YACS: a new Learning Classifier System using Anticipation

Pierre Gérard[1,2], Wolfgang Stolzmann[3], and Olivier Sigaud[1]

[1] Dassault Aviation, DGT/DPR/ESA
78, Quai Marcel Dassault, 92552 St-Cloud Cedex
[2] AnimatLab (LIP6), 8, rue du Capitaine Scott, 75015 PARIS
[3] DaimlerChrysler AG, Research and Technology, Alt-Moabit 96A, 10559 Berlin

pierre.gerard@lip6.fr
wolfgang.stolzmann@daimlerchrysler.com
olivier.sigaud@dassault-aviation.fr

Abstract. A new and original trend in the Learning Classifier System (LCS) framework is focussed on *latent learning*. These new LCSs call upon classifiers with a [*condition*], an [*action*] and an [*effect*] part. In psychology, *latent learning* is defined as learning without getting any kind of reward. In the LCS framework, this process is in charge of discovering classifiers which are able to anticipate accurately the consequences of actions under some conditions. Accordingly, the latent learning process builds a model of the dynamics of the environment. This model can be used to improve the *policy learning* process. This paper describes YACS, a new LCS performing latent learning, and compares it with ACS.

1 Introduction

The reinforcement learning framework [Sutton and Barto, 1998] considers adaptive agents involved in a sensori-motor loop. Such agents perceive situations through their sensors, and use these perceptions to select 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). This way, one can for instance simulate rats whose task is to learn the shortest path to the food.

Holland [Holland, 1976] presented the first ideas about LCSs (Learning Classifier Systems) designed to solve reinforcement learning tasks. The capability of generalizing while learning is the main advantage of LCSs with respect to other reinforcement learning systems like *Q-learning*[Watkins, 1989]. It allows to aggregate several situations within a common description so that the representation of the problem gets smaller. The first LCS, called CS1, can be found in [Holland and Reitman, 1978]. Wilson [Wilson, 1995] introduced an algorithm similar to Q-learning [Watkins, 1989] in LCSs instead of the traditional *Bucket Brigade* algorithm [Holland, 1985]. This work led to a revival of LCS research since the accuracy based approach in XCS overcomes the problem in previous LCSs where especially deferred reward leads to over-generalization [Wilson, 1989].

Additional to the generalization capabilities of LCSs in policy learning tasks, an internal model of the dynamics of the environment can be used to adapt the policy further and faster. In multi-step problems, the consequence of an action does not only consist in a reward, but also in the resulting new situation. In problems of that kind, an agent has the opportunity to consider two successive perceived situations. Thus it can learn to *anticipate* what happens immediately after the execution of an action. This learning process builds a model of the dynamics of the environment. Such a model endows the system with information about situation transitions and allows lookahead mechanisms. These mechanisms can be used either for planning or for hypothetical acting so as to speed up the policy learning process. Defining subgoals also allows to anticipate and to plan [Donnart and Meyer, 1996] what will happen far into the future, but our approach considers the *anticipated immediate effects* of an action.

The notion that the formation of action-effect relations is at the core of the acquisition of behavioral knowledge and the insight that anticipations are necessary for behavior reach far back in psychology. According to James [James, 1890], *"an anticipatory image [...] is the only psychic state which introspection lets us discern as the forerunner of our voluntary acts."*. In animal learning, Tolman [Tolman, 1932] proposed that learning is the process of discovering what leads to what (*i.e.* animals develop a "cognitive map", a sort of internal representation of the world). Seward [Seward, 1949] gives empirical results which showed that rats are indeed learning an internal representation of an environment without receiving any type of reward or punishment. Learning without environmental reward or punishment is called *latent learning*. Hoffmann [Hoffmann, 1993] proposed a psychological learning theory of *anticipatory behavioral control*. He postulates that conditional action-effect relations are learned latently by using anticipations.

In order to use LCS to learn a model of the dynamics of the environment, Holland [Holland, 1990] 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 [Riolo, 1991] implemented this idea in his CFSC2 and demonstrated its latent learning capability. A more explicit linkage is used in CXCS [Tomlinson and Bull, 2000]. 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 Anticipatory Learning Process (ALP) used in the ACS (Anticipatory Classifier System) is a further development of the *anticipatory behavioral control* theory introduced in psychology by Hoffmann [Hoffmann, 1993]. YACS (Yet Another Classifier System) also uses these ideas and both systems form explicit [*condition*][*action*][*effect*] classifiers. This formalism is similar to Sutton's DynaQ+ [Sutton, 1991] approach or Drescher's [*context*][*action*][*result*] rules [Drescher, 1991], but with a generalization capability. ACS and YACS both take advantage of the information provided by the succession of situations in order to drive explicitly the classifier discovering process. Therefore, they use heuristics instead of genetic algorithms, which are general but not explicitly driven by experience.

In section 2 we detail the heuristics used for the latent learning process in YACS. In section 3 we show how this system takes advantage of the model computed by the latent learning process to learn an optimal policy. Some experimental results are presented and discussed in section 4. In section 5 we show the main differences between ACS and YACS.

2 Description of the latent learning process in YACS

As ACS [Stolzmann, 1998], YACS deals with [*condition*][*action*][*effect*] classifiers, or C-A-E classifiers. C stands for [*condition*], A for [*action*] and E for [*effect*]. C parts take advantage of generality and may match several perceived situations. An A part specifies a particular action possible in the environment. 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.

A situation is divided into several features representing perceivable properties of an environment. For example, an agent in a grid world may perceive eight features, one for each adjacent cell. Thus, a situation is an ordered set of several discrete values, one for each of the perceived features of the environment. A C part has the same structure but it may contain *don't care* symbols "#". Such a symbol matches every specific value of the corresponding features of a perceived situation. The E part stores for each perceived feature the expected changes in the environment when the action of the classifier is chosen and when the perceived situation matches its condition. A specific value in the E part means *"if the classifier is fired, the feature of the perceived situation corresponding to the specific value will change and turn to that value at the next time step"*. The E part might contain *don't change* symbols "#". A *don't change* symbol in the E part means *"if the classifier is fired, the feature of the perceived situation corresponding to the* don't change *symbol will remain unchanged at the next time step"*.

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. It learns C and E parts separately.

In section 2.1, we explain how YACS learns E parts by direct comparison of successive perceived situations. Moreover, we present how it selects accurate classifiers in section 2.2. Finally, we detail in section 2.3 how YACS learns relevant conditions by successive specializations.

