Generalization and Latent Learning in Learning Classifier Systems

Pierre Gérard[1,2]

[1] Dassault Aviation, DGT/DPR/ESA
78, Quai Marcel Dassault, 92552 St-Cloud Cedex, France
[2] LIP6 - AnimatLab, 8, rue du Capitaine Scott, 75015 Paris, France
pierre.gerard@lip6.fr

Abstract. The main advantage of Learning Classifier Systems (LCS) with respect to other Reinforcement Learning (RL) techniques is to provide the system with a generalization capability. In this paper, we highlight the differences between two kinds of LCSs : some are used to directly perform RL, others are used to learn latently a model of the transitions between perceived situations thanks to anticipation capabilities. Such a model can be used to perform Dynamic Programming or planning and thus, the latent learning is complementary to the decision process.
We show how the notion of *generalization* differs whether the system anticipates (like ACS and YACS) or not (like XCS). Moreover, we show some limitations of the ACS/YACS formalism and propose a new formalism which allows to represent new regularities. We describe MACS, a new LCS using this new formalism to learn latently a model of the dynamics of the environment. We also compare experimentally MACS with YACS and highlight their difference in terms of generalization capability.

1 Introduction

The Reinforcement Learning framework (RL) [SB98] considers adaptive agents involved in a sensory-motor loop. Such agents perceive situations through their sensors, and use these perceptions to choose an action and act accordingly in the environment. As a result, they receive a scalar reward from the environment and 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.

In this framework, a major concern is the improvement of the policy learning process by simultaneously building an internal model of the dynamics of the interactions with the environment. 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. In animal learning, Seward [Sew49] first gave empirical evidence of *latent learning*: a learning process without environmental reward or punishment.

In the machine learning framework, when an agent is facing a multi-step problem, the consequence of an action does not only consist in a reward, but also in the resulting new situation. Thus it can learn to *anticipate* what happens immediately after the execution of an action and build a model of the transitions between successive perceived situations. This information can be used either for planning or for speeding up the RL process by hypothetically acting as in [SB98]. Such a model is also necessary for Dynamic Programming. Thus the latent learning is complementary to the decision process.

Another concern in the general RL framework is that of generalization. This capability of generalizing what is learned is the main advantage of LCSs [Hol76] with respect to other RL systems like *Q-learning* [Wat89]. It allows to aggregate several situations within a common description so that the representation of the problem gets smaller. In this framework, a new and original trend is focussed on *latent learning* and *anticipation*. This new trend leads to a reconsideration of the notion of generalization.

This paper studies the latent learning process in Markov environments which uses the generalization capability which is central in the LCS framework. More precisely, we propose a new way for considering generalization and show that the underlying concept is very different in the case when a system anticipates from the case when it does not.

In section 2, through XCS [Wil95], we briefly present the usual LCS approach to generalization as an extension of Q-learning [Wat89]. In section 3 we present the formalism used in the Anticipatory Classifier System (ACS) [Sto98] and Yet Another Classifier System (YACS) [GSS01] so as to combine generalization and latent learning. We also give an overview of the main mechanisms of YACS. In section 4 we present a kind of regularities in the transitions that neither ACS nor YACS are able to consider, and propose a new formalism to deal with that kind of regularities. In section 5 we describe MACS: a new LCS using this formalism. Experimental results are given in section 6 in order to compare it with YACS.

2 From Q-learning to Learning Classifier Systems

The first proposals for a LCS devoted to RL problems are presented in [Hol76]. The first LCS, called CS1, can be found in [HR78]. Wilson [Wil95] introduced an algorithm similar to Q-learning [Wat89] in LCSs instead of the traditional *Bucket Brigade* algorithm [Hol85]. This work led to a revival of LCS research since the accuracy based approach in XCS overcomes the over-generalization problem 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 $R : S \times A \times S \to \Re$ 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 which represents a quality function $Q : S \times A \to \Re$.

$Q(s, a)$ represents the expected payoff when the agent performs the action a in the state s, and after that follows the best policy.

At time step t, the qualities are updated according to the following formula based on the Bellman equation used in Dynamic Programming [Bel57]:

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

where s_t is the state resulting from taking the action a_{t-1} in the previous state s_{t-1}. α is the learning rate of a Widrow-Hoff delta rule[1] and γ the discount factor used in the Bellman equation [Bel57]. The effect of the Bellman equation is to make the qualities decrease as the (s, a) pairs lead to a state which is "far" from an set of distant reward sources.

