An Analysis of the Memory Mechanism of XCSM

Pier Luca Lanzi
Artificial Intelligence and Robotics Project
Dipartimento di Elettronica e Informazione
Politecnico di Milano
P.zza Leonardo da Vinci, 32 – I-20133 Milano Italia
lanzi@elet.polimi.it

ABSTRACT

We analyze the memory mechanism of XCSM, the extension of XCS with internal memory. Our aim is to explain some of the results reported in the literature, which show that XCSM fails to learn an optimal policy in complex partially observable environments. The analysis we present reveals that the XCSM's memory management strategy cannot guarantee the convergence to an optimal solution. We thus extend XCSM introducing a novel hierarchical exploration technique and modifying the technique used for updating internal memory. We apply the novel version of XCSM, called XCSMH, to a set of partially observable environments of different complexity. Our results show that XCSMH is able to learn an optimal policy in all the environments, outperforming XCSM in more difficult problems.

1 Introduction

The learning capabilities of adaptive agents are related to their perception of the environment. There are cases in which the agent is able to determine the state of the environment completely. The environment is thus said to be *completely observable* with respect to the agent sensors. However, in many situations the agent has only partial information about the state of the environment: agent sensors are not adequate to determine the state of the environment completely. The agent is then said to suffer from the *perceptual aliasing problem*, while the environment is *partially observable* with respect to the agent sensors.

When facing an environment that is partially observable a memory mechanism is usually introduced in order to cope with the lack of information deriving from the sensors (Kaelbling *et al.* 1996, McCallum 1996). Since the agent is unable to determine the best action solely looking at current inputs, it employs the internal memory to keep track of the past states so that it can develop an optimal strategy. This approach is also followed in Wilson's XCS classifier systems Wilson (1995).

According to its original definition XCS has no memory mechanism; therefore, it is able to learn optimal strategies only in environments that are completely observable. When applied to partially observable environments, XCS converges to a solution that cannot be improved under the assumption that the agent is memoryless (Lanzi 1998). XCS was extended with the memory mechanism proposed by Wilson (1995) in (Lanzi 1998). XCS with internal memory, named XCSM, was applied to three partially observable environments: `Woods101`, `Woods102`, and `Maze7`. Experimental results showed that XCSM successfully learns an optimal policy in simple environments, like `Woods101` and `Woods102`, while it fails to evolve an optimal solution in more complex environments, such as `Maze7`.

In this paper we study the memory mechanism of XCSM as implemented by Lanzi (1998). Our aim is to explain the results previously reported, in order to devise a new version of XCSM capable of learning in complex partially observable environments.

Initially, we study XCSM's behavior in `Maze7`, extending the analysis we previously presented in (Lanzi 1998). Then, we use a simple example in order to explain the phenomenon previously investigated. Our analysis suggests that XCSM cannot solve general partially observable problems, because the memory management strategy of XCSM cannot guarantee the convergence to an optimal strategy for disambiguating aliasing positions. Our results are very general and evidence a phenomenon, never noticed before, which concerns the evolutionary approach that XCSM implements. We validate our analysis showing the conditions which underlie the good performance of XCSM in simple environments, such as `Woods101` and `Woods102`.

We then introduce an extension of XCSM, which consists of (i) a different policy for updating the contents of the internal memory and (ii) a *hierarchical* exploration strategy. We apply the extended version of XCSM, called XCSMH, to the environments previously considered. We report experimental results which show that XCSMH is able to evolve an op-

timal policy for all the previous environments. Finally, we extend these results, comparing XCSM and the new XCSMH in `Maze10`, a much more difficult environment. The results we present show that XCSMH still converges to an optimal policy, outperforming XCSM.

The paper is organized as follows. Section 2 briefly overviews XCS as originally proposed by Wilson (1995), and XCSM as implemented in (Lanzi 1998). The experimental settings we use are described in Section 3. In Section 4 we summarize the results presented in (Lanzi 1998), extending the analysis of XCSM's performance in `Maze7`. We employ a simple example to explain previous results in Section 5; while in Section 6 we validate our analysis. Section 7 introduces the proposed extension of XCSM, called XCSMH; the new system is applied to the three environments previously presented in the literature, and to the new `Maze10` environment in Section 8. Finally, Section 9 ends the paper drawing some conclusions.

2 Description of XCS and XCSM

We shortly describe XCS as originally introduced by Wilson (1995), and the implementation of XCSM we presented in (Lanzi 1998).

2.1 Wilson's XCS

The XCS classifier system differs from Holland's framework (Holland 1986) by three major factors. First, in XCS classifier fitness is based on the accuracy of the prediction instead of the prediction itself. Accordingly, the traditional *strength* parameter is replaced by three parameters: (i) the prediction p, which estimates the payoff that the animat is expected to gain; (ii) the prediction error ε, that evaluates how much precise the prediction p is; finally, (iii) the fitness F, which evaluates the accuracy of the prediction p, and thus is a function of the prediction error ε. Moreover, XCS has a very basic architecture with respect to the original framework seeing that XCS has no internal message list, and no other memory mechanisms. Finally, in XCS the genetic algorithm is applied to environmental niches, as opposed to the panmictical GA. XCS works as follows.

