MACS, a new LCS that performs a new kind of generalization when compared with XCS, ACS or YACS. In [GS01a] and [GSS02], we compared the ability of YACS and ACS to build a model of the dynamics of the interactions between the agent and its environment. We showed that YACS provides an improvement over Stolzmann and Butz's ACS, in terms of learning speed as in terms of number of discovered classifiers. In this section, we provide experimental comparisons of YACS and MACS interacting with the environments Maze228, Maze252, Maze288 and Maze324, described in section 6.1.

We also provide experimental results about the use of MACS anticipation capabilities to provide this system with non-random policies. The results given in section 6.2 are discussed in section 6.3.

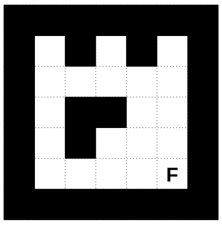

Figure 6: The Maze228 environment

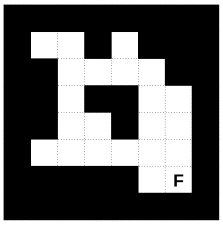

Figure 7: The Maze252 environment

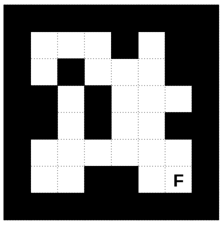

Figure 8: The Maze288 environment

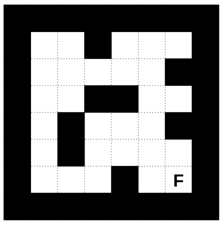

Figure 9: The Maze324 environment

6.1 The environments Maze228, Maze252, Maze288 and Maze324

Like the *Wilson woods problems* that are usually used as benchmarks in the LCS framework, Maze228, Maze252, Maze288 and Maze324 are grid worlds.

Each cell in those grids may be empty or contain either an obstacle ■ or food F. The agent is situated in a cell and is oriented toward one of the four cardinal directions. A perceived situation is described by nine attributes : eight corresponding to the adjacent cells and one to the cell the agent is situated in. An attribute may take three values: `0` (empty cell), `1` (■) or 2 (food)

The agent can choose between three actions: turning 90^o left, turning 90^o right or moving one cell ahead. In this case, if the cell in front of it contains an obstacle, the agent remains in its current cell.

Grid worlds are usually used as understandable representations of finite state automata. In these automata, actions imply transitions among states, represented as graph nodes. Such automata make it possible to represent any reinforcement learning problem with discrete states and actions. Even if grid worlds can only represent a sub class of reinforcement learning problems, they can help to apprehend complex environments. In particular, they provide an easy way to represent attributes. Moreover, maze problems provide regularities which may be used for generalization. In MACS as in YACS, we did not make any assumption concerning the particularities of grid worlds over general finite state automata.

The topologies of Maze228, Maze252, Maze288 and Maze324 are respectively illustrated in figures 6, 7, 8 and 9. Maze228 contains 19 non-terminal cells and $19{\times}4{\times}3{=}228$ transitions may be experienced in the environment. As maze 252 contains 21 non-terminal cells, it is possible for the agent to experience 252 transitions in it. Likewise, Maze288 and Maze324 respectively contain 25 and 26 empty cells and thus, 288 and 324 possible transitions.

From a qualitative point of view, the particularity of Maze288 is that it is less *"open"* than the others. Its left part actually contains two dead ends because the agent cannot move diagonally, while in the other environments, nothing equivalent exists. This originality of Maze288 leads an agent acting randomly to visit less often all the possible situations.

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, regardless of its orientation. In that case, the agent receives a reward of 1.0, it perceives the new situation to learn about the last transition, and a new trial starts.

