Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci 32
I-20133 Milano – Italia

Solving Problems in Partially Observable Environments with Classifier Systems
(Experiments on Adding Memory to XCS)

Pier Luca Lanzi
Technical Report N. 97.45
October 17^{th}, 1997

Solving Problems in Partially Observable Environments with Classifier Systems

(Experiments on Adding Memory to XCS)

Pier Luca Lanzi

Artificial Intelligence and Robotics Project
Diparitimento di Elettronica e Informazione
Politecnico di Milano
P.zza Leonardo da Vinci 32
I-20133 Milano Italia
Voice: +39-2-23993622
Fax: +39-2-23993411
E-mail: lanzi@elet.polimi.it

Technical Report N. 97.45

October 17^{th}, 1997

Abstract

XCS is a classifier system recently introduced by Wilson that differs from Holland's framework in that classifier fitness is based on the accuracy of the prediction instead of the prediction itself. According to the original proposal, XCS has no internal message list as traditional classifier systems does; hence XCS learns only reactive input/output mappings that are optimal in Markovian environments. When the environment is partially observable, i.e. non-Markovian, XCS evolves suboptimal solutions; in order to evolve an optimal policy in such environments the system needs some sort of internal memory mechanism.

In this paper, we add internal memory mechanism to the XCS classifier system. We then test XCS with internal memory, named XCSM, in non-Markovian environments of increasing difficulty. Experimental results, we present, show that XCSM is able to evolve optimal solutions in simple environments, while in more complex problems the system needs special operators or special exploration strategies. We show also that the performance of XCSM is very stable with respect to the size of the internal memory involved in learning. Accordingly, when complex non-Markovian environments are faced XCSM performance results to be more stable when more bits than necessary are employed. Finally, we extend some of the results presented in the literature for classifier system in non-Markovian problems, applying XCSM to environments which require the agent to perform sequences of actions in the internal memory. The results presented suggest that the exploration strategies currently employed in the study of XCS are too simple to be employed with XCSM; accordingly, other exploration strategies should be investigated in order to develop better classifier systems

1 Introduction

XCS is a classifier system proposed by Wilson [14] that differs from Holland's framework [2] in that (i) classifier fitness is based on the accuracy of the prediction instead of the prediction itself and (ii) XCS has a very basic architecture with respect to the traditional framework. Introducing (i) XCS develops a strong tendency to evolve near-minimal populations of accurate and maximally general classifiers; while (ii) permits a better insight of the learning mechanism which underlies the classifier system. Several common features of Holland's classifiers have in fact been removed in XCS, in order to simplify the study of the mechanism of learning. This has led to some interesting results: [14], see also [13], presents an analysis of the similarities between XCS and the Q-learning technique [10], while in [15] experimental results are presented showing that XCS can learn a more compact representation than that learned by tabular Q-learning.

According to the original proposal, XCS does not include an internal message list, as Holland's classifier system does, and no other memory mechanism either. XCS can thus learn optimal policy in Markovian environments where, in every situation, the optimal action is always determined solely by the state of current sensory inputs. But in many applications, the agent has only partial information about the current state of the environment, so that it does not know the state of the whole world from the state of the sensory input alone. The agent is then said to suffer from the *hidden state problem* or from the *perceptual aliasing problem*, while the environment is said to be *partially observable* with respect to the agent [3]. Since optimal actions cannot be determined only looking at the current inputs, the agent needs some sort of memory of past states in order to develop an optimal policy. Such environments are non-Markovian, also *Class 2* environments according to [12], and form the most general class of environments. When in *Class 2* environments XCS can only develop a suboptimal policy, in order to learn an optimal policy in such domains, XCS would require a sort of memory mechanism or local storage.

An extension to XCS was proposed [14] by which an internal state could be added to XCS like a sort of *"system's internal memory"*. The proposal consists of (i) adding to XCS an internal memory register, and (ii) extending classifiers with an internal condition and an internal action, employed to *sense* and *act* on the internal register. The same extension was proposed [13] for ZCS the *"zeroth level"* classifier system from which XCS was derived; the proposal was validated for ZCS in [1, 9] where experimental results were presented which showed that (i) ZCS with internal memory can solve problems in non-Markovian environments when the size of internal state is

limited [1]; while (ii) when size internal memory grows the learning become unstable [9].

Wilson's proposal has never been implemented for XCS and in the literature no results have been presented for extending XCS with other memory mechanisms. In this paper we validate Wilson's proposal for adding internal state to XCS. Experimental results we report, show that XCS with internal state, we call it XCSM, evolves optimal solutions in non-Markovian environments when a sufficient number of bits of internal memory is employed; while the system still converges to an optimal policy in a stable way when a larger internal memory is employed. We then extend the study of XCSM applying the system to `Maze7`, an environment which requires sequential use of the internal memory, in order to evolve an optimal solution. The results we present show that XCSM cannot solve `Maze7` when it works using exploration, while a suboptimal solution is easily evolved when XCSM works using only exploitation. Unfortunately, a classifier system which only works in exploitation is unacceptable since it eliminates all the search and generalization capabilities of XCS. Hence, we analyze the experiments in `Maze7` in order to understand why XCSM is not able to evolve an optimal solution when the system works in exploration. Our analysis suggests that the exploration strategies employed with XCS are too simple in order to guarantee the convergence to an optimal solution when internal memory is employed. Accordingly, we believe that other exploration strategies should be investigated in order to obtain better convergence in problems that involve environments that are partially observable. The paper is organized as follows. Section 2 discuss the hidden state problem and introduce the two principal classification for environments presented in the literature. Section 3 presents an overview of XCS, while Section 4 introduces the "woods" environments and the design of experiments. Section 5 discusses the performance of XCS in non-Markovian environments. Wilson's proposal and our implementation of XCS with internal state, we call it XCSM, is presented in Section 6. In Section 7, XCSM is applied to two non-Markovian environments, `Woods101` and `Woods102`. The stability of learning of XCSM is then discussed in Section 8, while in Section 9 the previous results are extended applying XCSM to `Maze7` a more difficult environment in which the optimal policy requires the agent to perform a sequence of actions on the internal memory. Conclusions and directions for future works end the paper.