At each time step the system input is used to build a match set [M] containing the classifiers in the population whose condition matches the detectors. If the match set is empty a new classifier that matches the input sensors is created through *covering*. For each possible action a_i the *system prediction* $P(a_i)$ is computed. $P(a_i)$ gives an evaluation of the expected payoff if action if action a_i is performed. Action selection can be *deterministic* (the action with the highest system prediction is chosen), or *probabilistic* (the action is chosen with a certain probability among the actions with a not null prediction). The classifiers in [M] that propose the selected action are put in the *action set* [A]. The selected action is performed and an immediate reward is returned to the system together with a new input configuration. The reward received from the environment is used to update the parameters of the classifiers in the action set corresponding to the previous time step $[A]_{-1}$. Classifier parameters are updated by the Widrow-Hoff delta rule (Widrow and Hoff 1960) using a Q-learning-like technique (Watkins 1989, Wilson 1995).

The genetic algorithm in XCS is applied to the action set. It selects two classifiers with probability proportional to their fitnesses, copies them, and with probability χ performs crossover on the copies while with probability μ mutates each allele.

An important innovation, introduced with XCS is the definition of *macroclassifiers*. A macroclassifier represents a set of classifiers which have the same condition and the same action using a new parameter called *numerosity*. Macroclassifiers are essentially a programming technique that speeds up the learning process reducing the number of *real*, macro, classifiers XCS has to deal with.

Since XCS was presented, two genetic operators have been proposed as extensions to the original system: Subsumption deletion (Wilson 1998) and Specify (Lanzi 1997*b*). Subsumption deletion has been introduced to improve generalization capabilities of XCS. Specify has been proposed to counterbalance the pressure toward generalization, in situations where a strong genetic pressure may prevent XCS from converging to an optimal solution.

2.2 Adding Internal Memory to XCS

As introduced in the previous section, XCS has no memory mechanism. Therefore, it can learn optimal policies for environments that are completely observable. When facing partial observable environments, XCS will typically evolve a solution which cannot be improved under the assumption that the agent is memoryless, as we have shown in (Lanzi 1998).

Wilson (1995) proposed an extension to XCS by which an internal memory register could be added to the system. Wilson's proposal, that we implemented in (Lanzi 1998), consists of (i) adding to XCS an internal register, and (ii) extending classifiers with an internal condition and an internal action, which are employed to *sense* and modify the contents of the internal register.

The internal register consists in a string of b bits; the the classifier's internal condition/action are strings of b symbols in the alphabet $\{0,1,\#\}$. For internal conditions, the symbols retain the same meaning they have for external condition, but they are matched against the corresponding bits of the internal register. For internal actions, 0 and 1 set the corresponding bit of the register to 0 and 1 respectively, while # leaves the bit unmodified. There are nine possible external actions, eight moves and one *null* action, encoded using two symbols in the alphabet $\{0, 1, \#\}$. Internal conditions/actions are initialized at random as usual.

In the rest of the paper, we refer to XCS with b bits of internal memory as XCSMb, simply to XCSM when the discussion is independent of the value b. Finally, we use XCS(M) for referring both to XCS and XCSM.

XCSM works basically as XCS. At the start of each trial, the internal register is initialized setting all bits to zero. At

each time step, the match set [M], the prediction array, and the action set [A] are built as in XCS. The only difference is that in XCSM the internal condition is considered when building [M], and the internal action is taken into account when building the prediction array. The action set [A] is computed as in XCS, while the external action and the internal action are performed in parallel. The credit assignment procedure is the same as for XCS.

3 Experimental Settings

The experiments we present in this paper are carried out in the well-known *woods* environments. These are grid worlds in which each cell can be empty, can contain a tree (a `T` symbol), or otherwise food (an `F` symbol). An animat, placed in the environment, must learn to reach food. The animat senses the environment by eight sensors, one for each adjacent cell, and it can move in any of the adjacent cells. If the destination cell is blank then the move takes place; if the cell contains food the animat moves, eats the food, and receives a constant reward; if the destination cell contains a tree the move does not take place. An experiment consists of 15000 problems that the animat must solve. For each problem the animat is randomly placed in a blank cell of the environment and then it moves under the control of the system until it eats a piece of food, receiving a constant reward. The food immediately re-grows and a new problem begins.

We employ the usual exploration/exploitation strategy used with XCS by Wilson (1995, 1998). Before a new problem begins the animat decides, with probability 0.5, whether it will solve the problem in exploration or in exploitation. When in exploration, the system decides, with a probability P_s (a typical value is 0.5), whether to select actions randomly or to choose the action that predicts the highest payoff. When in exploitation, the genetic algorithm is not active and the action which predicts the highest payoff is always selected.