2.1 Effect covering

The effect covering mechanism is the part of the latent learning process that is in charge of discovering accurate E parts (*i.e.* E parts representing actual effects of actions under some conditions). When the system learns accurate effects, it creates new classifiers with suitable E parts settled according to experience. During this 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[1] list with a finite length m.

YACS 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 computes the desired effect DE which is the E part of a classifier which could have been fired at the preceding time step, and whose E part reflects accurately the changes actually perceived in the environment. To compute the desired effect, YACS uses the *diff* operator which works as follows on features of the environment:

$$diff(f_t, f_{t-1}) = \begin{cases} \# \; if \; f_t = f_{t-1} \\ f_t \; otherwise \end{cases}$$

where f_t is a feature of the situation at the current time step, and f_{t-1} is the corresponding feature of the situation at the previous time step.

The *diff* operator is applied to each feature of the situations S_t and S_{t-1} to compute the desired effect DE.

At each time step, YACS checks the accuracy of the E part of every classifier of the action set (*i.e.* the set of classifiers which have been matched by S_{t-1} and whose actions are the same as the selected action A_{t-1}). Such a classifier should have an E part equal to the desired effect DE:

- If its E part equals DE, the classifier would have anticipated well, and we add a *good* marker to its trace T of anticipation mistakes and successes.
- In the other case, the E part of the classifier is wrong and we add a *bad* marker to its trace T.

Moreover, if no classifier has a correct E part, YACS chooses one of the classifiers whose E part is wrong. Then it builds a new classifier with the same C and A parts. Its E part is set to DE. As it would have anticipated well, a *good* marker is added to its empty trace T. This new classifier is finally added to the classifier set.

2.2 Selection of accurate classifiers

As YACS tries to build a set of classifiers that anticipate accurately, it has a delection 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.

[1] First In First Out: the first element added in the list is the first removed

– If the trace T of a classifier is full and if it contains *good* and *bad* markers, we say that the classifier *oscillates*: it sometimes anticipates well and sometimes not. In Markov and deterministic environments, the reason of these oscillations is that its condition is too general. It matches several different situations, each leading to a different situation just after the action. In order to distinguish between these situations, the condition must be further specialized.
– In all other cases, the classifier is kept.

This mechanism allows to keep only the accurate classifiers.

2.3 Specialization of conditions

In section 2.1 we have presented how YACS learns E parts. However, the anticipation of a classifier may only be accurate if its C part is at least partly specialized. A C part should be as general as possible in order to represent regularities in the environment. But it must be specific enough so that the classifier does not oscillate. This section explains how C parts are incrementally specialized so as to reach the right level of generality.

The MutSpec operator The classifier discovery problem is usually solved by a genetic algorithm using a creation process driven by mutation and crossover on classifiers selected according to their fitness. These operators do not explicitly take advantage of the experience of the agent.

As in the U-Tree algorithm [McCallum, 1996], 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 [Dorigo, 1994]. This operator selects a general feature of the C part[2] of a classifier, and produces one new classifier for each possible specific value of the selected feature. The E parts of every new classifier are the same as the E part of the original classifier, except when the specialization leads to an equality between the feature of the C part and the corresponding feature in the E part. In that case, a *don't change* symbol is added. The original classifier is discarded.

For instance, when the first feature is selected, and assuming that it might take only two specific values (`0` or `1`) the classifier `[#|#|#|#] [0] [#|#|#|#]` produces two new classifiers (`[0|#|#|#] [0] [#|#|#|#]` and `[1|#|#|#] [0] [#|#|#|#]`).

The C parts of the new classifiers are more specialized than the original C part.

Therefore, if the C part of the original classifier was matching several perceived situations, each resulting C part will match a subset of these situations. We want YACS to be able to choose the token to specialize in such a way that

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

the two resulting subsets have an equal cardinality, in order to prevent over-specialization.

The *expected improvement by specialization* estimates Choosing at random the token to specialize, as in Dorigo's original work, would lead to an over-specialization of the C parts and thus to a sub-optimal number of classifiers. We improve this selection process by using the *expected improvement by specialization* estimate i_s associated to each general feature 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 sub-sets 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 feature of the situation when the classifier anticipates well is always different from the value when it anticipates incorrectly, then this feature is very relevant for distinguishing between the situations covered by the C part. Thus, the C part must be specialized according to this particular feature, 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 the classifier belongs to the action set of last time step, S_t allows to check the accuracy of the E part.

- If the E part is *correct*, for each feature of the environment:
  - if a particular token of $BadS$ equals the corresponding feature of S_{t-1}, then the corresponding estimate i_s is *decreased* using a Widrow-Hoff[3] delta rule;
  - if a particular token of $BadS$ differs from the corresponding feature of S_{t-1}, then the corresponding estimate i_s is *increased* using a Widrow-Hoff delta rule;
- If the E part is *incorrect*, for each feature of the environment:
  - if a particular token of $GoodS$ equals the corresponding feature of S_{t-1}, then the corresponding estimate i_s is *decreased* using a Widrow-Hoff delta rule;
  - if a particular token of $GoodS$ differs from the corresponding feature of S_{t-1}, then the corresponding estimate i_s is *increased* using a Widrow-Hoff delta rule;

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

When a classifier sometimes anticipates well and sometimes not (*i.e.* when its anticipation trace contains *good* and *bad* markers), it oscillates and its C part

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

needs to be further specialized. If a classifier oscillates that way, thanks to the *effect covering* mechanism, the classifier set contains at least one other classifier with the same C and A parts and with a different E part.

The specialization process is cautious: YACS waits until all the classifier with the same C and A parts have been identified as *oscillating* classifiers, and until their anticipation traces are full. At this point, these classifiers select together the feature to specialize. The estimates i_s corresponding to each feature of the environment are summed among the classifiers, and the feature with the highest sum is chosen to be specialized. The *mutspec* operator is then applied to every classifier. Some of the classifiers produced by the *mutspec* operator always anticipate badly. They are removed by the *selection of accurate classifiers* process.

2.4 Useless classifiers and Condition covering

At this point, we have described the main mechanisms of the latent learning process. The mechanisms described in this section are devoted to deal with the useless classifiers and the covering of new situations.