2 Environments and Agents

The learning capabilities of adaptive agents are related to their environment, a successful agent in fact depends upon the regularities in its environment. Recently, many authors have addressed the problem of studying the interaction agent/environment rather than studying a specific agent and/or a particular environment separately. In the literature, there have been proposed some classifications for the possible interactions between an agent and its environment.

Wilson in [12] proposes a scheme to classify reinforcement learning environments with respect to the sensory capabilities of the agent. An environment belongs to *Class 0* and *Class 1* if the agent can determine the state of the environment completely that is, the sensory capabilities of the agent are sufficient to determine the entire state of the environment. When in *Class 2* environments, the agent has only partial information about the state of the environment that is, the sensors of the agent are not adequate to determine the state of the environment completely. *Class 2* environments are said to be *partially observable* with respect to the agent, or equivalently are *non-Markovian* with respect to agent's actions. Accordingly, the agent is said to suffer from the *hidden state problem*.

Littman in [7] presents a more formal classification of reinforcement learning environments, that is based on the simplest agent that can achieve optimal performance. Two parameters h and β characterize the complexity of an agent. An (h,β)-environment is best solved by an (h, β)-agent that uses the input information provided by the environment and at most h bits of local storage to choose the action which maximize the next β reinforcements. Previous classification can be easily mapped in this classification: *Class 0* and *Class 1* environments correspond to $(h = 0, \beta = 1)$ and $(h = 0, \beta > 1)$ environments, while *Class 2* environments corresponds to $(h > 0, \beta > 1)$ environments. In this paper we focus on non-Markovian environments, i.e. *Class 2* or $(h > 0, \beta > 1)$ environments, for which the agent needs a sort of memory mechanism to evolve an optimal solution. In the following, we present an example of *Class 2* environment, while we address the interested reader to [8] for more examples.

One of the most popular *Class 2* environments that has been proposed in the literature is `Woods101` [1] shown in Figure 1, that is also known as `McCallum´s Maze` [8]. An agent must learn how to reach food, `F` symbol; it sense the environment by means of eight sensors that tells him the content of the corresponding adjacent cells: food, obstacle or empty if the cell is free. The two cells, evidenced by by the two arrows, are identical from the agent point of view since they return

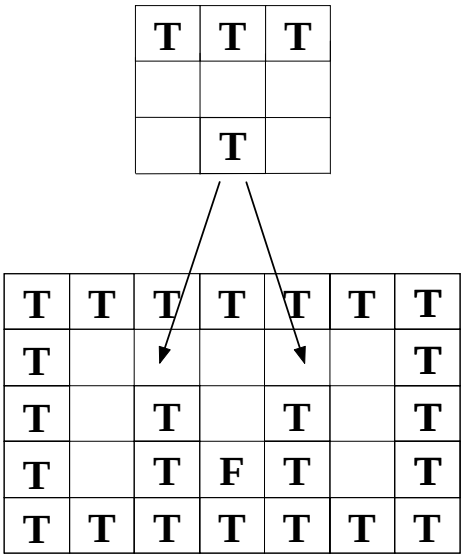

Figure 1: The Woods101 environment. Aliasing positions are indicated by the arrows.

the same sensory configuration, shown at the top of the figure. An optimal solution for `Woods101` requires two distinct actions for each cell: the right cell requires a *go south-west* movement, the left cell requires a *go south-east* movement. The agent when in these cells, cannot choose the optimal action only examining the current sensory inputs, but it needs to remember at least the previous state examined. Optimal policy can thus be obtained with one bit of internal memory that represents previous agent position: when the agent reaches the aliasing position from the left part of the maze, sets the internal register to 0, when it arrives from the right, the agent sets the internal register to 1. Accordingly, when in the aliasing state, the agent chooses the action *go south-east* or *go south west* if the register contains 0 or 1 respectively. Following Littman's classification `Woods101` is thus a $(h = 1, \beta > 1)$-environment.

3 The XCS Classifier System

We now give a brief review of XCS in its most recent version [15]. We refer the interested reader to [14] for the original XCS description or to Kovacs's report [4] for a more detailed discussion for implementors.