In order to evaluate the final policy evolved, in each experiment exploration is turned off during the last 2500 problems and the system works in exploitation only. System performance is computed as the average number of steps to food in the last 50 exploitation problems. Every statistic presented in this paper is averaged on ten experiments.

4 XCSM in Partially Observable Environments

We presented early results for XCSM in (Lanzi 1998) where the system was applied to three partially observable environments: `Woods101`, `Woods102` and `Maze7`. Results we reported show that XCSM is able to learn to reach food in `Woods101` and `Woods102`, while it cannot converge to optimal performance in `Maze7`. In this section, we briefly summarize the results presented in the original paper for the first two environments. Then, we discuss XCSM's behavior in `Maze7` extending the analysis we did in (Lanzi 1998).

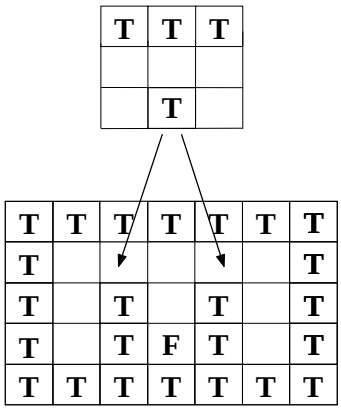

Figure 1 The Woods101 environment. Aliasing positions are indicated by the arrows.

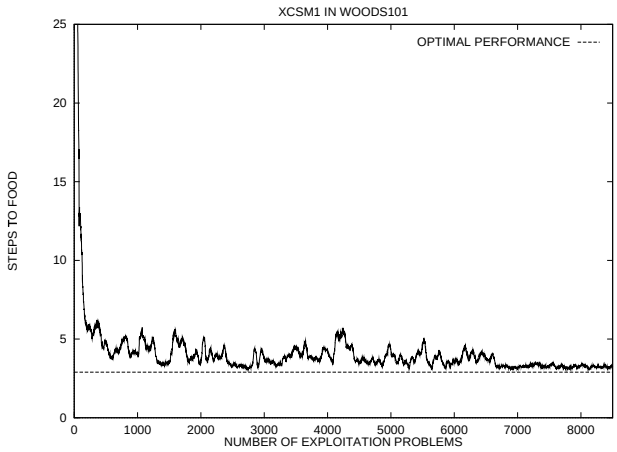

Figure 2 XCSM1's performance in Woods101.

4.1 XCSM in Woods101 and Woods102

The `Woods101` environment (Cliff and Ross 1994), shown in Figure 1, has two states which return the same sensory configuration, but require different optimal actions. Therefore, `Woods101` can be solved by an agent with one bit of internal memory. We apply XCSM1 with a population of 1600 classifiers to `Woods101`; the experimental results reported in Figure 2, show that XCSM1 is able to exploit the internal memory bit in order to evolve an optimal policy for `Woods101`.

Similar results are obtained if XCSM with two bits of internal memory is applied to `Woods102` (see Figure 3.a), a more difficult environment that has two types of aliasing states. The former, see 3.b, is encountered in four positions in the environment; the latter, shown in Figure 3.c, appears in two positions of the environment. An agent with two bits of internal memory, able to represent four distinct internal states, can thus disambiguate the aliasing positions in `Woods102`. We apply XCSM2 with a population of 2000 classifiers to `Woods102`. [1] Results reported in Figure 4 show that XCSM2 successfully evolves an optimal solution for `Woods102`.

[1] We refer the interested reader to the original paper (Lanzi 1997*a*) for an accurate discussion concerning XCSM's parameter settings and the generalization issue in `Woods102`.

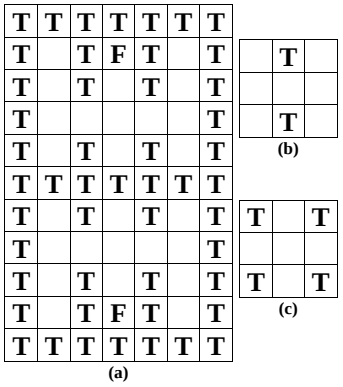

Figure 3 The Woods102 environment (a) with the corresponding aliasing states (b) and (c)

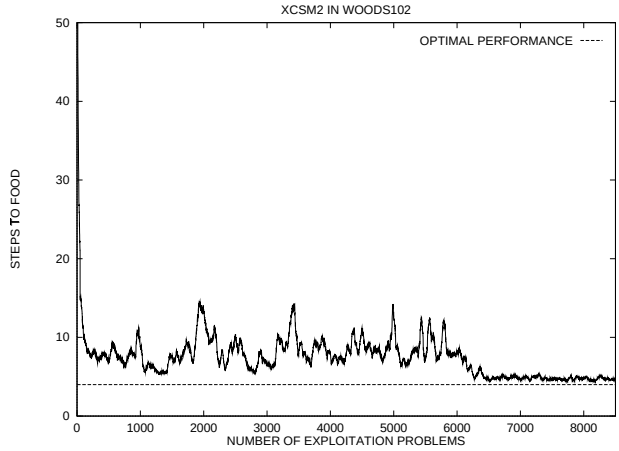

Figure 4 XCSM2 in Woods102.