Condition covering Given a particular possible action, when YACS gets a new situation S_t, it may happen that this situation does not match with any C part of the classifier set. In this case, YACS creates a new classifier. Its A part equals the considered action. The E part of the classifier is computed according to S_{t-1} in the same way as the desired effect (see section 2.1). Its C part is such that it matches S_t. It is also such that it is neither more general nor more specific than the C part of any other classifier with the same A part. With respect to the previous constraints, it is as general as possible. These C parts allow to add maximally general classifiers without introducing redundancies with already specialized ones.

Useless classifiers The system uses a set P of every perceived situation encountered during the lifetime of the agent. This set only contains one single instance of each situation. It is not ordered. When the agent comes to time step t, it perceives the new situation S_t. If the new situation is not present in P, it is added.

The specialization process may create classifiers which will never be used nor evaluated because their C part does not match any possible situation. To get rid of such classifiers, we remove every classifier which does not match any situation in the set P of already encountered situations.

The number of elements in P grows exponentially with respect to the number of perceived features in the environment. In huge environments, memory problems may occur and it would be worth taking advantage of generalization in order to reduce the size of this set. But in section 3 we show that the system must deal with information about specific situations in order to use dynamic programming. So this set P is also necessary for another part of the algorithm.

Until now, we have described the latent learning process in YACS. Thanks to effect covering and incremental specialization of conditions, it provides the system with a set of classifiers which anticipate well the changes in the environment. This set of classifiers is a model of the dynamics of the environment in terms of situation transitions. The differences with the latent learning process in ACS are discussed in section 5.3.

3 Description of the policy learning process in YACS

In order to take advantage of the model of the dynamics of the environment provided by the ACS, Butz and Stolzmann [Butz and Stolzmann, 1999] use goal directed planning. This solution is expensive in computational time. Moreover, it is not part of the reinforcement learning framework. It optimizes the model learning capabilities but does not improve the policy learning process.

YACS uses iterative algorithms derived from dynamic programming. Plain *value iteration* (see section 3.1) performs several steps of Value Iterations in order to adjust the qualities for action of each $(situation, action)$ pair. Using one single step of value iteration at each time step allows to find the optimal policy over several time steps and offers a good reactivity/planning tradeoff. Doing so is similar to performing "mental" actions as in DynaQ+ [Sutton, 1991].

In this section, we describe how YACS takes advantage of the model of the environment in order to speed up the reinforcement learning process and the computation of a policy. We first introduce the Value Iteration algorithm and the way YACS identifies the reward sources. Then we explain how the use of generality forbids to compute one single quality of action for each classifier[4]. We finally present how this problem is solved in YACS.

There are many ways for taking advantage of the model for performing reinforcement learning. We choose the algorithms presented in this section because they are among the most classical available.

3.1 Value Iteration

In order to compute an optimal policy, YACS uses a simplified variety of Value Iteration: a dynamic programming algorithm which solves the Bellman equations [Bellman, 1957]. It iteratively refines the qualities for every $(situation, action)$ pair using the formula:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s') \quad (1)$$

where

$$V(s) = max_a Q(s,a) \quad (2)$$

[4] Stolzmann [Stolzmann, 2000] also uses the term qualities but with a different meaning. The qualities of action are different from the qualities used in ACS, which are qualities of anticipation. Qualities of anticipation estimate the confidence in the anticipation, and not an expected discounted reward as the qualities of action.

$Q(s,a)$ is the quality of action a in situation s. It takes into account both the immediate and the future expected reward, discounted by a temporal discount factor γ. $V(s)$ is the desirability value of the situation s. $R(s,a)$ is the immediate expected reward when the agent performs action a in situation s. $T(s,a,s')$ is the probability to perceive the situation s' just after performing action a in situation s.

As YACS is designed to deal with deterministic environments, we do not use the transition probabilities and replace the expected future cumulative reward $\sum_{s'} T(s,a,s')V(s')$ by $max_{s'} V(s')$ where s' is a situation anticipated when the agent performs the action a in situation s.

3.2 Learning immediate rewards

The latent learning process brings information about situation transitions. In order to use a dynamic programming algorithm, YACS computes the immediate expected rewards corresponding to the $R(s,a)$ term in value iteration. An immediate reward estimate r is associated to each classifier. It estimates the immediate reward received by the agent if the action which it proposes is chosen when its C part matches the current situation.

Let us call R_t the reward given by the environment resulting from the action A_t when the perceived situation was S_t. The immediate reward estimate r of every classifier such that its C part matches S_t and its A part matches A_t is updated according to a Widrow-Hoff delta rule: $r \leftarrow (1-\beta)r + \beta R_t$

3.3 The problem induced by general conditions

In classifier systems without E parts, a classifier is kept when it helps to maximize the reward on the long run [Wilson, 1994], or when it is able to predict the reward accurately [Wilson, 1995]. But we use classifiers with an E part. In our case the decision to keep or to remove a classifier only relies on its ability to predict the next situation. Thus the fitness of a classifier does not take the reward into account (see section 2.2 about *latent learning*).

This way of estimating the fitness of a classifier gives rise to a new way of considering generality. A classifier is too general when a *don't care* symbol prevents the anticipation to be accurate, regardless of the expected reward. It is too specific if its E part would remain accurate if some *don't care* symbols were added in its C part, regardless of the payoff.

Let us consider an agent in a maze like the woods environments (see section 4.1). When it perceives an obstacle in a particular direction, and if it moves towards this direction, it will hit the obstacle and remain in the same cell. The agent does not need to pay attention to the obstacles in the other directions to anticipate well what will happen. Thus a classifier which could be interpreted in a maze environment as *"when the agent perceives an obstacle on the north, if it tries to move north, it will not perceive any changes in its situation: it will remain in the same cell"* anticipates well. It is accurate since it would not anticipate better if its C part were more specific. It is kept and will not be further specialized.

The problem is that such an accurate classifier is not too general with respect to anticipation, but it is too general to predict the discounted reward since its C part matches several different situations. Because of the discount factor γ used in Value Iteration, the desirability values associated to the situations which are close to a reward source are higher than the desirability values associated to the situations which are far from it. Therefore, it is not possible to compute a single quality of action for a classifier whose general C part matches several situations perceived at different distances from the reward.

To summarize, general C parts help to express regularities in the environment instead of simply providing a kind of selective attention. As a result, some classifiers may match several situations and thus it becomes impossible to compute a single quality of action for each general classifier. The system needs to deal with information about specific situations.