ă ă	Time to converge Average	 Std. Dev.	Nb. Classifiers Average
Maze228 - YACS (random)	9 295	1 787	199
Maze228 - MACS (random)	4 960	1 737	184.8
Maze228 - MACS (active)	3 001	1 006	181.8
Maze252 - YACS (random)	11 466	2 286	219
Maze252 - MACS (random)	6 695	2 353	193.8
Maze252 - MACS (active)	3 716	1 306	190.2
Maze288 - YACS (random)	20 983	7 114	249
Maze288 - MACS (random)	8 473	3 078	205.3
Maze288 - MACS (active)	4 379	1 686	204.4
Maze324 - YACS (random)	15 740	3 558	283
Maze324 - MACS (random)	11 398	4 538	229.6
Maze324 - MACS (active)	5 518	2 084	227.2

Table 3: Summary of the experimental results

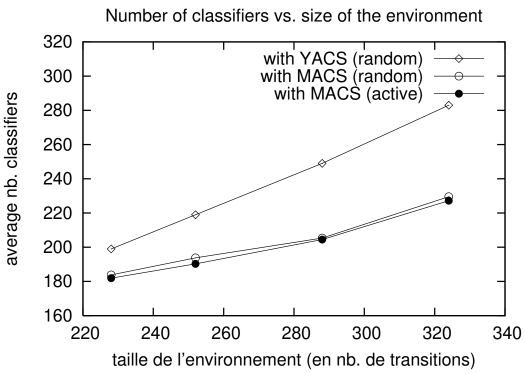

Figure 10: Number of classifiers against the size of the environment

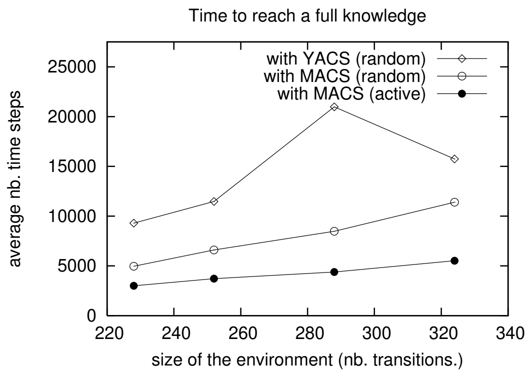

Figure 11: Time to converge against the size of the environment

6.2 Experimental results

In order to estimate the evolution of the accuracy and completeness of the model of the transitions provided by the classifier list and the integration mechanism (see section 3.2) 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 – *i.e.* if it anticipates a single situation only, and if this situation is the actual one. The percentage of knowledge is the ratio of transitions accurately modeled by the system against the total number of transitions to be modeled by the system. This percentage cannot be measured in real world experiments since it requires a perfect knowledge of the environment. However, its evaluation is only possible in simulated environments.

For YACS, as defined in [GSS02], the memory size m is set to 5 and the learning rates are set to 0.1. For MACS, we also used learning rates of 0.1 and e_o, e_r and e_a were all set to 5 (see section 4). The discount factor γ was set to 0.9.

We tested YACS in random exploration and MACS both in random and active exploration in each of the four environments. Table 3 summarizes the results of the different experiments. It shows the average over 100 experiments (and the associated standard deviation) of the time to reach a perfect knowledge of the environment. It also shows the average number of classifiers necessary to model the dynamics of the interactions with the environment.