4.2 XCSM in Maze7

`Woods101` and `Woods102` are simple problems, seeing that in both environments: the optimal solution requires the agent to visit at most one aliasing state before it reaches the food, and the goal state is also very near to the aliasing cells (Lanzi 1998).

We thus introduced `Maze7` in order to test XCSM in an environment where the animat has to evolve an optimal strategy to visit more aliasing positions before it can eat, and must perform longer sequences of actions to reach the goal state.

`Maze7`, see Figure 5, is a simple environment consisting of a linear path of nine cells to food; it has two aliasing positions, indicated by the dashed circles. As the experimental results confirm, `Maze7` is much more difficult to solve than `Woods101` and `Woods102`.

We apply XCSM1 to `Maze7` with a population of 1600 classifiers. Results are reported in Figure 6; as usual, during the last 2500 problems exploration is turned off to evaluate the final policy evolved. Figure 6 shows that while exploration acts the system does not converge to an optimal solution, but when the final population is evaluated turning off exploration, see the arrow at beginning of the peak, XCSM1 evolves an optimal solution to the problem.

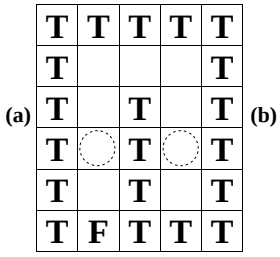

Figure 5 The Maze7 Environment. The two aliasing position, (a) on the left and (b) on the right, are evidenced by dashed circles.

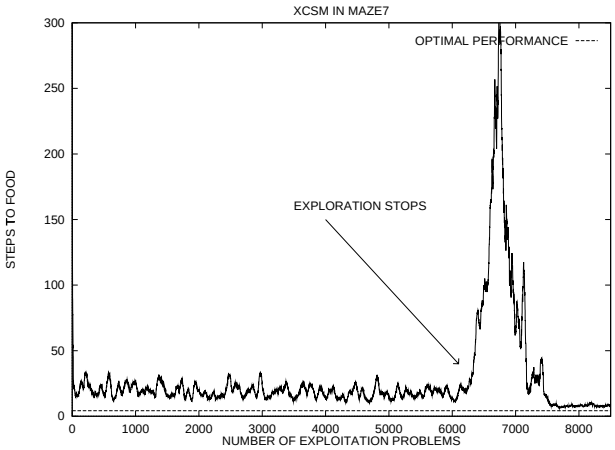

Figure 6 XCSM1's performance in Maze7.

The analysis of the population dynamic evidences that, when exploration acts, the system is not able to learn an optimal policy to reach the positions at the end of the corridor. Therefore, XCSM's performance drops when an experiment starts in one of the positions for which the optimal policy has not evolved, so that the overall performance oscillates. Most important, when the exploration stops, at the beginning of the peak in Figure 6, the performance drops indicating that the final policy causes the animat to loop in some positions of the environment. XCSM detects this situation because the prediction of the classifiers involved dramatically decreases (Wilson 1995). Therefore, XCSM starts replacing such low predictive classifiers through covering. The final policy, at the end of the peak, is thus built from classifier created by the covering operator. This is also confirmed by the results we obtain applying XCSM1 to `Maze7` only in exploitation, i.e. the genetic operators do not act and always the best action is selected. Results reported in Figure 7 show in fact that the system, on the average, converges to a nearly optimal solution, suggesting that `Maze7` is a simple problem for XCSM. Moreover, the analysis of single runs (Lanzi 1998) shows that in many cases this basic version of XCSM converges to the optimum, while seldom it evolves nearly optimal solutions.

At this point, we may conclude that a basic version of XCSM, which only relies on covering and exploitation, could be an adequate solution to the general problem of learning to

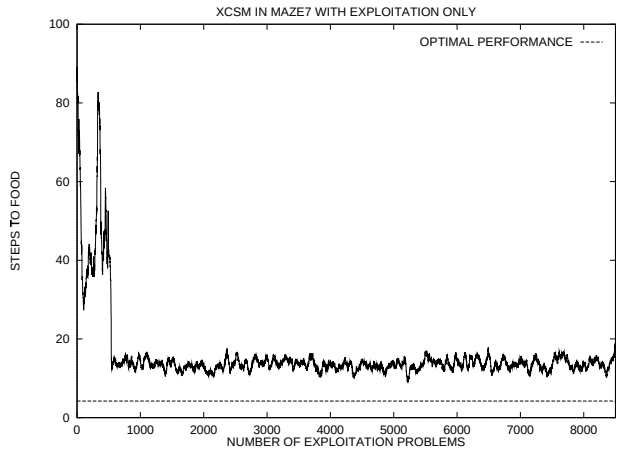

Figure 7 XCSM1 in Maze7 using exploitation only: the GA is off and always the best action is selected.

reach food in partially observable environments. However, a system only based on covering and exploitation is not acceptable; in fact, it is worth noticing that:

- A system that does not employ any exploration cannot be guaranteed to converge to an optimal policy; moreover, as the complexity of the environment increases, XCSM is more likely to get stucked in local optima.
- Since generalization in XCS is achieved through evolution, a system only based on covering would throw away all the generalization power of XCS(M).