Classifiers in XCS have three main parameters: the prediction p_j, the prediction error ε_j and the fitness F_j. Prediction p_j gives an estimate of what is the reward that the classifier is expected to gain. Prediction error ε_j estimates how precise is the prediction p_j. The fitness parameter F_j evaluates the accuracy of the payoff prediction given by p_j and is a function of the prediction error

ε_j. 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 as the fitness weighted average of the classifier predictions that advocate the action a_i in the match set [M]. The value $P(a_i)$ gives an evaluation of the expected reward 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 r_{imm} 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 $[A]_{-1}$. Classifiers parameters are updated as follows.

First, the Q-learning-like payoff P is computed as the sum of the reward received at the previous time step and the maximum system prediction discounted by a factor γ ($0 \leq \gamma < 1$). P is used to update the prediction p_j by the Widrow-Hoff delta rule [11] with learning rate β ($0 < \beta \leq 1$): $p_j \leftarrow p_j + \beta(P - p_j)$. Likewise, the prediction error ε_j is adjusted with the formula: $\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$. Fitness update is slightly more complex. Initially, the prediction error is used to evaluate the classification accuracy κ_j of each classifier as $\kappa_j = exp(\ln \alpha(\varepsilon_j - \varepsilon_0)/\varepsilon_0)$ if $\varepsilon_j > \varepsilon_0$ or $\kappa_j = 1$ otherwise. Subsequently the relative accuracy κ'_j of the classifier is computed from κ_j and, finally, the fitness parameter is adjusted by the rule $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$. 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*. Whenever a new classifier has to be inserted in the population, it is compared to existing ones to check whether there already exists a classifier with the same condition/action pair. If such a classifier exists then the new classifier is not inserted but the *numerosity* parameter of the existing classifier is incremented. If there is no classifier in the population with the same condition/action pair then the new classifier is inserted in the population. 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 [15] and Specify [6]. Subsumption deletion has been introduced to improve generalization capabilities of XCS. Subsumption deletion acts when classifiers created by the genetic component have to be inserted in the population. Offspring classifiers created by the GA are replaced with clones of their parents if: (i) they are specialization of the two parents that is, they are "subsumed" by their parents, and (ii) the parameters of their parents have been updated sufficiently. If both these conditions are satisfied the offspring classifiers are discarded and copies of their parents are inserted in the population; otherwise, the offspring classifiers are inserted in the population. Specify has been proposed to counterbalance the pressure toward generalization in environments that allow few generalizations. Specify acts in the action set when oscillating classifiers are detected; this condition is detected comparing the average prediction error of classifiers in the action set $\varepsilon_{[A]}$ with the average prediction error of classifiers in the population $\varepsilon_{[P]}$. If $\varepsilon_{[A]}$ is twice larger than $\varepsilon_{[P]}$ and the classifiers in [A] have been updated, on average, at least N_{Sp} times then a classifier is randomly selected from [A] with probability proportional to its prediction error. The selected classifier is used to generate one offspring classifier in which each # symbol is replaced with a probability P_{Sp} with the corresponding digit in the system input. The resulting classifier is then inserted in the population and another is deleted if necessary.

4 Design of Experiments

Discussions and experiments presented in this paper are conducted in the well-known "woods" environments. These are grid worlds in which each cell can be empty, can contain a tree, "`T`" symbol, or otherwise food, "`F`". 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 reward; while if the destination cell contains a tree the move does not take place. If the animat has an internal state, it can modify the contents of the register performing an *internal action* in parallel with the *external action* performed in the environment. The set of external actions, in such a case, is enriched with a *null* action so that the animat can "sit and think" that is, it can modify its internal state, without acting in the environment.

Each experiment consists of a number of problems that the animat must solve. For each problem

the animat is randomly placed in a blank cell of the environment. Then it moves under the control of the classifier system until it enters a food cell, eats the food, and receives a constant reward. The food immediately re-grows and a new problem begins. We employed the following exploration/exploitation strategy. Before a new problem begins the animat decide with a 50% probability whether it will solve the problem in exploration or in exploitation. When in exploration, the system decide, 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 reward. When in exploitation the GA does not act and the animat selects the action which predicts the highest reward. In order to evaluate the final solutions evolved, in each experiment exploration is turned off during the last 2500 problems and the system works in exploitation only. 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.

5 XCS in non-Markovian Environments

XCS has no internal message list as Holland's classifier system, thus it only learns optimal policies for Markovian environments in which optimal actions are solely determined by the state of current inputs. When the environment is non-Markovian, XCS converges to a suboptimal policy. As an example consider the `Woods101` environment (see Figure 1 in Section 2) in which two states, indicated by the arrows, return the same sensory configuration to the animat but require two different optimal actions: The right cell requires a *go south-west* movement while the left cell requires a *go south-east* movement. The animat, when in these cells, cannot choose the optimal action only examining the current sensory inputs, but it needs to remember at least the previous state examined. Figure 2 compares the performance of XCS in `Woods101`, solid line, with the optimal performance that the system can achieve with one bit of internal memory, dashed line. As we expected, XCS does not learn an optimal solution for `Woods101`, but it converges to a suboptimal policy, that is displayed using a vector field in Figure 3. Lines in each free position corresponds to the best action that the final policy suggests. As it can be noticed, XCS assigns equal probability to the two actions *go south-east*/*go south-west* when the animat is in the two aliasing positions that is, the animat can go to the food if the correct action is selected, or it can go back to another position for which the optimal action is to return into the aliasing cell. This policy is an efficient stochastic solution for the `Woods101` problem, and is very similar to the one