The problems studied by LCSs are such that the so called *states* in the MDP framework are composed of several attributes representing perceivable properties of an environment. We prefer to speak about perceived situations rather than states. For instance, an agent in a grid world like a maze may perceive eight features, one for each adjacent cell. By doing so, a situation is an ordered set of several discrete values, one for each of the perceived attributes of the environment. The actions are also divided into attributes representing different effectors.

Thus it is possible to consider several situations within a single condition by using *don't care*[2] symbols "#". We say that it *matches* any particular value of the considered attribute. Turning an attribute into a *don't care* symbol makes the condition more general (it matches more situations). For instance, assuming that the attributes of the situations may only take values `0` or `1`, the condition `[#01]` is more general and matches the specialized situations `[001]` and `[101]`. The condition `[#0#]` matches 4 situations: `[000]`, `[001]`, `[100]` and `[101]`.

In [Lan00], Lanzi shows how it is possible to shift from a tabular representation of a 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 (or so called *classifiers*). The condition `C` may contain *don't care* symbols and match several situations, the action `A` represents the orders to the effectors of the agent, and `p` is the prediction of the long term payoff when choosing action `A` in any of

[1] 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 with the formula: $x \leftarrow (1 - \beta)x$

[2] A condition is a conjunction of attribute values. Thanks to the the *don't care* symbols, some of them may be omitted. Then set of conditions is a disjunction of conditions. Thus, a condition set is a disjunctive normal form. As a result, any boolean function of predicates over the values of the attributes may be represented as a set of conditions.

the situations matched by the condition `C`. In order to maximize the long term payoff, when it perceives a situation, the system will prefer the actions proposed by the classifiers with the highest payoff prediction `p` among the classifiers which condition matches the situation.

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

The main problem is to set `C` and `A` parts so that the *don't care* symbols are set in the right places. To do so, LCSs usually use *Genetic Algorithms* (GAs) so as 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 classic genetic operators like *crossover* [3] or *mutation*[4]. They also use a selection mechanism relying on *fitness* values: any classifier with a low fitness is removed from the set.

In traditional LCSs, the fitness of a classifier is equal to its payoff prediction `p`. This way, the classifiers with the lowest payoff prediction are removed. This is problematic because the discount factor (γ in the equation 1) has a decreasing effect on the payoff predictions as the classifiers are appliable (or *fireable*) far from an immediate reward[5]. XCS overcomes the resulting over-generalization problem by decorrelating the payoff prediction and the fitness. In XCS, three estimates are associated to each classifier instead of one:

– a payoff prediction `p`,
– a prediction error `e`
– a fitness `f` which is an inverse function of the prediction error and represents the *accuracy* of the classifier.

This way, the fitness of a classifier does not depend directly on its prediction anymore, but on its ability to predict accurately the long term payoff.

Kovacs showed in [Kov97] that XCS discovers *accurate* and *maximally general* classifiers. A classifier is said to be accurate if in every situation matched by its condition, taking the proposed action(s) always actually leads to the same expected payoff. It is said to be *maximally general* when it cannot contain any more *don't care* symbol without becoming inaccurate. Any classifier which corresponds to (s, a) pairs at different ranges of payoff is not accurate because it is too general.

Moreover, the classifier system tends to be *complete*, *i.e.* every possible (s, a) pair is covered by a classifier since the fitness relies on the accuracy of the prediction and not on the optimality of the action.

So, XCS builds a complete payoff map of the environment and takes advantage of generalization to reduce the number of classifiers. Moreover, according

[3] a new classifier is a combination of different segments of its parents

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

[5] the Bucket Brigade algorithm leads to the same problem

to the way XCS decides whether a classifier is accurate or not, generalization can only occur among (s, a) pairs at a same range of expected payoff. We will show in section 3 how adding an effect part to the classifiers leads to a different conception of generalization which allow generalizations among (s, a) pairs at different ranges of expected payoff.

3 An overview of YACS

In single step problems, the agent gets a situation from its environment, selects an action and receives a scalar reward which indicates if the problem was solved or not. But in multi-step problems, the agent needs to act more than once so as to solve the problem. Thus, in addition to a reward, the agent also gets a new situation as a result of its last action. In this kind of problems, it makes sense to use LCSs to learn to anticipate.