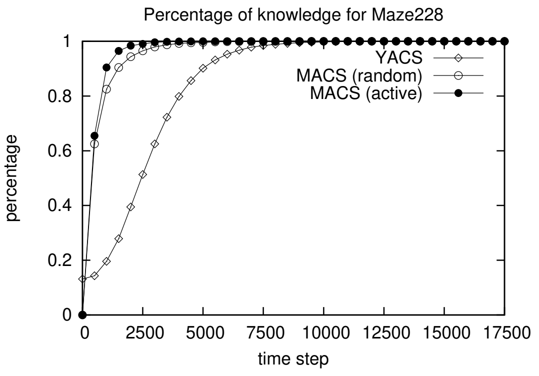

Figure 12: Evolution of the percentage of knowledge in Maze228

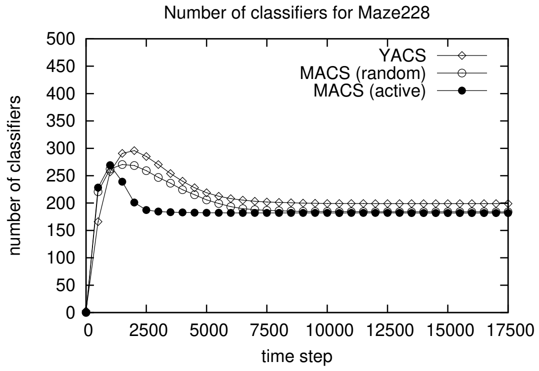

Figure 13: Evolution of the number of classifiers in Maze228

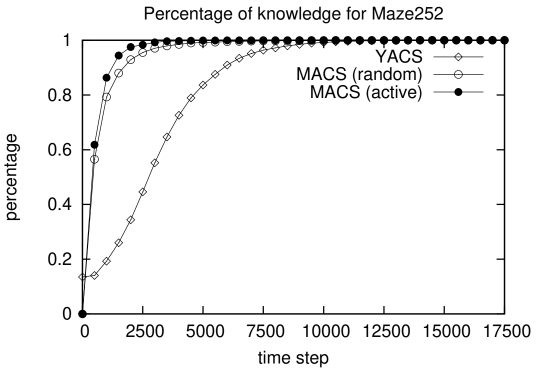

Figure 14: Evolution of the percentage of knowledge in Maze252

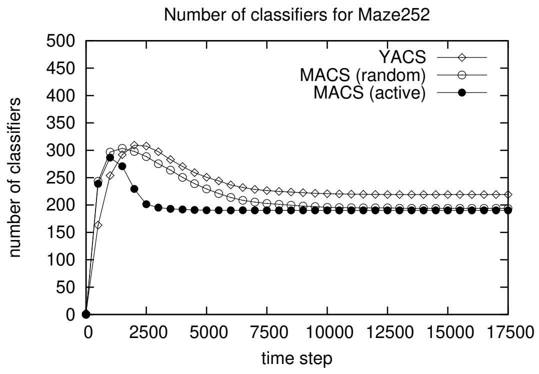

Figure 15: Evolution of the number of classifiers in Maze252

We use unilateral statistical Wilcoxon tests to check the hypothesis of the equivalence of convergence time distributions, against the hypothesis stating that the convergence times for one set of experiments are lower than the times for another set. For each environment, considering one 100 valued sample per experiment, the Wilcoxon tests accept, with thresholds lower than 10^{-5}, the hypothesis that MACS converges faster than YACS in random walk, and that MACS converges faster with active exploration. ă

Figures 10 and 11 show the relationships between the size of the environment on the one side, and the average number of classifiers and the average time to converge, on the other side.

Figures 12, 14, 16, and 18 show the evolution of the percentage of knowledge when YACS and MACS interact with the different environments. Figures 13, 15, 17, and 19 show the evolution of the number of classifiers in the same experiments. All these results are averaged over 100 experiments.

Figures 20, 21, 22 and 23 show how the average number (over 100 experiments) of time steps required by MACS to reach the food evolves along successive trials, when exploration and exploitation are jointly used.

6.3 Experimental results analysis

6.3.1 MACS *vs.* YACS in random exploration

The semantics of the classifiers are different in YACS and in MACS. With the YACS formalism, classifiers model transitions as a whole while, in the MACS formalism, each classifier predicts the