3.4 Associating desirability values to every already encountered situation

Let us tackle the problem induced by general conditions in another way. In YACS, a classifier stores at least one possible transition between two situations when an action is performed. The latent learning process in YACS discovers classifiers such that one classifier represents several transitions involving the same action, but matching in different actual situations. To do so, YACS makes use of general C parts and stores the changes resulting from the action in the E parts. Thus the system looses the information about the transitions between actual specific situations by aggregating them in a single classifier with a general condition. This loss of information about actual specific situations forbids to use Value Iteration to take advantage of the model. It also forbids to perform "mental" actions as [Sutton, 1991] does in DynaQ+.

In order to recover the missing information, YACS associates a desirability value to each specific situation in the already perceived situation set P (see section 2.4). This desirability value corresponds to the $V(s)$ term in formula (2). The transitions are consequently stored and the immediate rewards are stored in the classifiers and the missing information is stored in the already perceived situation set. The resulting model is much smaller than a Q-table storing qualities for every $(situation, action)$ pairs.

To compute the desirability value associated with a situation S, we first identify all the classifiers whose C part matches the situation S. If such a classifier was fired, the immediate reward would be stored in its r estimate and the expected cumulative reward would be the maximum among the value of the situations anticipated by the firable classifiers, multiplied by the discount factor γ.

In order to determine the situation S_{t+1} anticipated by a classifier given a situation S_t, YACS uses the *passthrough* operator. The anticipated situation is

$S_{t+1} = passthrough(S_t, E)$ where E is the effect of the classifier. This operator works as follows on the features of the situations:

$$passthrough(f_s, f_e) = \begin{cases} f_s \ if \ f_e = \# \\ f_e \ otherwise \end{cases}$$

where f_s is a feature of the situation S_t and f_e is the corresponding feature of the E part of the classifier.

So the system computes values for each situation and takes advantage of the model supplied by the latent learning process to update these values without actually performing the actions. When the classifier system anticipates well in every situation, the agent is able to quickly adapt its behavior to new reward sources.

3.5 Selecting an action

When YACS perceives a situation S_t from the environment, it selects all the classifiers whose C part matches. These classifiers anticipate the following situation by computing $passthrough(S_t, E)$. Then they compute a quality of action by adding the immediate reward estimate r and the discounted expected reward which is the value of the following situation multiplied by the discount factor γ. The selected action is the action of the classifier corresponding to the highest computed quality of action.

4 Experiments with YACS

In sections 2 and 3, we have presented YACS in two parts. The latent learning process is in charge of building a model of the dynamics of the environment. The policy learning process takes advantage of the model to compute the optimal policy. This section presents experimental results of YACS solving *Wilson woods problems*. The simulated woods environments are described in section 4.1.

The latent learning and policy learning processes both take place in a general reinforcement learning task. In section 4.2 we present the experimental results for the general reinforcement learning task when YACS interacts with Maze4 and Maze6. In section 4.3, we present experimental results specifically dedicated to the latent learning process.

4.1 The Maze4 and Maze6 woods environments

In woods environments, the agent is situated in a maze cell and perceives the eight adjacent cells. A cell can either be empty, or contain an obstacle (■) or food (F). It can move towards any of these cells. The agent is allowed to try to move towards an obstacle. In this case, it will remain in the same cell.

Maze4 (see figure 1) has been investigated by [Lanzi, 1997] and Maze6 (see figure 2) by [Lanzi, 1999]. The experiments we present in this paper involve YACS interacting with these environments.

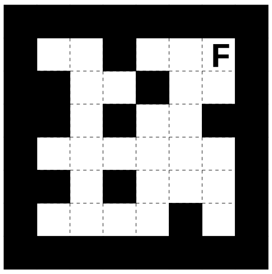

Fig. 1. The Maze4 environment

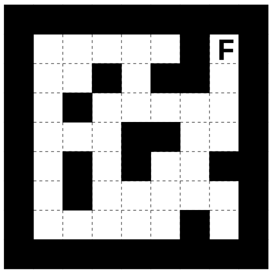

Fig. 2. The Maze6 environment

The experiments are divided into trials. The agent starts a trial in a free cell chosen randomly. This random choice makes the exploration easier and allows to check if the agent learns to reach the goal from any position in the maze. A trial ends when the agent reaches the cell with food. In that case the agent gets a reward of 1, it gets a new perceived situation, and a new trial starts.

Since the agent starts a trial in a random cell, the optimal number of actions to reach the food is not constant over trials. Averaged over every possible starting cell, the optimal number of actions is 3.5 in Maze4 and 5.2 in Maze6. As we let the agent perceive the situation in the cell with the food, the number of time steps is one more than the number of actions. Thus, the average optimal path is 4.5 time steps long in Maze4 and 6.2 in Maze6.

In our experiments, the agent is allowed to perform any action in every situation. As a result, YACS has to discover classifiers which model the transitions which do not lead to any change. This is the case when an action leads the agent to hit an obstacle and remain in the same cell. There are respectively 93 and 135 transitions of that kind in Maze4 and Maze6. By taking advantage of generality, these transitions can be modeled with 8 classifiers in each case: one classifier for each possible action, by paying attention to the presence of a block in the direction corresponding to the action. There are no other regularities in Maze4 and Maze6. Since the total number of possible transitions is 208 in Maze4 and 288 in Maze6, the optimal numbers of classifiers YACS should reach are respectively 123 $(208 - 93 + 8)$ and 161 $(288 - 135 + 8)$ for Maze4 and Maze6.

ACS [Butz et al., 2000b] has also been tested in Maze4 and Maze6 environment. In these experiments, the agent was also allowed to perform *a-priori* inefficient actions, but in ACS, the classifiers which do not model a change are deleted in the long run. Under these conditions, 115 $(123 - 8)$ classifiers can accurately model the dynamics of the Maze4 environment and 153 $(163 - 8)$ classifiers are needed for Maze6.

4.2 Reinforcement learning

In sections 2 and 3, we presented YACS in two parts. The latent learning process is in charge of building a model of the dynamics of the environment. The policy learning takes advantage of the model to compute the optimal policy.

Both processes take place in the general reinforcement learning framework. In a first part, we present experimental results which show how the generalization problem is solved by the use of desirability values associated to each specific situation. However, these experiments show limitations which are due to the exploration/exploitation tradeoff in YACS. These problems are discussed in a second part.