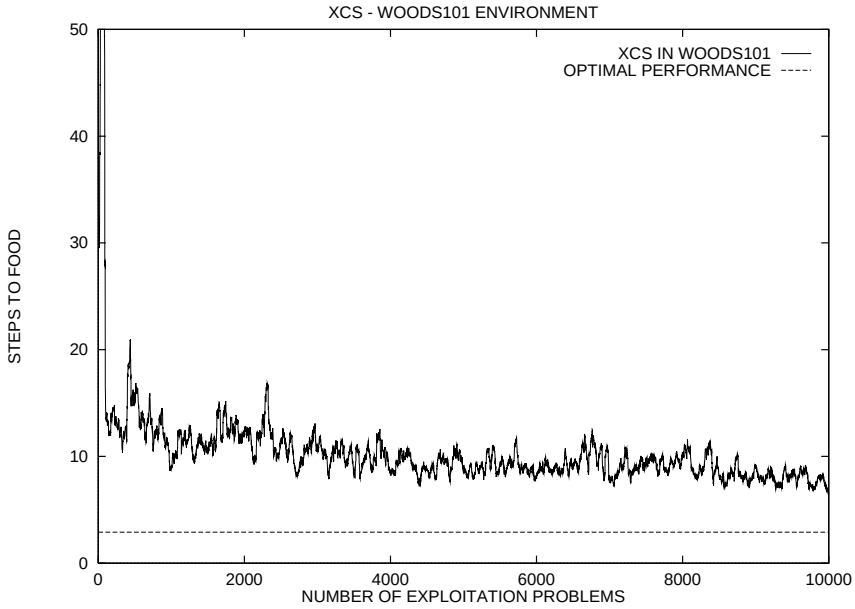

Figure 2: XCS in Woods101.

found for the same environment with ZCS [1]. In order to evolve an optimal solution, XCS needs some sort of memory mechanism.

6 Adding Internal Memory to XCS

We now extend XCS with internal memory as done for ZCS in [1]. An internal register with b bits is added to XCS architecture; classifiers are extended with an internal condition and an internal action that are employed to "sense" and modify the contents of the internal register. Internal condition/action consist of b characters in the ternary 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 internal 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 with "don't care" symbols inserted in internal parts with probability $P_{I\#}$. The new parameter $P_{I\#}$ is introduced to separate the concept of *generalization in the environment* that is based on external condition/action of the classifier and *generalization in the internal memory* of the system. Experimental results, not presented here, suggest in fact that it is useful to distinguish the concept of how many generalizations an environment allows and how many generalizations the corresponding *hidden state* problem allows. In the rest of the paper, we refer to XCS with b bits of

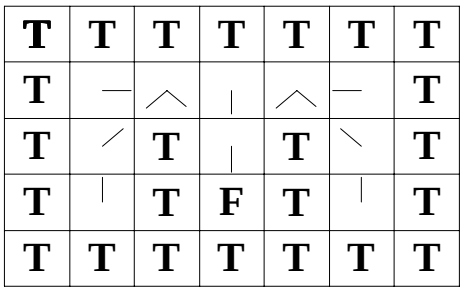

Figure 3: Vector field for the policy in Maze.

internal memory as XCSMb, while we refer simply to XCSM when the discussion is independent of the value b.

XCSM works basically as XCS. At the start of each trial, the internal register is initialized setting all bits to zero. Then at each time step the system input and the contents of the register are used to build the match set [M] containing the classifiers in the populations whose external condition matches the system inputs and whose internal condition matches the contents of the internal register. The system prediction $P(a_i, s_j)$, which evaluates of the expected reward if action pair a_i, s_j is performed, is computed as in XCS. As for XCS, action selection is *deterministic* in exploitation, and *probabilistic* in exploration. Classifiers in [M] which propose the selected action pair are put in the *action set* [A]. The external action and the internal action of the selected action pair are performed in parallel and an immediate reward r_{imm} is returned to the system with a new sensory input. The reward received from the environment is employed to update classifiers parameters corresponding to the previous time step; The credit assignment procedure is the same as for XCS. Specify is applied to bits of both condition parts; bits of internal condition are specified using the contents of the internal register.

7 XCSM in non-Markovian Environments

We apply XCSM to two non-Markovian environments in order to test whether the system can learn optimal policies in environments that are partially observable. First, we consider `Woods101` an environment with two aliasing states that can be solved employing one bit of internal memory. Then, we apply XCSM2 to `Woods102`, an environment with four aliasing states.