In our opinion, the behavior we observed points out some important aspects of the evolutionary approach for learning in partially observable domains, that XCSM implements. Accordingly, we think it is essential to study the real causes of XCSM's performance in order to improve our knowledge of learning classifier systems in general, and specifically of XCS(M).

5 Analysis of XCSM's Behavior

We may formulate two hypotheses in order to explain XCSM's behavior in `Maze7`. First, we may assume that the genetic component somehow prevents the system from converging to an optimal policy. On the other hand, we may also hypothesize that the random exploration strategy, employed in XCS(M), is not adequate for evolving optimal solutions in general partially observable environments. As we observed in (Lanzi 1998), in fact, exploration in XCS is done solely "*in the environment*" and therefore relies both on the structure of the environment and on the exploration strategy used, that in our case is random. On the contrary, XCSM has also to perform exploration "*in the internal memory*" in order to develop the best policy to disambiguate aliasing situations. Such an exploration is uniquely based on the agent strategy, that in XCSM is still random.

We now analyze how random exploration influences XCSM's performance when the system is applied to `Maze7`, by studying a simple example.

Consider the two aliasing positions in `Maze7`, evidenced in Figure 5 by the dashed circles. A possible optimal policy, which disambiguates the two aliasing positions (a) and (b), consists in two classifiers which advocate the actions: *go-south* when the internal register is set to 1 in position (a), and *go-north* when the internal register contains 0 in position (b). [2] Assume that XCSM has successfully evolved this policy and that, at this point, the system starts exploration.

Since exploration in XCSM is random, it may happen that the agent enters position (b) with the internal register set to one. The agent can now select any of the actions which appear in the match set and, for example, it may experiment the action *go-south*.

Observe that, if such a situation occurs, the classifiers activated in position (b) are the same which predict the optimal action in position (a). However, in the two positions, the action *go-south* has different payoffs. Therefore, the classifiers that are optimal in (a), become inaccurate when experimented in (b), because they are applied in *the same situation* with different payoff levels. Accordingly, since in XCSM (like in XCS) fitness is based on accuracy, when classifiers that are optimal in (a) are experimented in (b), their fitness decreases. Consequently, they reproduce less and may be selected for deletion through the evolution process.

As it can be noticed, our argumentation does not concerns overgeneralization (Lanzi 1997*b*) nor XCS(M) definition. On the contrary, the problem we evidence uniquely depends on the strategy that the system employs to explore the environment. However, as we show next, the phenomenon we observed has more general implications that cannot be solved simply introducing a new exploration technique.

6 Hypothesis Verification

Before we proceed any further, it is essential to discuss some issues that the analysis we presented arises.

First, we shall verify whether our explanation of XCSM's behavior in `Maze7` is correct. We thus apply the system to `Maze7` when the best action is selected both in exploration and in exploitation. Notice that in the previous experiment (whose results are reported in Figure 7) the GA was turned off, while now the GA is on during exploration. Our aim is to verify that the phenomenon we observed is indeed a consequence of the exploration strategy, and does not depend on the genetic component.

The results reported in Figure 8 show that XCSM1 converges to an optimal solution for `Maze7` when the action selection is always deterministic and the GA is on. These results therefore support our explanation of XCSM's behavior in `Maze7`.

Moreover, comparing these results with the ones for the version of XCSM that works in exploitation only (see Fig-

[2] For simplicity's sake, we do not consider the whole optimal policy which would require that the agent enters position (a) and (b) with the correct internal memory configuration.

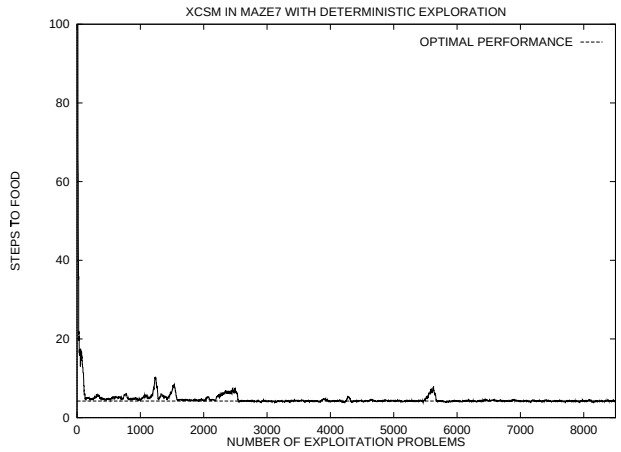

Figure 8 XCSM1's performance in Maze7 when the GA is on during exploration and always the best action is selected both in exploration and in exploitation.

ure 7), it is worth noticing that XCSM1 is able to converge to an optimal solution even if deterministic action selection is employed both in exploration and in exploitation. This suggests that, since the problem is quite simple, the exploration performed by the genetic operators is sufficient to guarantee the convergence to an optimal solution in `Maze7`.