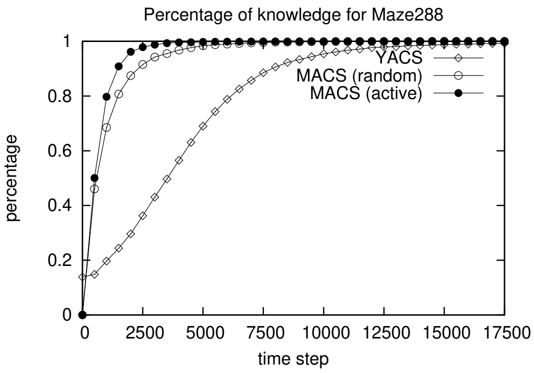

Figure 16: Evolution of the percentage of knowledge in Maze288

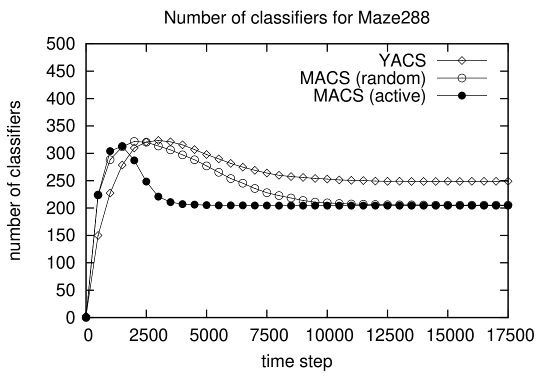

Figure 17: Evolution of the number of classifiers in Maze288

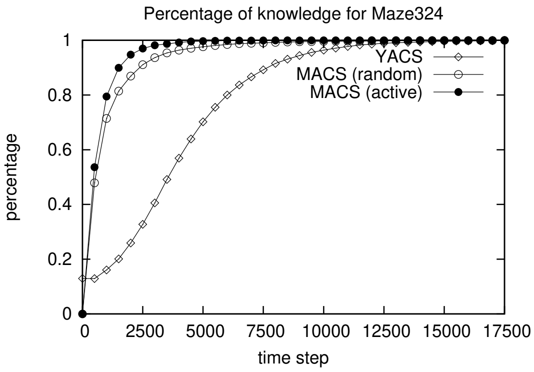

Figure 18: Evolution of the percentage of knowledge in Maze324

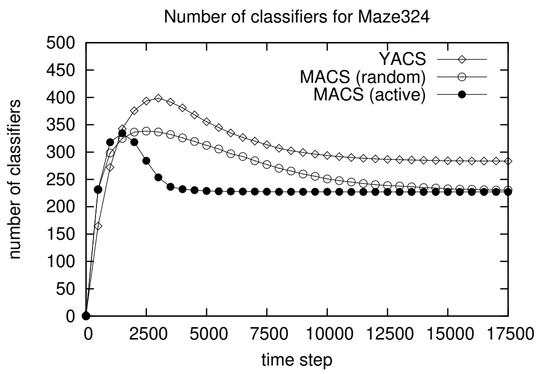

Figure 19: Evolution of the number of classifiers in Maze324

value of a single attribute only. In environments supplying few regularities across attributes, the number of rules discovered by MACS should be higher.

Moreover, there are no *don't care* symbols in the formalism of MACS. As a result, regularities like "moving toward a wall does not make the situation change" are represented with more classifiers in MACS than in YACS – one classifier for each value of each attribute – by taking advantage of its ability to represent regularities involving different attributes of the situations. But the number of such classifiers remains the same whatever the size of the grid world is.

Despite this, in Maze228, like in Maze252, Maze288 and Maze324, MACS converges toward a smaller number of classifiers than YACS, thanks to the new regularities taken into account in the MACS formalism. Indeed, the larger the environment, the more MACS outperforms YACS when considering the ratio between the number of discovered classifiers and the number of actual transitions. MACS exhibits an ability to represent many regularities which are independent from the particular topologies of the mazes, but that concern mazes in general:

- In MACS, every transition involving a turning action is modeled with the same number of classifiers regardless of the number of cells in the grid world. As YACS is not able to represent regularities across attributes, more classifiers are required to model these transitions as the size of the environment grows.

- The only attributes which are difficult to predict for MACS correspond to the cells in front of the agent, when it moves forward and when there is no wall in front of it. Whether such situations occur frequently or not depend on the topology of each particular maze, not on

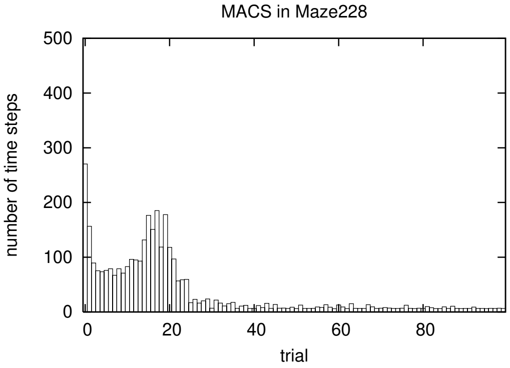

Figure 20: Evolution of the number of time steps to achieve successive trials in Maze228