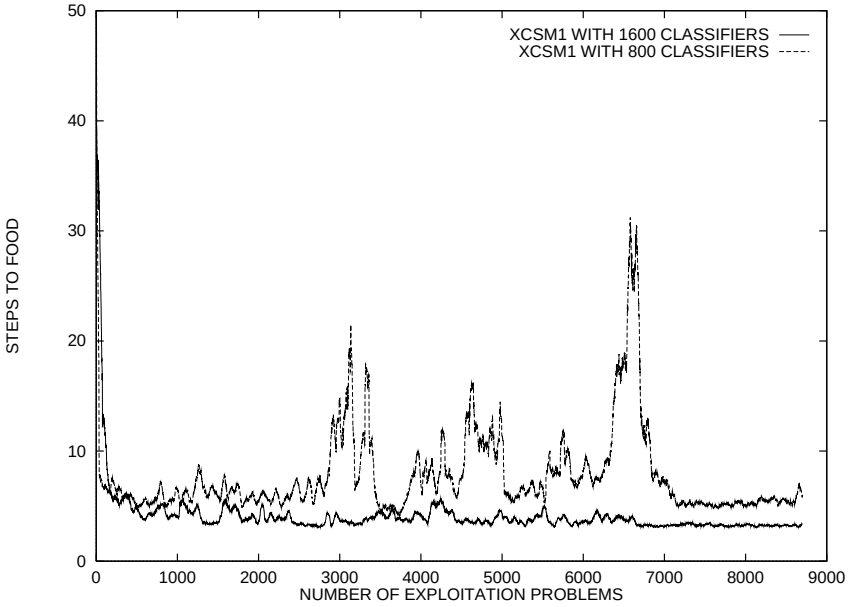

Figure 4: XCSM1 in Maze with 1600 and 800 classifiers.

7.1 XCSM in Woods101

We apply XCSM to the `Woods101` environment presented in Section 5, which has two aliasing states and, as pointed out previously, can be solved by an animat with one bit of internal memory. XCSM1 is applied to `Woods101` with a population of 1600 and 800 classifiers, Subsumption deletion and Specify do not act. XCSM parameters are set as follows: β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_{\#}$=0.2, P_I=10.0, ε_I=0.0, F_I=10.0, $P_{I\#}$=0.5, P_s=0.5. [1] Results for XCSM1 with 1600 classifiers reported in Figure 4 (solid line) show that when exploration acts, before 6500 problems, the performance is slightly suboptimal since the system tends to perform internal actions that are not optimal; then, when exploration stops the system performance becomes very stable. Results for XCSM1 with a population of 800 classifiers, dashed line in Figure 4, show that the system converges to a slightly suboptimal policy that is quite unstable. The analysis of each experiment for XCSM1 with 800 classifiers shows that in most of the experiments the system converges to an optimal solution that is also stable, as the example reported in Figure 5 shows. Sometimes it happens that the convergence is not so stable as in Figure 5 but, at a certain point of the evolution, the population is suddenly corrupted due to unlucky exploration, see Figure 6. Accordingly, XCSM performance drops, higher peak in Figure 6, but then the generalization mechanism of XCS(M) recovers the dangerous situation and the system finally

[1] Some of these parameters have not been presented in the XCS overview but are reported here for completeness. We refer the reader to [14] for a complete discussion of those parameters.

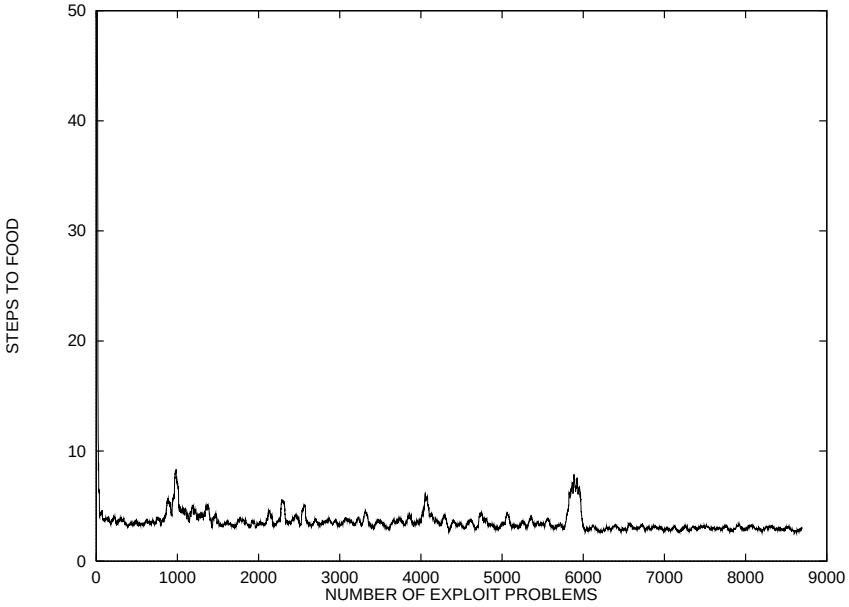

Figure 5: Best convergence for XCSM1 in Maze with 800 classifiers.

converges to optimal performance. Unfortunately there are cases in which, due to a strong genetic pressure, the generalization mechanism is too slow for recovering from a corrupted population, and the system is not able to evolve an optimal solution, as shown in Figure 7. This phenomenon was already reported in [6] for Markovian environments where the Specify operator was proposed for recovering potentially dangerous situations. Accordingly, if we apply XCSM1 with Specify to `Woods101` using 800 classifiers, the system is able to guarantee a stable performance as shown in Figure 8.