Our analysis appears to be very general, and thus should apply to all the environments. Consequently, it may seem that our explanation is contradicted by the positive results we reported in Section 4.1 and in (Lanzi 1998) for `Woods101` and `Woods102`. Therefore it is important to analyze XCSM's behavior in `Woods101` so to explain the different results that the system produces in `Woods101` and `Maze7`.

The analysis of the single runs of XCSM in `Woods101` shows that also in `Woods101`, classifiers, that are optimal in one of the two aliasing positions, may become inaccurate because they are experimented in both the aliasing positions. However, since in `Woods101` the aliasing cells are near to the goal state, the agent visits these positions frequently. Moreover, since the two positions are symmetric with respect to the food cell, the payoff levels for the same action in both positions are similar and, most important, the two aliasing states are visited almost with the same frequence. Consequently, classifiers which advocate the optimal actions in the two aliasing positions are able to survive, even if they may be applied to situations that have different payoff levels. In XCS(M), in fact, inaccurate classifiers are deleted through evolution, that is they tend to reproduce less and consequently to be deleted. Nevertheless, this mechanism is slow (Lanzi 1997*b*); accordingly, in `Woods101` its effects are counterbalanced by the frequent exploitation of classifiers which form the optimal policy. On the contrary, we observe that in `Maze7`:

(i) The two aliasing positions are not symmetric with respect to the goal state; thus, the payoff levels of the same action are significantly different in the two positions.

(ii) Aliasing position (a) is near the goal state, therefore, it is visited more frequently than position (b) which is at the end of the corridor.

The motivations which underlie the difference in XCSM's performance between `Maze7` and `Woods101` are now clear. Because of (i), in `Maze7` the classifiers, that are optimal in a certain aliasing position, are more likely to become inaccurate when they are experimented in different aliasing states because the payoff levels are very different. Moreover, since position (b) is far from the food cell, it is visited less often than position (a). Accordingly, classifiers that are optimal with respect to position (a) tend to survive, because they are experimented frequently in (a) and seldom in (b). On the contrary, classifiers that are optimal in (b) tend to be selected for deletion because they are often experimented in (a) and seldom in (b).

7 An Extension to XCSM

According to its definition, XCSM should learn an optimal policy in partially observable environments, by evolving a strategy to update the contents of the internal memory. The goal is to associate the state of the internal memory to the actual agent position in the environment, so to disambiguate aliasing situations. However in XCSM, since exploration is random, the system may enter the same position with different memory settings. In such a case, the memory mechanism becomes useless with respect to the aliasing problem, because the contents of internal memory register are no longer related with the absolute agent position.

7.1 Possible Solutions

The analysis we presented so far reveals that the XCSM's memory management strategy may not guarantee the convergence to optimum in general partially observable environments. Therefore, we need to devise another strategy for exploiting internal memory in order to solve general problems.

As a first solution, we may decide to resign the evolutionary approach completely. Accordingly, we could extend XCS by adapting one of the methods presented in the reinforcement learning literature, like (McCallum 1996). Although this kind of approach may lead to good results, it would be an ad-hoc solution, not well integrated with the philosophy underlying Wilson's classifier system.

Another possible solution would consists in extending XCS with an internal message list as in the original Holland's framework. However, as already discussed in (Wilson 1994), internal messages tend to be long and the evolution of appropriately coupled classifiers becomes increasingly improbable as the message length increases.

7.2 Description of XCSMH

We now introduce an extension to XCSM, we call it XCSMH, which is capable of learning an optimal strategy in general partially observable environments. XCSMH retains the same structure of XCSM, while it differs from the previous system by two major factors:

- In XCSMH internal actions are performed only if the corresponding external action causes a change in the sensory inputs; that is, if the agent has changed position in the environment.
- XCSMH employs a *hierarchical exploration* strategy: the system first selects the internal action deterministically; then, it selects the external action with the usual random exploration.

These changes that we introduce may appear to be supported by obscure motivations. However, as we show next, they are strictly related with the analysis of XCSM's behavior we presented in the previous sections. In the following, we discuss each change in detail.

7.2.1 Internal Actions

As we noticed in the previous section, the main problem of XCSM is that internal actions act independently with respect to the agent position in the environment. The internal state can thus be modified even if the animat does not change its position in the environment. Accordingly, classifiers that are accurate in one specific position may become inaccurate because they are experimented in other positions of the environment corresponding to different payoff levels.

XCSMH associates the execution of the internal action with a transition in the environment: An internal action is in fact performed only if the corresponding external action has caused the agent to move in the environment. The contents of the internal memory in XCSMH are thus associated with the sequence of environmental states the agent visited. The internal memory in fact does not represent the result on an execution of an *internal program* anymore (Cliff and Ross 1994). The register, in fact, cannot be updated independently from what the agent experiments in the environment. The content of the internal memory in XCSMH rather gives a *compact* representation of the agent past experiences. As an immediate consequence of this memory update policy, XCSMH no longer employs *null* external actions.