3.1 A formalism for modeling regularities

In order to use a LCS to learn a model of the dynamics of the environment, Holland [Hol90] proposed an implicit approach. With internal messages, it is possible to use tags that specify if a current "action" posted to a message list actually specifies an action or an anticipation. Riolo [Rio91] implemented this idea in his CFSC2 and demonstrated its latent learning capability. A more explicit linkage is used in CXCS [TB00] where probabilistically linked classifiers result in cooperations among classifiers. The linkage evolves an implicit representation of an anticipated effect by the linked successive conditions.

In contrast with all these approaches, the ALP (Anticipatory Learning Process) used in ACS [Sto98] is a further development of the *anticipatory behavioral control* theory introduced in psychology by Hoffmann [Hof93]. YACS [GSS01] uses the same idea and both systems form explicit `condition-action-effect` classifiers, noted `C-A-E`. This formalism is similar to Sutton's DynaQ+ [Sut91] approach or Drescher's `context-action-result` rules [Dre91], but with a generalization capability.

The `E` part represents the effects of the considered action in the situations matched by the condition. 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 "`#`", an `E` part is also divided into several attributes but may contain *don't change* symbols "=". Such a symbol means that the attribute of the perceived situation corresponding to the *don't change* symbol remains unchanged. A specific value in the `E` part means that the value of the corresponding attribute *changes* to the value specified in the `E` part.

For instance, let us consider the classifier `[#0#1] [0] [=10=]`. It anticipates the effects of the action [0] in 4 potential 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);
- the second attribute will change from 0 to 1;
- the third attribute will change to 1, whatever the initial value is (when the attributes may take more than two values, this case is different from the previous one);
- the last value of the attribute remains 1.

These cases are illustrated in table 1.

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

Table 1. Illustration of the anticipation mechanism in YACS

This formalism allows the classifiers to represent regularities in the interactions with the environment like for instance *"In a grid maze, when the agent perceives a wall in front of him, whatever the other features are, moving forward drives the agent to hit the wall, and no change will be perceived"*.

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. As in XCS, a classifier is said to be *maximally general* if it cannot contain any other *don't care* symbol without becoming inaccurate. But here, it is said to be *accurate* if in every situation matched by its condition, taking the proposed action always leads to the same changes in the perceived situations. The accuracy of the prediction in ACS and YACS is about the next situation and not about a payoff like in XCS.

Thus, what is called *generalization* in ACS or YACS is not the same as the generalization studied in XCS by [Lan97]. In ACS and YACS, generalization is allowed by the joint use of *don't care* and *don't change* symbols. This generalization allows to represent regularities of the transitions between successive situations and provides the system with:

- a kind of *selective attention* when some situations can be identified by paying attention to several attributes only;
- the ability of considering several situations with a single condition, so as to reduce the complexity of the model of the dynamics of the environment.

As YACS does not generalize depending on a payoff prediction, it is able to generalize over situations which are at different ranges of expected payoff. As a result, it does not make sense to store information about the expected payoff in the classifiers. The list of classifiers only models the transitions in the environment.

So as to perform reinforcement learning, YACS must deal with information about specific situations. So, it uses a set `P` of every perceived situation encountered during the lifetime of the agent[6]. This set only contains one single instance of each already perceived situation. Each situation is valued by the expected payoff when reaching the considered situation.

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

3.2 The heuristics used in YACS

In the following sections, we briefly give the main mechanisms of the *latent learning* process as it was more precisely described in [GSS01] and [GS01].

ACS and YACS both take advantage of the information provided by the succession of situations in order to drive the classifier discovering process. Therefore, they use heuristics instead of GAs, which are general but not explicitly driven by experience. YACS explores the solution space rationally, so as to be able to tackle large problems like the Sheepdog problem described in [SG01]. As discussed in [GSS01], the main difference between ACS and YACS is that YACS decorrelates the discovering of `C` and `E` parts.

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

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

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

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

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

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

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

The specialization process of YACS uses the *mutspec* operator introduced by [Dor94]. This operator selects a general feature of the C part[7] of an *oscillating* classifier, and produces one new classifier for each possible specific value of the selected feature. YACS improves the selection of the features to specialize by using the *expected improvement by specialization* estimate i_s associated to each *don't care* symbol in the C part of each classifier. This value estimates how much the specialization of the token would help to split the situation set covered by the C part into several subsets of equal cardinality.