In order to understand why a population 800 classifiers can be too small it is useful to analyze how the search space changes when internal memory is introduced. As shown in [14], XCS builds a complete mapping for the function $X \times A \Rightarrow P$ from states/actions pairs to predicted rewards. Since XCSM keeps a all the features of XCS, it still tends to build a complete mapping of such function that for XCSM must also represent the mapping for internal conditions and internal actions. Hence, when one bit of internal memory is employed the function that XCSM1 tries to map becomes $X \times \{0,1\} \times A \times \{0,1,\#\} \Rightarrow P$, thus the search space becomes six times larger. At this point it is interesting to analyze how the number of macroclassifier in the population varies for the experiments with XCSM1 using Specify with 800 classifiers. Results reported in Figure 9 show that, although the maximum number of classifier is set to 800 the population of macroclassifiers oscillates between 245 classifiers and 280 classifiers; while a complete mapping of the function above requires 540 classifier, thus XCSM successfully evolves a compact representation of the

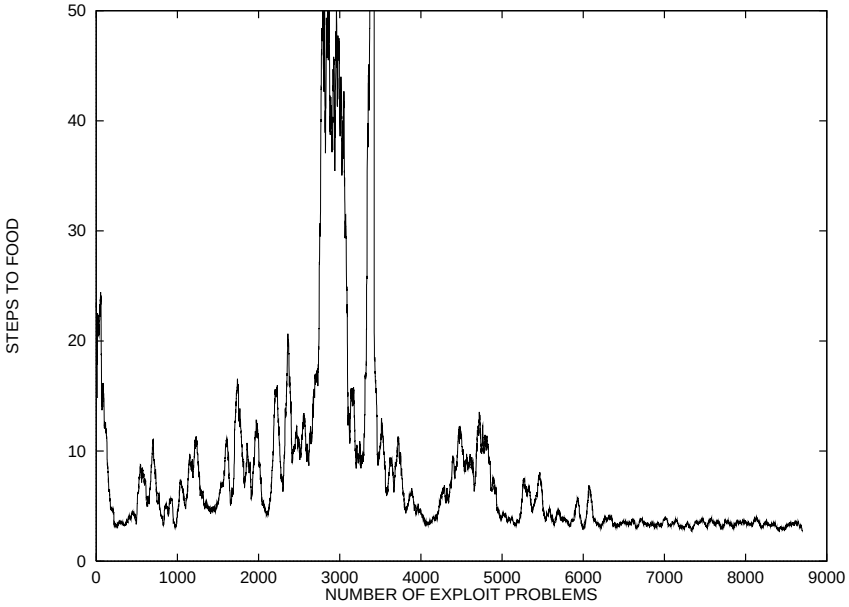

Figure 6: XCSM1 in Maze with 800 classifiers. An unlucky exploration causes a corruption of the population (higher peak) that is successfully recovered by the generalization mechanism of XCS(M).

function which maps state/action pairs in predicted rewards.

7.2 XCSM in Woods102

As a second experiment, we test XCSM in `Woods102` [1], a more difficult environment shown in Figure 10(a). `Woods102` has two types of aliasing states: the former, see 10(b), is encountered in four different positions in the environment, while the latter, see 10(c), is at one of two different positions in the environment. An internal state with two bits, giving four distinct internal states, should be sufficient to disambiguate the aliasing states in order to converge to an optimal policy. XCSM2 is applied to `Woods102` with 2400 classifiers. Results reported in Figure 11 show that XCSM2 cannot converge to a stable policy in `Woods102` when Specify does not act. The system initially reaches a suboptimal policy, first slope, then the learning becomes unstable and the population is rapidly corrupted; finally, when exploration stops at the beginning of the big slope, the performance drops. The analysis of single experiments shows that the performance of the system is very similar for each run, accordingly the behavior of XCSM2 cannot simply depend on unlucky exploration, as for `Woods101`, but it indicates a more general convergence problem. We already analyzed the behavior of XCS for Markovian problems in [5] where we showed that XCS fails to converge to an optimal solution when the agent does not visit all the areas of the environment frequently. Accordingly, we proposed a new exploration strategy called *teletransportation* which

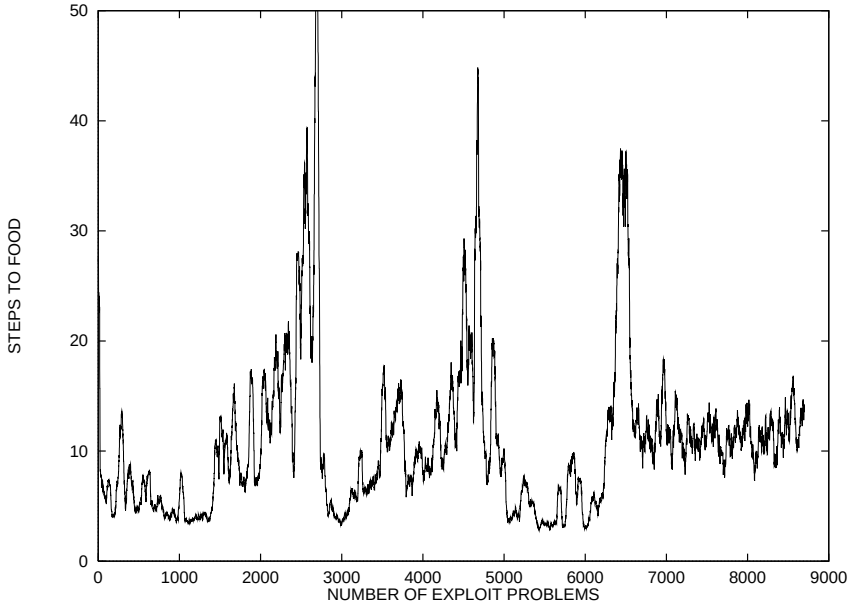

Figure 7: XCSM1 in Maze with 800 classifiers. An unlucky exploration causes a corruption of the population that the system is not able to recover. Accordingly XCSM1 is not able to converge to an optimal solution.