7.2.2 Hierarchical Exploration.

The second change we introduce in XCSMH concerns the exploration strategy. As we discussed previously, random exploration may cause the agent to enter different aliasing states with the same internal memory configuration. Consequently, the system fails to associate the state of the internal memory to the actual agent in the environment.

In XCSMH, we replace the usual random exploration strategy by a *hierarchical exploration* in order to limit the set of internal actions that the agent can perform in each position. As a consequence, the system will tend to associate a specific internal state with each position in the environment. The strategy we propose works as follows:

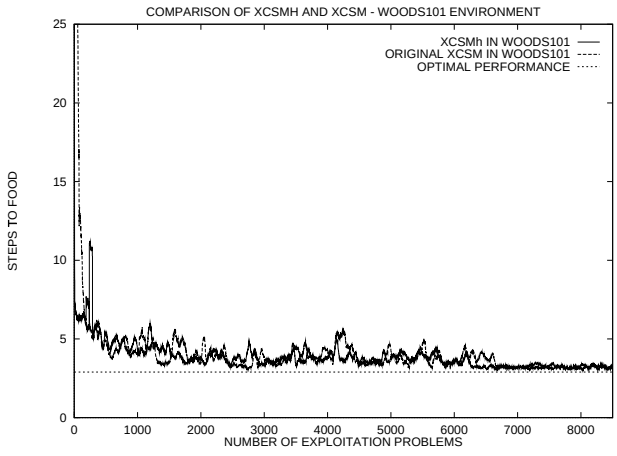

Figure 9 Comparison of the new XCSMH1 (solid line) and XCSM1 (dashed line) in Woods101.

- First, the internal action is selected as the one corresponding to the internal/external action pair which predict the highest payoff.
- Then, the external action is chosen randomly among the action pairs which have the same internal action which has been selected in the previous step.

The exploration strategy, we propose, selects the internal action deterministically so that, in each environmental niche, the classifiers which advocate the best internal action will tend to survive. On the other hand, the external action is selected randomly, as usual, to guarantee the exploration of the environment.

Hierarchical exploration may appear similar to the deterministic strategy, because the internal action is always selected as done in exploitation. However, the analysis of the final populations evidences that the system tends to build a complete mapping of the payoffs for external actions, while it tends to associate an unique internal action for each position of the environment.

8 Experimental Results

We now apply XCSMH to two of environments discussed in the previous sections in order to compare XCSM and the novel XCSMH.

First we apply XCSMH1 to `Woods101` with the same parameter settings employed for XCSM1. Results reported in Figure 9 show that in `Woods101` the performances of the two systems are almost identical. Experimental results, not reported here, show that similar results are also obtained in `Woods102`.

The performance of the two systems becomes significantly different when they are applied to `Maze7`. As the results in Figure 10 show, XCSMH1 (solid line) performs better that

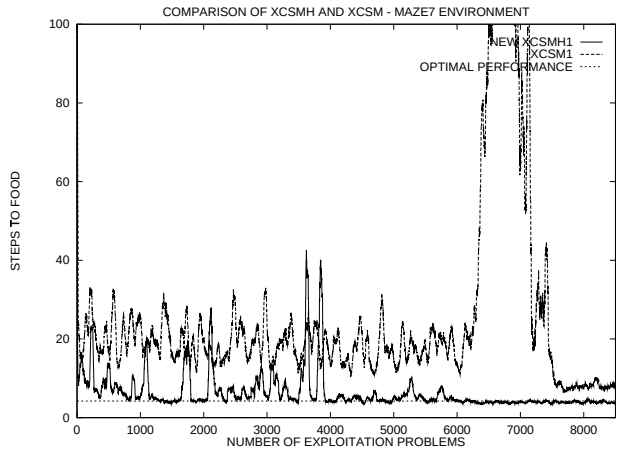

Figure 10 Comparison of the new XCSMH1 (solid line) and XCSM1 (dashed line) in Maze7.

T	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
T								**T**
T		**T**		**T**		**T**		**T**
T		**T**		**T**		**T**		**T**
T		**T**	**F**	**T**		**T**		**T**

Figure 11 The Maze10 Environment.

XCSM1 (dashed line): the system we propose easily converges to an optimal policy for `Maze7`.[3]

Finally, we compare XCSMH1 and XCSM in the new `Maze10` environment, see Figure 11. `Maze10` is more difficult than `Maze7` since it has three aliasing positions, similar to the ones in `Woods101`, and necessitates longer sequences of actions to reach the goal state. As it can be noticed, two of the three aliasing states of `Maze10` require the same optimal action; accordingly, one bit of internal memory is sufficient to disambiguate the aliasing positions. We apply XCSMH1 and XCSM1 to `Maze10` with a population of 2000 classifiers. Results in Figure 12 show that XCSM1's behavior in `Maze10` is similar to the one we observed in `Maze7`. As in `Maze7` in fact, XCSM1 is not able to evolve an optimal policy when in exploration, while the system converges to a nearly optimal solution when the system acts only in exploitation.