Generalization of conditions Even if the specialization process is cautious, it may produce classifiers with a C part at a sub-optimal level of generality, in particular when YACS specializes C parts while it did not experience many possible situations. Thus, a generalization mechanism is required to backtrack on early inaccurate specializations.

Like the specialization process, the generalization uses heuristics in order to take advantage of experience to drive the process. Thus the YACS approach differs from ACS since its generalization process does not use GAs as in [BGS00].

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

These estimates are used to build new subsets of classifiers from initial subsets of classifiers with the same A and E parts. The new classifiers are more general, their number is smaller and they do not conflict with other classifiers in the system.

4 Representing more regularities of the environment

Up to that point, we described how YACS performs latent learning so as to build a model of the transitions in the environment. It builds a list of C-A-E classifier, each of them representing one or more transitions. Generalization allows to represent regularities in the dynamics of the interactions with the environment and we showed that the regularities represented in the ACS/YACS formalism are different from the one represented by the XCS formalism. Indeed, the accuracy

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

of a C-A-E classifier depends on its ability to anticipate the perceived changes in the environment, and not to predict the expected payoff.

Generalization in YACS is allowed by the joint use of *don't care* symbols in the C part and *don't change* symbols in the E part of the classifiers.

As ACS, YACS is able to determine whether a particular attribute is changing or not, but it cannot represent regularities across different attributes because it considers situations as an unsecable whole. To make this point clear, let us consider an agent in a grid maze. Each cell may be free or may contain a block. The agent perceives the eight adjacent cells (0 for a free cell and 1 for a block). The eight perceived attributes are considered clockwise, starting with the cell in front of the agent. The agent can move forward or turn 90 degrees clockwise or counterclockwise. Turning clockwise results in a two positions shift of the attributes to the left. 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 not able 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. Considering generalization thanks to *don't change* symbols forbids to represent such regularities. In the ACS/YACS formalism, the new value of an attribute only depends on the previous value of the same attribute, which is not generally the case.

So as to overcome this problem, it is necessary to decorrelate the attributes in the E parts while in ACS and YACS, the classifiers anticipate every attribute at once. To do so, we propose to build the E parts as situations, 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 his left when it perceived a wall behind, whatever the other attributes were (thanks to the "#" symbol). Moreover, it does not provide information about the new values of other attributes (thanks to the "?" symbol). Thanks to these new *don't know* symbols, a classifier may anticipate only a few attributes and the overall system gains the opportunity to discover new regularities.

Again, this proposal for a new formalism drives to a new conception of generalization. As usual, a classifier is said to be *maximally general* if it could not contain any more *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 actually leads the attributes to take the value specified in the effect part, when the attribute of the effect is not a *don't know* symbol.

As a result, the anticipating unit is not anymore the classifier but the whole LCS. Given a situation and an action, a single classifier is not able to predict the next situations. The system needs a mechanism which *integrates* the proposals of every classifier so as to build a complete situation, without any *don't know* symbol, as described in table 2.

[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. In the integration process, the LCS checks for the `E` parts of every classifier such that its `A` part matches the action and its `C` part matches the situation. The value of each attribute which is not a *don't know* symbol is a possibility for this attribute. The integration process builds all the possible anticipated situations with respect to the possible values of every attribute. If all the classifiers are accurate, this process only proposes one possible anticipation.

5 Description of MACS

As defined in section 4, an `E` part may contain several *don't know* and several specific attribute values. But we have adopted a radical 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.

In this section we detail the latent learning mechanisms of MACS, a new LCS designed to take advantage of the new formalism proposed in section 4. Most of them are very similar to those of YACS (see [GSS01] and [GS01] and do only affect the latent learning process. The policy learning process remains unchanged.

5.1 Evaluation and selection of the accurate classifiers

This part of the process is in charge of evaluating the accuracy of the classifiers and removing them if necessary. Two integer values `g` and `b` are associated to every 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 each time step.