Experimental results The agent explores the environment by random walk until the trace T of each classifier contains only *good* markers. At this point, YACS assumes that the dynamics of the environment is well known. The agent does not move randomly anymore and the system switches into exploitation mode[5].

For every experiment, the learning rate parameter is set to $\beta = 0.1$ and the classifier's memory size is set to $m = 5$. As we do not tackle the exploration/exploration tradeoff in these experiments, the discount factor γ does not matter, excepted for float precision concerns. We set it to 0.9.

Figures 3 and 4 present the number of time steps needed by the agent to finish one trial. The results are averaged over 10 experiments. Figures use a logarithmic scale for the y axis.

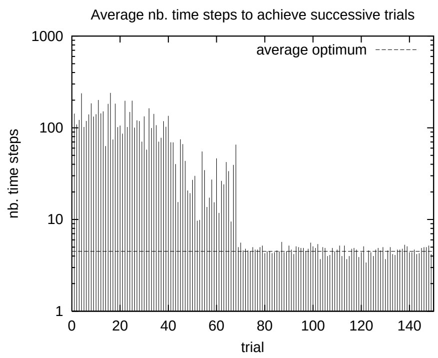

Fig. 3. Average number of time steps to finish successive trials in Maze4

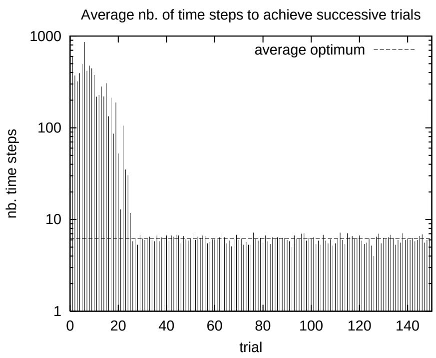

Fig. 4. Average number of time steps to finish successive trials in Maze6

When the memory size m is smaller, the specialization process is less cautious, sub-optimal specializations occur more often in early time steps and thus, the system converges towards a higher number of classifiers. When the memory size m is higher, the bad classifiers are removed after a longer time and thus, the maximal number of classifier ever reached in the experiments increases.

[5] In the environments discussed here, it's sufficient to switch from exploration to exploitation. But it in environments that change after a long time, it might be necessary to switch back from exploitation to exploration. This could also be done in dependence of the traces.

Maze6 is a larger environment than Maze4 and the average optimal path to the food is longer. Thus the number of time steps to reach the food in random mode is larger in Maze6. That is why YACS needs less trials to build a model of the dynamics of the Maze6 environment, but each trial is longer in terms of time steps.

Figures 5 and 6 show the number of time steps needed by the agent to finish successive trials for a typical single experiment. As long as the system is in

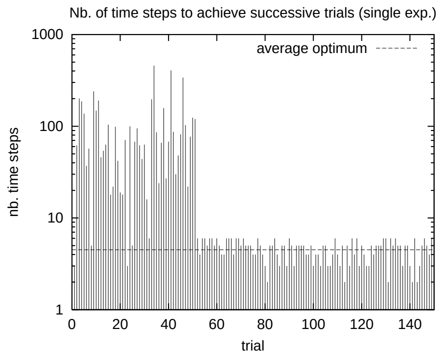

Fig. 5. Sample number of time steps to finish successive trials in Maze4

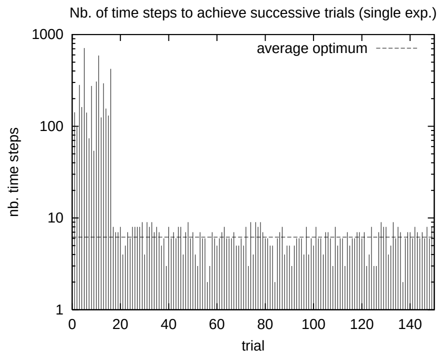

Fig. 6. Sample number of time steps to finish successive trials in Maze6

exploration mode, the number of time steps remains high. As soon as the system switches to exploitation mode, the behavior becomes optimal with respect to the quality of the model of the environment. The number of time steps in later trials is not constant because each trial starts in a random cell and thus, the optimal path to the food does not always have the same length.

In the Maze6 experiments, the system always discovers the shortest path to the food. In one of the experiments in the Maze4 environment, the behavior was sub-optimal. This problem is discussed in the next section.

Discussion In LCSs with anticipation capabilities, the main concern is usually the latent learning process. In [Butz et al., 2000b] for example, the policy learning process does not take advantage of the model and uses a modified Q-learning technique. Thus, the agent must actually act in the environment in order to discover the optimal actions.

When they take advantage of the model to speeds up the policy learning process, ACS and YACS [Stolzmann et al., 2000] face the same problem: if the model is not accurate, the agent exhibits a sub-optimal behavior. This is the case in our experiments with Maze4. The average number of time steps needed by YACS to finish a trial converges to 4.75 instead of 4.5. The problem comes from the inaccuracy of the criterion used by the system to switch from exploration mode to exploitation mode. In one of the 10 trials, every classifier discovered by the latent learning process only contained *good* markers, but the representation was not perfectly accurate because some classifiers were still over-general. As a result, the behavior of the agent was sub-optimal in the exploitation phase.

During the early time steps, the system contains a lot of over-general classifiers. Such classifiers match different situations and may let the agent consider transitions which cannot actually be encountered in the environment. When the policy learning process relies on such a model, the agent might repeat endlessly the same sequence of actions.

In order to avoid such problems, an LCS with latent learning abilities needs specific mechanisms. In DynaQ+, [Sutton and Barto, 1998] uses an exploration heuristic and gives an exploration bonus to the qualities for $(situation, action)$ pairs. This bonus increases as the pairs have not been used for a long time. This way, actions that improve the model are encouraged and the system draws the benefits of a good exploration/exploitation tradeoff.

Such algorithms may be adapted to systems like ACS and YACS, which take advantage of the generality to express regularities in the environment, contrarily to DynaQ+.

In this section, we presented some experimental results for general reinforcement learning tasks with YACS. Because of the problem induced by general C parts (see section 3.3), one cannot compute a single quality of action for each classifier. The system must deal with information about specific situations. Without the desirability values associated to each specific situation, YACS would not be able to compute an optimal policy at all. Our experiments with Maze4 and Maze6 show that the mechanism we propose solves the problem.