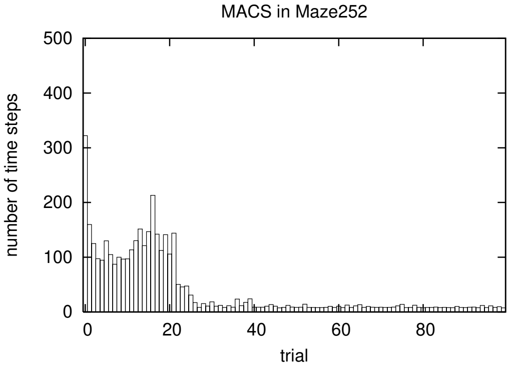

Figure 21: Evolution of the number of time steps to achieve successive trials in Maze252

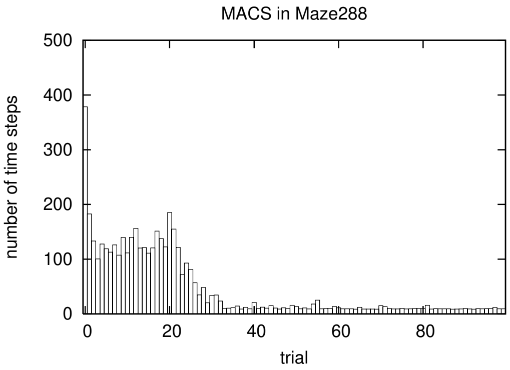

Figure 22: Evolution of the number of time steps to achieve successive trials in Maze288

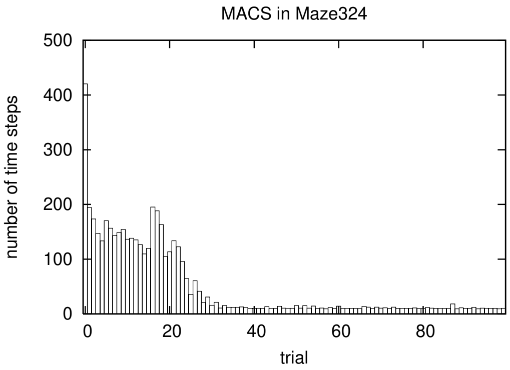

Figure 23: Evolution of the number of time steps to achieve successive trials in Maze324

the general structure of the mazes[9]. In that case, the latent learning process has to take into account regularities which occur less frequently. In order to reduce the number of classifiers necessary to predict such attributes, a solution could be to provide MACS with a mechanism making it possible to build E parts with several specific symbols, as initially proposed in the formalism (see section 3.2).

Regularities of the first kind are quickly discovered by MACS. Indeed, in the first 1000 time steps of the experiments (in any of the tested environments), the percentage of knowledge grows very fast before slowing down and the complete model is learned more quickly in MACS than in YACS.

Such learning speed of rotation actions is due to the fact that, despite the random exploration, there are many relevant examples to drive the learning process, since the concerned regularities are independent from the particular topology of each environment. This is very different when regularities that rely on the topology are concerned. In that case, the more the classifiers are specialized, the more the system experiments transitions that do not provide additional information, and the learning process slows down accordingly.

Despite this random exploration, the learning speed of MACS is linear in the size of the tested environment while the particular topology of Maze288 is problematic for YACS (see section 6.1).

[9]Or the regularities induced by any group of markov problems in general

6.3.2 MACS with active exploration and payoff maximization

When MACS uses active exploration, the average time to reach a complete knowledge is improved over the random exploration situation. Moreover, as shown in figure 11, the improvement gets higher as the size of the environment grows. The evolution of the convergence speed is better than linear in the size of the environment. In addition, the number of classifiers stabilizes faster.

MACS also demonstrates its ability to use the model of the dynamics of the environment in order to learn a policy with respect to the payoff. In this respect, the learning process can be divided into several parts.

During the very first trials, MACS mostly learns the transitions that correspond to the rotations. Because the model is highly inaccurate, MACS cannot propagate the internal reward very well and the system does not take much advantage of the Value Iteration algorithm. The resulting behavior almost looks like a random behavior.