With this information, YACS checks every classifier such that its `C` part matches with S_{t-1} and its `A` matches with A_{t-1}:

- if its `E` part matches S_t[8] then the classifier anticipated well and `g` is increased;
- if its `E` part does not match S_t then the classifier anticipated badly and `b` is increased;

A classifier which always anticipates badly after some evaluations is supposed inaccurate and is removed. This number of evaluations is a parameter of the system, noted e_r. A classifier is removed when `g`$= 0$ and `b`$= e_r$. Another parameter e_a of the system indicates how many evaluations are needed to assume that a classifier is *accurate*.

5.2 Specialization of conditions

As in YACS, a classifier which anticipates sometimes well and sometimes not is said to *oscillate* because its condition part is too much general and matches with too many situations. Thus it must be specialized.

Here again, this process is not driven by a GA but by heuristics which take advantage of estimates.

The estimates An *expected improvement by specialization* estimate i_s associated to each general attribute of the `C` part of each classifier (*i.e* to each *don't care* symbol). This value estimates how much the specialization of the token would help to split 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 when it anticipates badly, then this attribute is very relevant for distinguishing between the situations covered by the `C` part. Thus, the `C` part must be specialized according to this particular attribute, and the estimate i_s should get a high value.

In order to compute the estimates i_s, each classifier memorizes the situation $BadS$ preceding the last anticipation mistake and the situation $GoodS$ preceding the last anticipation success. Each 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 $BadS$ equals the corresponding attribute of S_{t-1}, then the corresponding estimate i_s is *decreased*;
  - if a particular attribute of $BadS$ differs from the corresponding attribute of S_{t-1}, then the corresponding estimate i_s is *increased*;

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

– if the classifier does not anticipate well, for each attribute:
  - if a particular attribute of $GoodS$ equals the corresponding attribute of S_{t-1}, then the corresponding estimate i_s is *decreased*;
  - if a particular attribute of $GoodS$ 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. The initial values are 0.5. A specialized attribute is given the same default i_s value of 0.5.

The process A classifier is said to *oscillate* when $\mathtt{g}+\mathtt{b} > e_o$ and $\mathtt{g}\times\mathtt{b} > 0$, where e_o is a parameter of the system, representing the number of evaluations necessary to declare a classifier as oscillating. The specialization process is cautious: MACS waits until all the classifiers with the same `E` and `A` parts have been identified as either oscillating or accurate[9].

At this point, the *oscillating* classifiers select together the attribute to specialize. The estimates i_s are summed among the oscillating classifiers, and the attribute with the highest sum is chosen to be specialized. The *mutspec* operator is then applied to every oscillating classifier. Some of the classifiers produced by the *mutspec* operator always anticipate badly. They are removed by the *selection of accurate classifiers* process. An new classifier which does not match with 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 (see 3).

5.3 Generalization of conditions

Even if the specialization process is cautious, it 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 coming back on early sub-optimal specializations. As the specialization process, the generalization uses estimates and heuristics in order to take advantage of experience to drive the process.

The estimates In order to compute the i_g estimates, MACS checks each time step every classifier whose `A` part matches A_{t-1} and whose `C` part does *not* match S_{t-1}.

Considering such classifiers, for each specialized attribute of 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:

[9] In YACS, the corresponding process was considering sets of classifiers with the same `C` and `A` parts. In MACS, the specialization process simply tries to distinguish between situations to determine which can be the causes of the considered effects. This is closer to the *differentiation* mechanism studied in psychology.

- 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 an i_g value of 0.5.

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

The process The generalization process considers sets of *accurate* classifiers with the same A and E parts. With such a set of classifiers, MACS builds a new set of classifiers which are more general or equal to the original ones. These new classifiers are such that they do not match situations already matched by classifiers with a different E part which does not match the considered one.

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

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

The new C part may lead to a conflict with other classifiers with the same A part but an *incompatible* E part. Two E parts are said incompatible if the values of the only non *don't know* symbols are different - *i.e* if they don't match. Two C parts are conflicting 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 3).

In this conflicting case, MACS does not add the new and general classifier to the new set, but the original one.

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

To do so, MACS checks iteratively every possible pair of classifiers. When the C part of a classifier is more general than another, it is kept and the other is removed.

This process allows to replace several classifiers with a smaller or equal number of classifiers. The C part of the new classifiers cover the same situations. Thus they do not conflict with other classifiers in the system (classifiers with incompatible E parts are not overlapping).