4.3 Latent learning

In order to solve reinforcement learning problems, YACS performs latent learning and policy learning. In this section we present experimental results concerning the latent learning efficiency in YACS.

The latent learning process is in charge of building a small and accurate model of the dynamics of the environment. Thus the system should build an accurate model with as less classifiers as possible, by discovering maximally general classifiers. In order to estimate the evolution of the accuracy of the model over successive time steps, we use a measure of the *percentage of knowledge* provided by the model. This measure is similar to the one used by [Butz et al., 2000b]. For each possible transition in the environment, we check if the classifier system contains at least one *reliable* classifier able to model the transition. In YACS, we say that a classifier is *reliable* if its trace T is full and never contained *bad* markers (*i.e.* if it always anticipated well[6]). The percentage of knowledge is the ratio of possible transitions covered by a *reliable* classifier. This percentage is 1

[6] In ACS, a classifier would be said to be *reliable* if its quality is higher than 0.9. As the criterion to decide whether a classifier is reliable or not, the percentage of knowledge measure is not perfectly identical in both systems.

when the effects of every (*situation*, *action*) pairs is accurately anticipated by the model. It is 0 when the consequences of no (*situation*, *action*) pair is well anticipated.

As in the previous section, the parameters are set to $\beta = 0.1$, $m = 5$. All the results are averaged over 10 experiments.

Experiments with Maze4 and Maze6 Figure 7 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments. Figure 8 shows the same information for the experiments with the Maze6 environment.

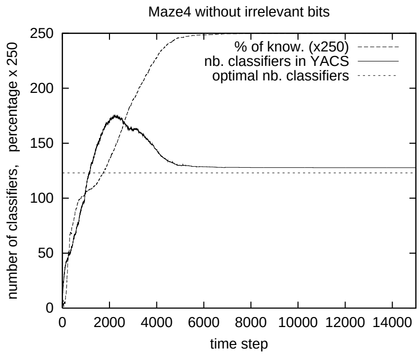

Fig. 7. Evolution of the number of classifiers in Maze4

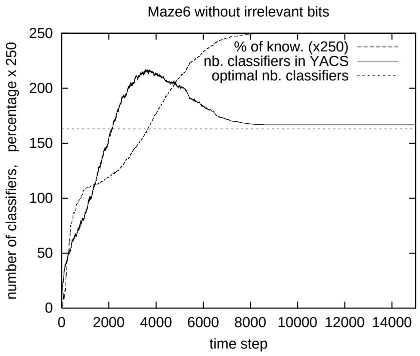

Fig. 8. Evolution of the number of classifiers in Maze6

The average number of classifiers discovered by YACS converges towards 127.7 for Maze4 and 166.7 for Maze6, which are respectively 102 and 103 percent of the optimal numbers of classifiers. As YACS does not use any generalization mechanism yet, it cannot reconsider early bad choices of the specialization process. Thus, irrelevant specializations in the early time steps may lead the system to over-specialization.

During the first time steps (up to 500), the percentage of knowledge grows very fast. YACS tries to model the transitions which do not lead to any change in the situation. Thus, YACS discovers quickly classifiers like *"when there is an obstacle in the north and I try to go north, nothing changes in the environment"*. Such classifiers are very easy to discover, since they require only one specialization of the conditions. Moreover, they can be applied in many situations in the environment. As a result, when one is discovered, the number of (*situation*, *action*) pairs which are anticipated well grows significantly. The other relevant conditions need more specializations to be discovered. Thus, the percentage of knowledge grows slower.

One can notice that the evolution of the number of classifiers is not monotonic. In a first part, YACS creates new classifiers by effect covering or condition specialization. Some of the classifiers created by the *mutspec* operator anticipate badly and need time to be evaluated and removed from the set. During the second part of the evolution, the main activity of YACS is to remove such classifiers, until the number of classifiers stabilizes close to the optimum.

Adding irrelevant information Considering a real robot that is equipped with different kinds of sensors, perceptions are normally not as accurate as in the maze environments. For example, the robot could detect different degrees of brightness in its environment. Therefore, the trials in one experiment can occur under different light conditions. For a system without any generalization capability this would result in a new perceived situation, each time the light conditions change. Here we show that both ACS and YACS are able to handle such irrelevant perceived features.

In order to simulate such a light scenario we introduce some further features into the perceived situations. These attributes are randomly set to 0 or 1 when a new trial starts and keep the same value during the whole trial. As the added perceived features are irrelevant to distinguish between situations, the optimal number of classifiers remains the same when irrelevant bits are added (see section 4.1).

Figure 9 presents the evolution of both the number of classifiers and the percentage of knowledge for the Maze4 experiments when 3 irrelevant bits are added. Figure 10 shows the same in the Maze6 environment.

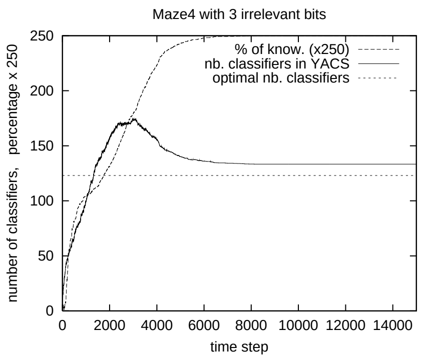

Fig. 9. Evolution of the number of classifiers in Maze4 with 3 irrelevant bits

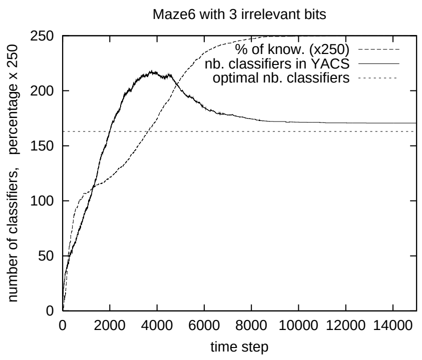

Fig. 10. Evolution of the number of classifiers in Maze6 with 3 irrelevant bits

The average number of classifiers converges towards 133.3 for Maze4 and 170.6 for Maze6, which are respectively 108 and 106 percent of the optimal numbers of classifiers. When these results are compared with the results presented in figures 7 and 8, one can notice that YACS is a bit more robust with respect to irrelevant bits in Maze6 than in Maze4. As Maze6 is larger than Maze4, a randomly walking agent takes a longer time to finish a trial in Maze6. Thus, the agent remains a longer time in situations where the irrelevant bits do not change. The *expected improvement by specialization* estimate is disturbed in more occasions in small environments like Maze4 and works better in larger environments like Maze6.