tries to guarantee an uniform exploration of the environment. Teletransportation works as follows: when in exploration, the animat is placed randomly in a blank cell of the environment; then it moves following one of the possible exploration strategies proposed in the literature. If the animat reaches a food cell by a maximum number M_{es} of steps then the exploration ends; otherwise, if the animat does not find food by M_{es} steps, it is "teletransported" to another blank cell and the exploration phase is restarted. This strategy guarantees, for small M_{es} values, that the animat visits all the areas of the environment with the same frequency.

We now extend the results presented in [5] to non-Markovian environments applying teletransportation for solving `Woods102` with XCSM2. Teletransportation for XCSM is implemented as done for XCS except for the fact that in XCSM the internal register is reset every time the animat is teletransported. We apply XCSM2 with teletransportation to `Woods102` with a population of 2400 classifiers. Results reported in Figure 12 show that the system rapidly converges to an optimal policy that is also stable.

7.3 Discussion of the Results

Results presented in this section, show that XCS with the internal memory mechanism proposed by Wilson is able to converge to optimal solutions in non-Markovian environments. When XCSM fails to converge to optimal performance it depends not on the memory mechanism but rather

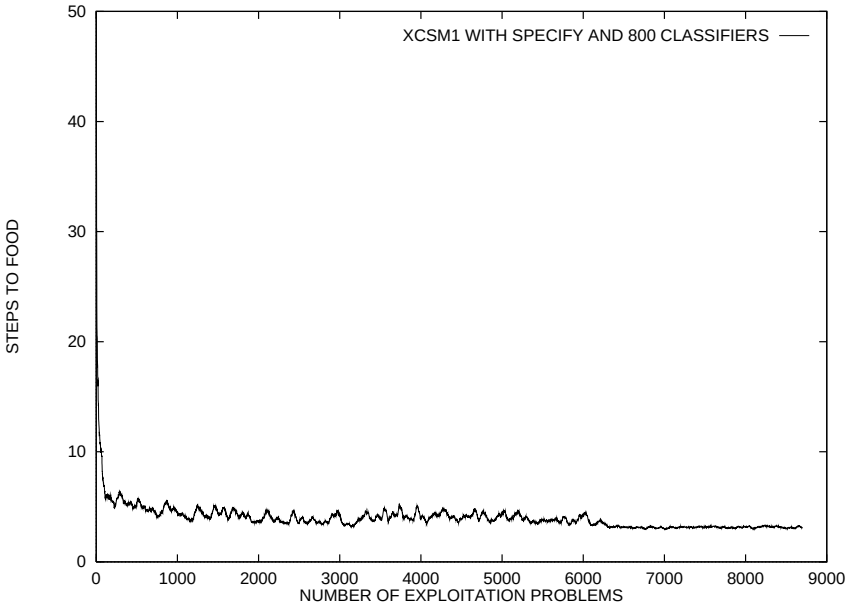

Figure 8: XCSM1 with Specify in Maze with 800 classifiers.

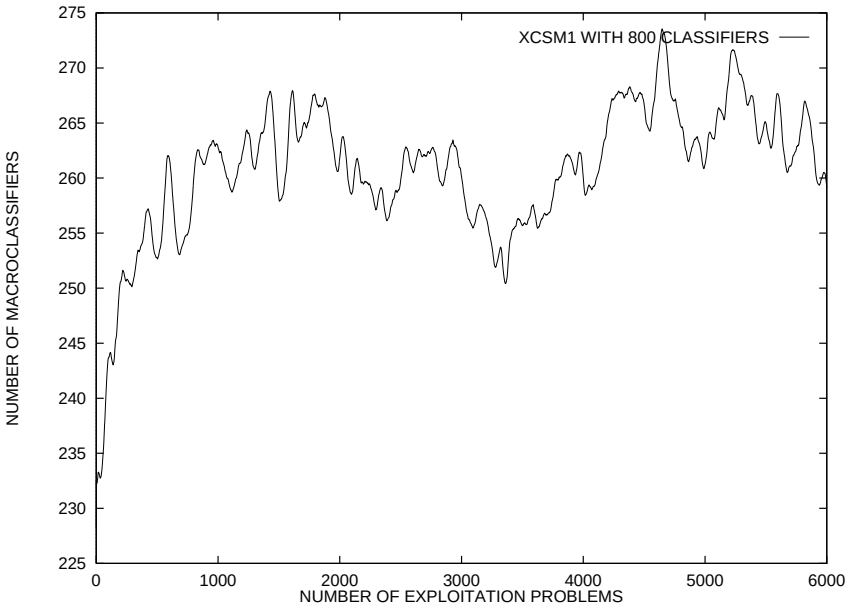

Figure 9: The size of the population in macroclassifiers for XCSM1 with Specify in Maze when 800 classifiers are employed.