5.4 Transition covering

So as to be complete, a model needs to cover every encountered transition in the environment. For some reasons, the model may suffer from incompleteness :

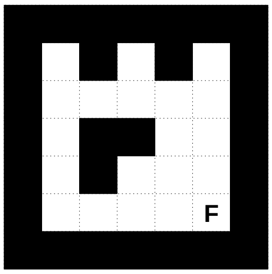

Fig. 1. The Maze228 environment

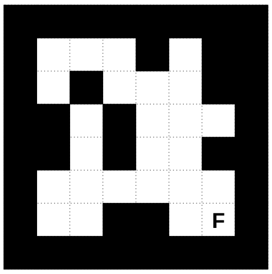

Fig. 2. The Maze288 environment

- the system is initialized with an empty list of classifiers;
- the *selection of accurate classifiers* may remove inaccurate or oscillating classifiers because of local exploration;
- the *condition specialization* process only adds a specialized classifier if it matches at least one already perceived situation. As long as the agent did not experience every possible situations, some relevant classifiers may not be added.

So, at every time step, the system covers the transition composed by S_{t-1}, A_{t-1} and S_t. For each attribute of S_t, it considers an effect such that the only specific attribute (which is not a *don't know* symbol) takes the corresponding value in S_t. Considering such an effect, among the classifiers with an A part corresponding to A_{t-1}, the system checks whether there is at least one such that its `C` part matches S_{t-1} and the `E` part equals the considered effect. If it is not the case, the system *covers* the transition by adding in the system a new classifier.

The `A` part of this covering classifier is set to A_{t-1} and its `A` part is set to the considered effect. The `C` part is such that it is as general as possible. It does not match any of the classifiers with the same `E` and `A` parts, but matches S_{t-1}.

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 this part, we provide experimental results of YACS and MACS interacting with the Maze228 and Maze288 environments described in section 6.1. The results given in section 6.2 are discussed in section 6.3.

6.1 The Maze228 and Maze288 environments

As the *Wilson woods problems* usually used as benchmarks in the LCS framework, Maze228 and Maze288 are grid worlds. Such grid worlds are intuitive representations of finite state diagrams where the situations are splitted into attributes and where transitions offer some regularities.

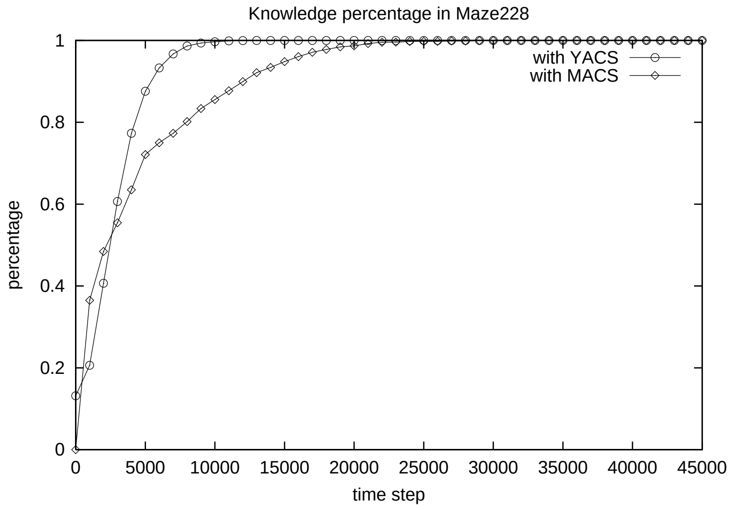

Fig. 3. Evolution of the percentage of knowledge in Maze228

Each cell of the grid may be empty or contain 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 contains nine attributes : eight corresponding to the adjacent cells and one to the cell it is situated in. An attribute may take three values: `0` (empty cell), `1` (■) or 2 (food) The attributes are ordered in a situation from the front to the back and from the left to the right.

The agent is allowed 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 hits it and remain in the same cell.

The topology of these two environments are illustrated in figures 1 and 2. Maze228 contains 19 non-final cells. The agent can be situated in these cells according to 4 directions. For each of these situations, if may choose any of 3 actions. Thus, $19\times4\times3$=228 transitions are to be modeled in Maze288. As maze 288 contains 24 non-final cells, it is possible for the agent to experience 288 transitions.

The experiments are divided into trials. The agent starts a trial in a free cell chosen randomly. A trial ends when the agent reaches the cell with food. In that case the food does not disappear, the agent gets a reward, it perceives the situation, and a new trial starts.