XCSMH1 immediately reaches an almost optimal solution. As it can be noticed, when exploration acts XCSMH's performance is quite unstable. `Maze10` is in fact a quite difficult environment, therefore the system is extremely sensible to unlucky exploration. However, when the final population is evaluated XCSMH's performance results to be optimal.

[3] Observe that the vertical scale of Figure 10 has been reduced in order to better compare the performances of the two systems with respect to the optimal performance. Unfortunately, part of the peak corresponding to XCSM1's performance goes off-scale.

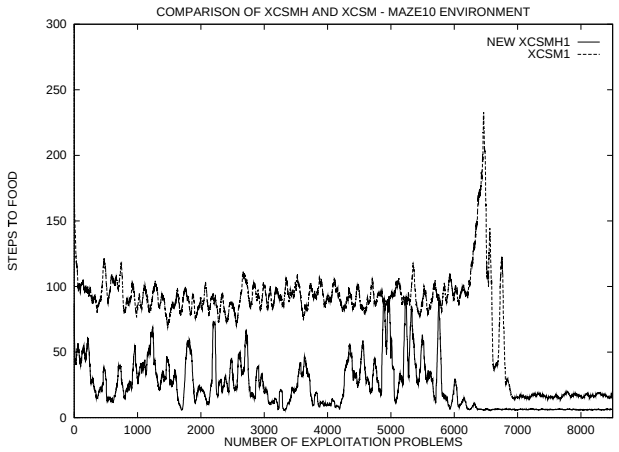

Figure 12 Comparison of the new XCSMH1 (solid line) and XCSM1 (dashed line) in Maze10.

9 Conclusions

We presented an analysis of the memory mechanism of XCSM in order to explain the results reported in the literature.

We started from the early results presented in (Lanzi 1998), where we showed that XCSM easily converges to an optimal policy in simple environments, but it is not able to solve more difficult problems, such as `Maze7`. Our analysis evidenced that the memory mechanism, as proposed by Wilson (1995) and implemented in (Lanzi 1998), is not able to evolve an optimal solution for general partially observable problems. As a simple example showed, XCSM can in fact fail to associate the state of the internal memory to the actual agent position in the environment.

We introduced an extension to XCSM, we call it XCSMH, capable to solve the problems evidenced by our analysis. The new system basically retains the same structure of XCSM, while it differs from XCSM because: (i) XCSMH employs a different policy for updating the internal memory; (ii) XCSMH uses a hierarchical exploration strategy to select actions. Our results show that XCSMH is able to evolve an optimal policy for all the environments previously presented. Most important, when applied to a more difficult environment, such as `Maze10`, XCSMH still converges to an optimal solution, outperforming XCSM.

Acknowledgments

I wish to thank Marco Colombetti for the support and for the many interesting discussions. Stewart Wilson for reviewing early versions of this papers and for the many discussions and suggestions. Finally, my thanks also go to an anonymous reviewer.

References

Cliff, Dave and Susi Ross (1994). Adding memory to ZCS. *Adaptive Behaviour* **3**(2), 101–150.

Holland, John H. (1986). *Machine learning, an artificial intelligence approach. Volume II*. Chap. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems, pp. 593–623. Morgan Kaufmann.

Kaelbling, Leslie Pack, Michael L. Littman and Andew W. Moore (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*.

Lanzi, Pier Luca (1997*a*). Solving Problems in Partially Observable Environments with Classifier Systems (Experiments on Adding Memory to XCS). Technical Report 97.45. Dipartimento di Elettronica e Informazione - Politecnico di Milano. Available at http://www.elet.polimi.it/people/lanzi/listpub.html.

Lanzi, Pier Luca (1997*b*). A Study on the Generalization Capabilities of XCS. In: *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann.

Lanzi, Pier Luca (1998). Adding Memory to XCS. In: *To appear in the Proceedings of the IEEE Conference on Evolutionary Computation*. IEEE Press.

McCallum, R. Andrew (1996). Hidden state and reinforcement learning with instance-based state identification. *IEEE Transations on Systems, Man and Cybernetics - Part B (Special issue on Learning Autonomous Robots)*.

Watkins, C.J.C.H. (1989). Learning from delayed reward. PhD Thesis, Cambridge University, Cambridge, England.

Widrow, B. and M. Hoff (1960). Adaptive switching circuits. In: *Western Electronic Show and Convention*. Vol. 4. Institute of Radio Engineers (now IEEE). pp. 96–104.

Wilson, S. W. (1994). ZCS: a zeroth level classifier system. *Evolutionary Computation* **1**(2), 1–18.

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

Wilson, Stewart W. (1998). Generalisation in the XCS classifier system. In: *To appear in the Proceedings of the Third Annual Genetic Programming Conference (GP-98)*. (MIT Press, Ed.).