[Butz et al., 2000b] presents similar results for ACS in Maze4 with three irrelevant bits. YACS needs an equivalent number of time steps to stabilize the number of classifiers with respect to ACS without its generalization mechanism. However, the number of classifiers discovered by ACS without the genetic algorithm is much higher than in YACS. When ACS uses the generalization mechanism, it takes a longer time to stabilize the number of classifiers, and this number still remains higher than the number of classifiers discovered by YACS. This difference is discussed in section 5.3.

5 Comparison between YACS and ACS

In the previous sections we presented YACS. In this section, we point out the main differences between ACS and YACS. We assume that the reader is familiar with the ACS classifier system and we do not give an introduction here.

In section 5.1 we present problems which are already tackled by ACS and not by YACS. In section 4.3, we have shown that the number of classifiers produced by ACS and YACS are different. In section 5.2 we show that this difference does not come from the meaning of the classifiers. Thus, in section 5.3, we focus on the main differences between the classifiers discovery processes in ACS and YACS. In particular, we enlight how YACS decorrelates C and E parts more than ACS does, and how it is more cautious in the specialization process.

5.1 Extent of tackled problems

The first difference between ACS and YACS is the extent of problems tackled by the two system.

ACS is able to deal with non-Markov problems while YACS is not. In such problems, the perceived situation of the agent does not bring enough information to discriminate between different states of the environment. The agent perceives the same situations in different states. These situations are said to be *aliased*. The agent must deal with internal states in order to distinguish between such situations and be able to decide the optimal action in each actual state. To solve this problem, ACS learns action sequences [Stolzmann, 2000]. The agent executes a sequence without paying attention to the environment until it is terminated. The blind actions allow to bridge aliased situations.

Though everything presented until now was devoted to deterministic environments, [Butz et al., 2001] proposes an improvement of ACS which allows the system to deal with stochastic environments. On the contrary to deterministic environments, the actions and the perceptions are not totally reliable. When there is noise on the perceptions, some features of the perceived situations may not correspond to the actual features with some probabilities. When there is noise on the actions, an action in a particular situation may lead to different situations with some probabilities. By storing for each classifier a propability enhanced E part, ACS is able build a stochastic model of an environment. YACS only deals with deterministic environments.

So far, ACS is more mature than YACS. In the following, we compare YACS to the core of ACS, without the extensions allowing it to deal with non-Markov and stochastic environment.

5.2 Similarities between ACS and YACS

The number of classifiers produced by YACS in Maze4 with 3 irrelevant bits is different from the number of classifiers produced by ACS in the same experiment (see section 4.3). This could come from the criterion to decide whether a classifier has a correct E part in a specific case or not. Then the two systems would produce different optimal representations of the dynamics of the environment.

This criterion seems very different in ACS and YACS. In this section we will show that in fact, they are very similar.

ACS considers the current state S_t, the current action A_t and the next state S_{t+1} in the Anticipatory Learning Process (ALP). YACS uses S_{t-1}, A_{t-1} and S_t. These are only different ways of looking at the same things. Here, we use the terminology of YACS.

[Stolzmann, 1998] gives a detailed description of the comparison used in ACS. An anticipation of the next state $passthrough(S_{t-1}, E)$ is computed, then compared with S_t. In YACS, the desired effect $DE = diff(S_t, S_{t-1})$ is compared with E. This looks very different, but we will show that both kinds of comparison are very similar.

The $passthrough$ and $diff$ operators are defined for all features of the environment. Let s_{t-1} be a feature of S_{t-1}, s_t the corresponding feature for S_t, c the corresponding component of the C part and e the corresponding component of the E part.

The table 1 compares the criterion $e = diff(s_t, s_{t-1})$ used in YACS with the criterion $passthrough(s_{t-1}, e) = s_t$ used in ACS. In this table, "$\star$" means "different from #, s_t and s_{t-1}".

s_t	e	$diff(s_t, s_{t-1})$	$passthrough(s_{t-1}, e)$	YACS	ACS
$= s_{t-1}$	#	#	s_{t-1}	true	true
$\neq s_{t-1}$	#	s_t	s_{t-1}	false	false
$= s_{t-1}$	s_t	#	s_t	*false*	*true*
$\neq s_{t-1}$	s_t	s_t	s_t	true	true
$= s_{t-1}$	$\star$	#	$\star$	false	false
$\neq s_{t-1}$	$\star$	s_t	$\star$	false	false

Table 1. Truth table

The table 1 shows that the equation $e = diff(s_t, s_{t-1})$ is equivalent to the equation $passthrough(s_{t-1}, e) = s_t$ except in one case, when $s_{t-1} = s_t = e$.

In ACS a specialization of a component e in the E part always implies a specialization of the corresponding component c in the C part. If no generalization is used in ACS, c cannot become a # symbol in the future. Since every active classifier must belong to the match set, $c = s_{t-1}$. Thus c would be equal to e, but this is impossible, because only changing components are specialized in the E part. The *specialization of unchanging components* does not influence this.

Thus, if no generalization is used in ACS the comparison used in ACS is equal to the comparison used in YACS. There is only a difference if generalization is used in ACS. To summarize, we can say that the comparisons in ACS and YACS are very similar but the way specialization takes place is different. This point is discussed in the next section.

5.3 Differences between ACS and YACS

YACS is designed so as to decorrelate the acquisition of relevant C and E parts. This is an important conceptual difference between ACS and YACS. An E part contains by itself the information about the anticipated effects occurring in the environment, regardless of the condition parts. A specific value for a particular feature in the E part always indicates that a change occurs, regardless of the structure of the C part. But the classifier does not specify the initial value of the feature if it has not been specialized in the C part. YACS guarantees that the value of a particular feature of the C part of a classifier is never equal to the corresponding value in its E part. If they should be equal, the correct value in the E part is a *don't change* symbol.

This fact affects the latent learning process. The first part of this process in YACS is to set E parts to actually perceived changes in the environment. The second part of the process is in charge of discovering relevant C parts. A C part must be able to discriminate between situations so that the E part discovered by the first mechanism is always correct when the classifier can be fired. This way of creating new classifiers decorrelates the C and E parts discovery.