6.2 Experimental results

In order to estimate the evolution of the accuracy and completeness of the model over successive time steps, we use a measure of the *percentage of knowledge* provided by the model. For each possible transition in the environment, we check

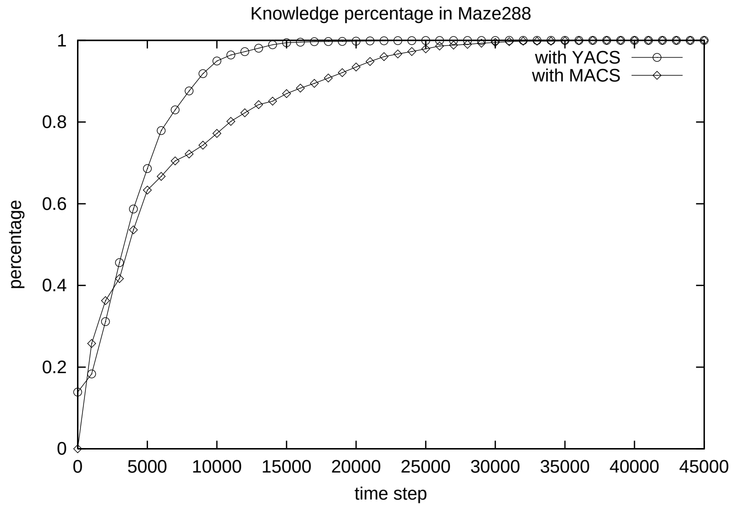

Fig. 4. Evolution of the percentage of knowledge in Maze288

if the classifier system is able to model accurately the transition - *i.e.* anticipates only one situation, which is the actual one. The percentage of knowledge is the ratio of transitions accurately modeled by the system.

As the purpose of the paper is to study only *latent learning*, the agent runs in random exploration only. All the results are averaged over 10 experiments. For YACS, as defined in [GS01], 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, e_o was set to 3, e_r to 4 and e_a to 5 (see section 5).

The figures 3 and 4 present the evolution of the percentage of knowledge when YACS and MACS interact with Maze228 and Maze288. Both systems always converged toward a complete and accurate model of the transitions.

The figures 5 and 6 present the evolution of the percentage of knowledge when YACS and MACS interact with Maze228 and Maze288. In Maze228, YACS always converged toward 199 classifiers and MACS converged to an average of 174 classifiers (minimum 170 and maximum 180). In Maze288, YACS always converged toward 249 classifiers. MACS converged to an average of 191,3 classifiers (minimum 182 and maximum 208).

6.3 Discussion

The semantics of the classifiers are different in YACS and in MACS. With the YACS formalism, the classifiers model transitions as a whole, while in the MACS formalism, each classifier only predicts the value of one single attribute. In environments with a few regularities across attributes, the number of rules discovered by MACS should be higher.

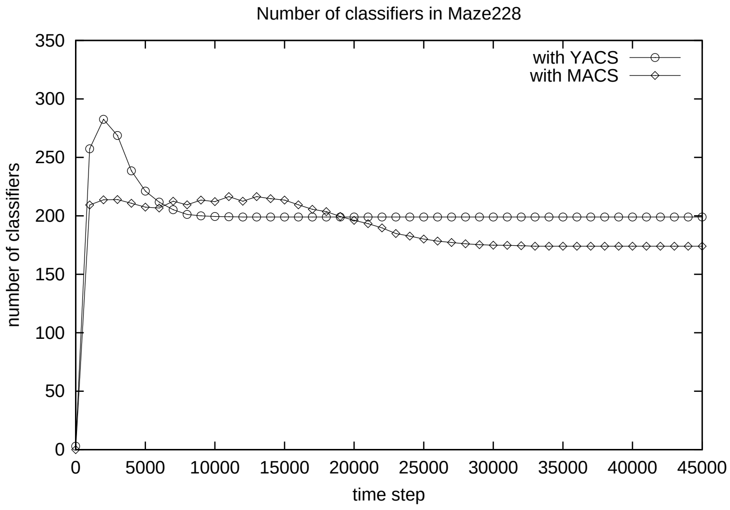

Fig. 5. Evolution of the number of classifiers in Maze228

Moreover, there are no *don't care* symbols in the formalism of MACS. As a result, regularities like "moving toward a wall doesn't make the situation change" are represented with more classifiers in MACS than in YACS : one 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 Maze288, MACS converges toward a smaller number of classifiers than YACS thanks to the new regularities taken into account in the MACS formalism. This highlights the fact that the formalism of MACS is more powerful than the previous one. Indeed, even if Maze288 is 25% bigger than Maze228, MACS only needs 10% more classifiers to model its interactions with it, while YACS needs 25% more. MACS exhibits an ability to represent many regularities which are general to the structure of every environments of the same kind:

– In MACS, every transition involving a turning action is modeled straightforwardly 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 there is no wall in front of it. This information relies on the topology of each particular maze, not on the general structure of mazes. In order to reduce the number of classifiers necessary to predict such attributes, a solution could be to provide MACS

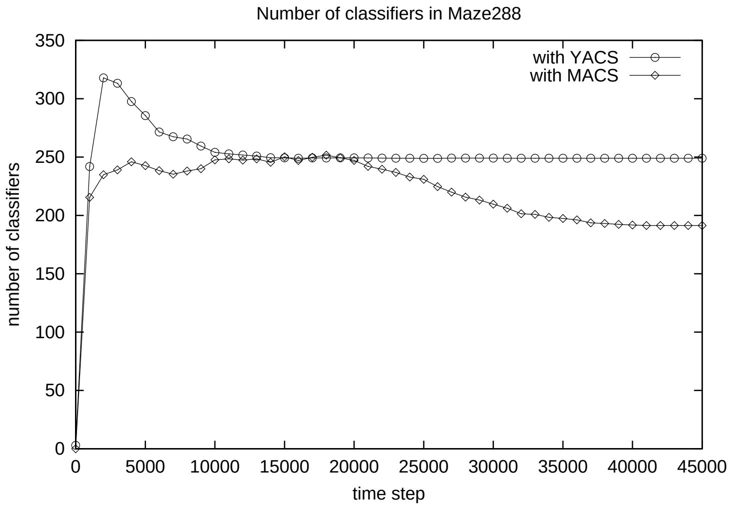

Fig. 6. Evolution of the number of classifiers in Maze288

with a mechanism which allows to build `E` parts with several specific symbols, as initially proposed in the formalism (see section 4).

The first kind of regularities are discovered quickly by MACS. Thus, in the first 5000 times steps of the experiments (in Maze228 like in Maze288), the percentage of knowledge grows very fast before slowing down. In YACS, the progression of this knowledge percentage is more regular and on the long run, the complete model is achieved more quickly. We believe that YACS is favored by its *effect covering* mechanism, but further experiences will be necessary to understand in more detail this phenomenon.

In order to speed up the convergence of MACS, we plan to use Dynamic Programming so as to drive the exploration of the environment. The active exploration we envision relies on the already accurately modeled transitions, and the reward depends on the expected information gain in each situation. This gain is a function of the accuracy of each classifier such that its `C` part matches the considered situation. With such a mechanism, the agent is able to take advantage of an incomplete model to speed up the building of a complete one.

7 Conclusion

In this paper, we described several LCSs, each of them casting a new light on the concept underlying generalization in the LCS framework. In particular, we enlighted how LCSs like XCS consider generalization with respect to an expected payoff while 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 to learn a model of the dynamics of the environment which draws benefits from generalization.

In an experimental study, we showed that the new formalism actually allows a more powerful generalization with respect to YACS, but that it takes more time to build a model. We emphasis the need of exploration strategies to overcome this problem. These exploration strategies should use MDPs on partially learned models of the environment so as to actively search relevant information.

However, YACS and MACS are still bounded to deterministic Markov problems. But we think that the absence of an effect covering mechanism in MACS should help to learn models of stochastic environments. In a short term, we envision to develop modular LCSs that take advantage of anticipation to solve POMDPs.

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.

[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.

[GS01] P. Gérard and O. Sigaud. Adding a generalization mechanism to yacs. In *Proceedings of the Genetic and Evolutionary Computation Conference 2001 (GECCO01)*, (to appear) 2001.

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

[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.

[Kov97] T. Kovacs. XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. technical report version. Technical report, chool of Computer Science, University of Birmingham, Birmingham, U.K, 1997.

[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.

[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.

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

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

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

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

[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.