Once the first regularities have been learned, MACS becomes able to plan one step ahead when performing rotation actions, and it is more likely to learn what is happening when moving ahead. Thus, the behavior mostly consists of long straight lines and MACS experiences many different situations. As a result, it reaches the goal more often : the number of time steps to achieve the successive trials decreases.

As the knowledge of MACS about what leads to what when moving ahead improves, the system becomes able to build more complex exploration policies and therefore stays longer without reaching the goal. Thus, the time to reach the goal increases until there is almost no information to gain.

At this point, thanks to the aggregation method of exploration and exploitation policies, MACS starts maximizing the payoff. The behavior becomes optimal with respect to the payoff, although it may happen that MACS reconsiders early suboptimal specializations. In that case, the new classifiers must be validated and MACS switches temporally back to active exploration. This phenomenon explains the peaks in the late trials.

7 Discussion

7.1 Latent learning in MACS and the uncertain

In [GSS02], we showed that the mechanisms of YACS improve the learning speed over ACS. In this paper, we proposed a new formalism for the problem of anticipation in the LCS framework. We showed how MACS, which uses this formalism, improves the learning speed over YACS, then over ACS.

Nevertheless, we pointed in [GSS02], that extensions have been added to ACS in order to deal with the uncertain[10], while YACS only deals with markov and deterministic environments. Such extensions do not exist for MACS either.

The outcomes of an action in a particular situation may be uncertain because:

- the environment is stochastic. In that case, the perceptions or the actions may be noisy, and the outcomes of an action in a given situation are not always the same. This case may occur when the sensors or the effectors are not absolutely reliable;
- some perceptions are ambiguous. In that case, the agent perceives the same situation in different states of the environment (see figure 24). Then, the information provided by the current perception is not sufficient to choose the optimal action, and the problem is said non-markov. The agent must deal with an internal state to disambiguate the aliased perception. The internal state is defined by an information about past situations and actions.

In order to deal with stochastic environments, ACS [BGS01] uses multiple effect parts per classifier, each valued by a probability measure. Some specific heuristics have been added to ACS

[10] The learning speed improvements of YACS over ACS have been shown by considering ACS without these extensions.

to deal with this new feature. In MACS, as in YACS, the heuristics presented in section 4 should also be modified to tackle stochastic environments, but the estimates could be kept. Indeed, MACS estimates are robust to noise, because they use Widrow-Hoff equations. Rather than designing new heuristics, the estimates could also be used to bias usual genetic algorithms.

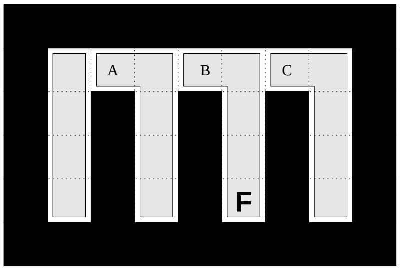

Figure 24: In this environment, the agent always faces north and perceives the eight surrounding cells. The agent can move to any of the eight surrounding cells. Many of the cells are ambiguous. For instance, the agent perceives the same situation in the cells marked a, b and c. The consequences of an east movement in any of these cells are different. Inside each of the highlighted zones, there are no aliased situations

In order to deal with ambiguous perceptions, ACS uses action sequences. In the LCS framework, several other ways have been explored. CXCS[11] [TB00] builds sequences of classifiers and XCSM[12] [Lan98] uses explicit memory registers to define internal states. In the general Reinforcement Learning framework, Wiering [WS97] and Sun [SP00] propose to learn how to divide a non-markov problems into several markov ones (see figure 24).

These techniques all require that information about changes of the internal state can be associated to transitions between situations. Unfortunately, MACS classifiers do not represent transitions as a whole, but provide only partial models of the transitions, corresponding to the single attribute anticipated by each classifier. As a result, a particular classifier may be involved in several transitions and, conversely, several classifiers must be grouped to model one complete transition.. Therefore, the solutions mentioned above could be adapted to YACS, but the partial anticipations in MACS forbid the use of such techniques.