In ACS, the E part discovery is performed by the *specialization of changing components* process. When a classifier anticipates badly, this part of the ALP may create a new classifier by specialization of both C and E parts. Thus, the discovery of E parts in ACS relies on incremental specializations. By contrast, the E part covering process of YACS sets them to actually observed effects in one single stage, regardless of the number of *don't change* symbols. In YACS, *specialization* does not make sense for E parts since they do not contain *don't care* but *don't change* symbols.

The specialization of changing components process in ACS specializes simultaneously C and E parts in order to store the changes in the environment. It brings information about the specific values resulting from the actions (as in YACS) but also about the initial corresponding values. However, this joint specialization mechanism yields two drawbacks.

First, specializing only the changing features may forbid to identify features which are very discriminative but which are not changed by the action. In order to deal with that problem, ACS includes the *specification of unchanging components* process, which allows to discriminate between some situations when the specialization of an unchanging feature would be required.

The second problem of the specialization of changing components process is that it may introduce over-specialization by specializing irrelevant but changing features in the C part. Thus, correlating C and E parts in the ALP offers many occasions to specialize irrelevant features. YACS decorrelates C and E parts and directly identifies the features which are the most relevant to distinguish between situations so as to get able to anticipate well.

YACS takes advantage of this information to drive the *condition specialization* process. Some of the classifiers created by this process do not anticipate well (see section 2.3) and must be removed thanks to the *selection of accurate classifiers* process. In contrast, ACS always produces classifiers that anticipate at least as well as their parent. When the condition specialization process in YACS creates useless classifiers, they are removed (see section 2.4).

The *condition specialization* process is driven by experience and does not care whether the feature is changing or not. The *expected improvement by specialization* does neither identify features which are not correctly anticipated nor changing features. It focuses on the features of the C parts and does not care about particular features of the E parts. Drawing such information from experience takes time and YACS specializes less often than ACS does. It is more cautious and leads to less over-specialization problems than the specialization process of ACS.

Thus, ACS heavily relies on its generalization mechanism [Butz et al., 2000a] in order to provide a model of the environment which is as compact as possible. ACS uses a genetic algorithm in order to correct over-specialization cases. As traditional genetic algorithms, the search mechanism relies on mutation and crossover operators. It is not explicitly driven by experience. As a result, it creates classifiers which anticipate badly. Moreover, if a particular situation can be identified by specializing in several different ways, a situation may be covered by more than one single classifier. Thus, even if every classifier is at the right level of specialization, the genetic algorithm may introduce redundancies which lead to a sub-optimal number of classifiers.

However, the generalization mechanism of ACS is able to reconsider irrelevant early specializations. As YACS does not take advantage of such a mechanism, it is not able to do so. Thus, when YACS fails to choose the most relevant feature to specialize, it introduces over-specialization in the model of the environment and cannot correct the mistake. This absence of a generalization will become even more crucial if the system is interacting with a changing environment.

6 Conclusion and Future Work

The latent learning process builds a model of the dynamics of the environment even in the absence of rewards. It models how the actions modify the perceived situations. This modeling process uses information about successive perceived situations. The information used is available at each time step. So, latent learning systems make an intensive use of the perceptual feedback offered by the sensorimotor loop. Thus, they can quickly identify relevant and general classifiers.

Moreover, since the latent learning process provides information about the situations' transitions, it can speed up the policy learning process. With the latent learning and policy learning component, one can build a complete reinforcement learning system.

In this paper, we described the two components of YACS and showed experimentally how this system is able to solve maze problems. We focussed on the latent learning process and described its two main parts: the effect covering and the condition specialization process. YACS showed its ability to learn a compact model of the dynamics of the environment by taking advantage of information available at each time step. The experiments also confirmed that the cautious specialization process in YACS leads to less over-specialization than the corresponding process in ACS.

But even if the specialization process is very cautious in YACS, it does only lead to *near* optimality and YACS needs a dedicated generalization mechanism. In a short term, YACS will be enhanced with such a process, which will explicitly take advantage of experience to drive the generalization without genetic algorithms. The generalization mechanism of YACS will draw benefits of *expected improvement by generalization* estimates. These estimates should allow to decide if a classifier would anticipate correctly even if the corresponding specialized feature of the C part is general. As in the specialization mechanism, the estimates are updated according to the experience. The generalization of the C parts of accurate classifiers with the same A and E parts will be driven by the *expected improvement by generalization* estimates.

In a middle term, YACS should be enhanced to tackle non-Markov problems. Where ACS uses action sequences to bridge aliased situation, YACS will adopt a different strategy and use internal states as Lanzi [Lanzi, 1998] does. It will identify aliased situations when a classifier whose C part is already completely specialized *oscillates* (see section 2.3).

Acknowledgments The authors would like to thank Martin Butz for valuable discussions and useful comments which helped to improve YACS.

References

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

[Butz et al., 2000a] Butz, M. V., Goldberg, D. E., and Stolzmann, W. (2000a). 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)*.

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

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

[Butz and Stolzmann, 1999] Butz, M. V. and Stolzmann, W. (1999). Action-planning in Anticipatory Classifier sytems. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program*.

[Donnart and Meyer, 1996] Donnart, J. and Meyer, J. (1996). Learning reactive and planning rules in a motivationally autonomous animat. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 3(26):381–395.

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

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

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

[Holland, 1976] Holland, J. (1976). Adaptation. *Progress in theorical biology*.

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

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

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

[James, 1890] James, W. (orig. 1890). *The principles of psychology*, volume 2. Harvard University Press, Cambridge.

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

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

[Lanzi, 1999] Lanzi, P. L. (1999). An analysis of generalization in the XCS Classifier System. *Evolutionary Computation*, 2(7):125–149.

[McCallum, 1996] McCallum, R. A. (1996). Learning to use selective attention and short-term memory. In Maes, P., Mataric, M., Meyer, J.-A., Pollack, J., and Wilson, S. W., (Eds.), *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 315–324, Cambridge, MA. MIT Press.

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

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

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

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

[Stolzmann et al., 2000] Stolzmann, W., Butz, M. V., and Goldberg, D. E. (2000). First cognitive capabilities in the Anticipatory Classifier System. In Meyer, J.-A., Berthoz, A., Floreano, D., Roitblat, H., and Wilson, S., (Eds.), *From animals to animats: Proceedings of the Sixth International Conference on Simulation of Adaptative Behavior*, pages 287–296, Paris, France. MIT Press.

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

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

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

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

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

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

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

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