7.2 Relations between Dyna and MACS

In section 5, we described MACS as a Dyna architecture [Sut91]. The main difference between MACS and *DynaQ+* relies in the learned model of the environment. In *DynaQ+*, 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 specify complete transitions. Conversely, in MACS, each classifier only provides a prediction concerning one attribute.

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 $T\ : s_1 \times ... \times s_d \times a_1 \times ... \times a_e \rightarrow s_1 \times ... \times s_d$[13]. 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. The global transition function can be obtained by integrating the partial anticipations.

The MACS architecture illustrated in figure 26 shows how the latent learning part of MACS can be considered as a modular system, each module anticipating one attribute. By contrast, figure 25 shows the monolithic architecture of a *DynaQ+* or ACS/YACS model. Each of these modules provides an approximation of one partial transition function, each predicting one single value.

[11]CXCS stands for Corporate XCS
[12]XCSM stands for XCS with Memory
[13]where e is the number of effectors, ans d is the number of perceived attributes

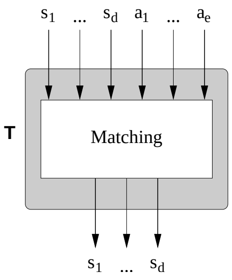

Figure 25: YACS architecture for latent learning.

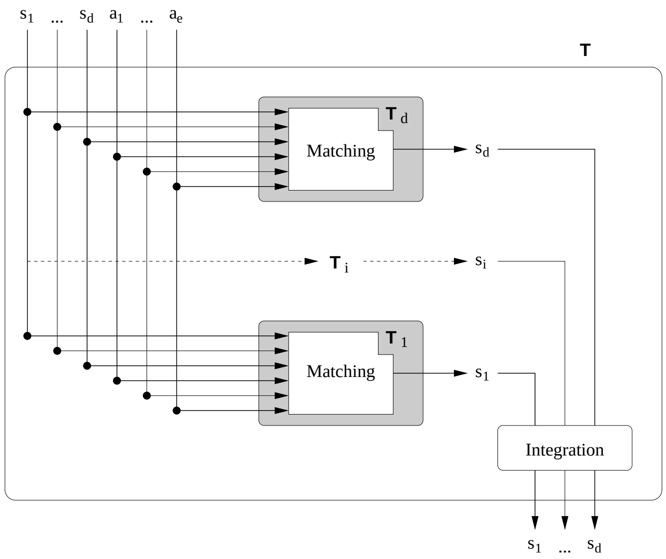

Figure 26: MACS architecture for latent learning.

This architecture suggests that one could replace MACS modules by any function approximation system. This way, it should be possible to take advantage of usual LCSs (which do not use special effect parts) to draw the benefits of anticipation in reinforcement learning problems.

For symbolic functions, it is possible to use well known systems as XCS for these partial anticipations. For numerical functions, it should be possible to use numerical function approximators from the LCS field like XCSF [Wil01], neural networks, locally-weighted function approximators or any other, provided that it is incremental.

8 Conclusion

In this paper, we described several LCSs, each of them casting a new light on the concept of generalization in the LCS framework. In particular, we enlighted how most LCSs – like XCS – consider generalization with respect to an expected payoff, while other LCSs – like ACS or YACS – consider it with respect to anticipated effects in terms of situations. We also enlighted some limitations of the formalism of ACS and YACS. To overcome these limitations, we proposed MACS, a new LCS which uses a different formalism. This formalism makes it possible to use additional regularities for generalization, in the latent learning process of the model of the dynamics in the interactions between the agent and its environment.

Such a model is a prerequisite for the application of Dynamic Programming iterative algorithms for planning. With MACS, we used a Dyna architecture to separate the information about the transitions, and the information about the reward. This kind of architecture makes it possible to use techniques from operational research to speed up the reinforcement learning process. This way, MACS is able to compute policies for active exploration and payoff maximization. Among different methods for aggregating criterions, we have chosen a hierarchical aggregation in order to tackle the exploration/exploitation tradeoff.

MACS formalism for latent learning does not consider situations as an unsecable whole, but it decorrelates the attributes, making it possible to represent regularities across attributes. Experimental results 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. In addition, we showed how decorrelating attributes in the effect part, leads to consider an anticipatory system as a modular system, composed of several systems, each of them predicting one single value. Then, it is possible to use regular and widely studied learning algorithms to learn a model of the environment so that it becomes possible to take advantage of Dynamic Programming techniques to speed up the reinforcement learning process.

We now intend to design a general numerical function approximator with a learning classifier system which makes use of estimates to drive the learning process. This new system will next be integrated in a MACS architecture. This way, we intend to build an anticipatory reinforcement learning system, for stochastic and continuous environments.

References

[Bel57] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton NJ, 1957.

[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)*, 2000.

[BGS01] M. V. Butz, D. E. Goldberg, and W. Stolzmann. Probability-enhanced predictions in the Anticipatory Classifier System. In *LNAI1996: Advances in Learning Classifier Systems.* Springer-Verlag, 2001.

[BXM01] E. Bernardó, Llorà X., and Garrel J. M. XCS and GALE : a comparative study of two Learning Classifer Systems with six other learning algorithms on classification tasks. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Proceedings of the fourth international workshop on Learning Classifer Systems*, 2001.

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

[Dre91] G. L. Drescher. *Made-Up Minds: A constructivist approach to Artificial Intelligence.* MIT Press, Cambridge MA, 1991.

[GS01a] P. Gérard and O. Sigaud. Adding a generalization mechanism to YACS. In L. Spector, E. Goodman, A. Wu, W. B. Langdon, Voigt H. M., M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO01)*, pages 951–957, San Francisco, CA, july 2001. Morgan Kaufmann.

[GS01b] P. Gérard and O. Sigaud. YACS : Combining anticipation and dynamic programming in Classifier Systems. In W. Stolzmann, P. L. Lanzi, and S. W. Wilson, editors, *LNAI 1996 : Advances in Classifier Systems.* Springer Verlag, 2001.

[GSH99] T. Gal, T. J. Stewart, and T. Hanne, editors. *Multicriteria Decision Making: Advances in MCDM Models, Algorithms, Theory, and Applications.* Kluwer Academic Publishers, 1999.

[GSS02] 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*, 2002. to appear.

[Hof93] J. Hoffmann. *Vorhersage und Erkenntnis [Anticipation and Cognition].* Hogrefe, 1993.

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

[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.

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

[HR78] J. H. Holland and J. S. Reitman. Cognitive Systems based on adaptive algorithms. *Pattern Directed Inference Systems*, 7(2):125–149, 1978.

[KLM96] L. P. Kaelbing, M. L. Littman, and A. W. Moore. Reinforcement Learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[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.

[Lan98] P. L. Lanzi. Adding memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98)*. IEEE Press, 1998.

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

[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.

[RV81] B. Roy and P. Vincke. Multicriteria analysis: Survey and new directions. *European Journal of Operational Research*, 8(3):207–218, 1981.

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

[Sew49] J. P. Seward. An experimental analysis of latent learning. *Journal of Experimental Psychology*, 1949.

[SG01] O. Sigaud and P. Gérard. Using Classifier Systems as Adaptive Expert Systems for Control. In W. Stolzmann, P.-L. Lanzi, and S. W. Wilson, editors, *LNAI 1996 : Advances in Classifier Systems*. Springer-Verlag, 2001.

[SP00] R. Sun and T. Peterson. Automatic partitioning for multi-agent reinforcement learning. In J. A Meyer, A. Berthoz, D. Floreano, H. L. Roitblat, and S. W. Wilson, editors, *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior (SAB2000)*, 2000.

[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*, volume 1813 of *Lecture Notes in Artificial Intelligence*, pages 194–208, Berlin, 2000. Springer-Verlag.

[Tol32] E. C. Tolman. *Purposive behavior in animals and men*. Appletown, New York, 1932.

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

[Wil89] S. W. Wilson. A critical review of Classifier Systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, Los Altos, California, 1989. Morgan Kaufmann.

[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.

[WS97] M. Wiering and J. Schmidhuber. Hq-learning. *Adaptive Behavior*, 6(2):219–246, 1997.