T	T	T	T	T	T	T
T		T	F	T		T
T		T		T		T
T						T
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	T	T	T	T	T	T

(a)

	T	
	T	

(b)

T		T
T		T

(c)

Figure 10: The Woods102 environment (a) with the corresponding aliasing states (b) and (c)

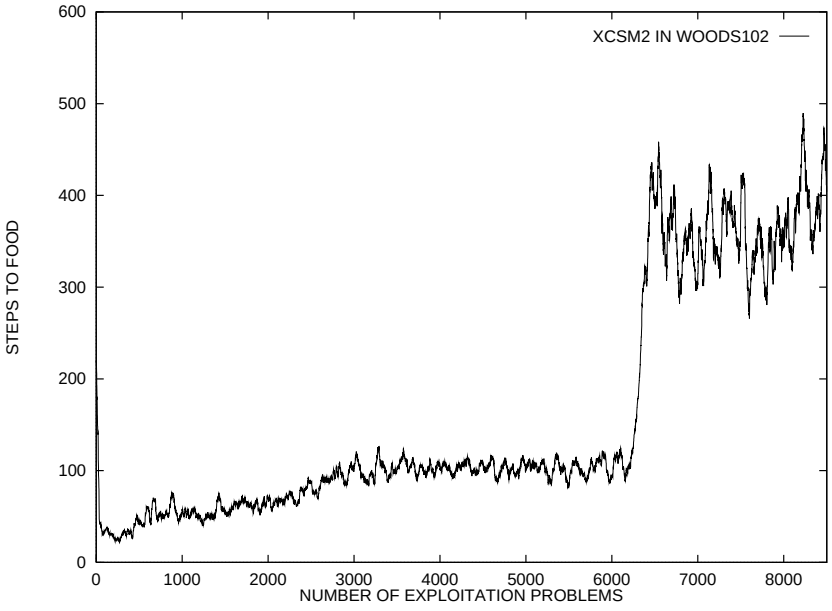

Figure 11: XCSM2 in Woods102.

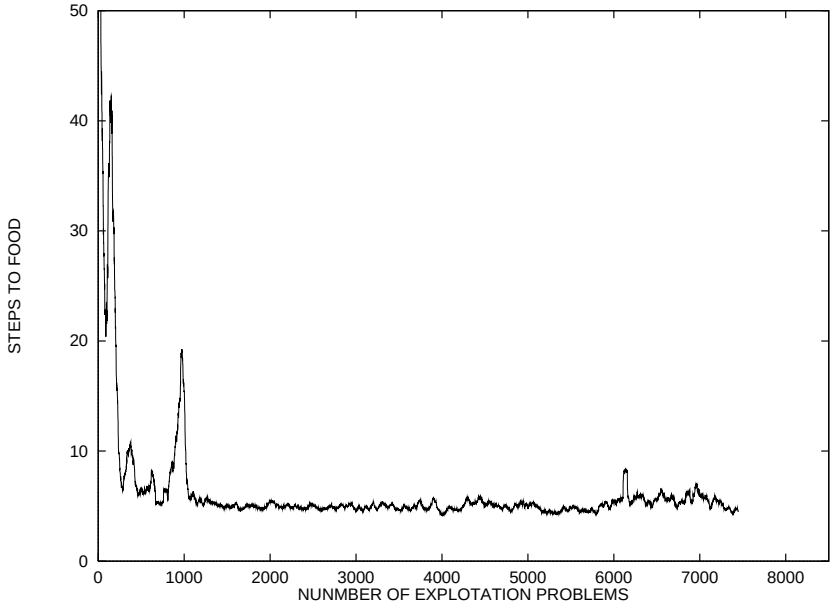

Figure 12: XCSM2 with teletransportation in Woods102.

on more general phenomena. First, unlucky exploration can cause a corruption in the population that sometimes may be not overcome by the generalization mechanism of XCS; in such cases, the Specify operator is useful to counterbalance the effects of unlucky exploration, as the experiments with the `Woods101` environment show. Most important the system fails to converge to optimal performance when, due to the structure of the environment, the agent is not able to visit all the areas of the environment uniformly [5]. Accordingly, the exploration strategy called teletransportation, introduced in [5] for Markovian environments, can be employed to guarantee an uniform exploration in non-Markovian environments in order to evolve an optimal policy, as the experiments with `Woods102` show.

8 Stability of Learning in XCSM

Results presented in [9], pag. 20, for ZCS with internal memory show increasing instability in performance for increasing internal memory sizes. We now apply XCSM to a series of environments using different size of internal memory to test the stability of the system. The hypothesis we test is that the generalization mechanism of XCS can lead to a stable and optimal policy even if large internal memory sizes are employed.

As a first experiment, we apply XCS, XCSM1 and XCSM2 to `Woods1` shown in Figure 13. The right and left edges of the grid are connected and so are the top and the bottom edges. `Woods1` is a *Class 1* environment that does not require an internal state and is easily solved by XCS. The

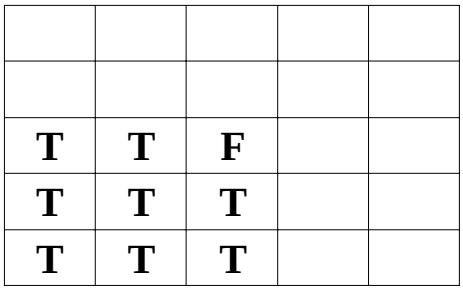

Figure 13: The Woods1 Environment.

hypothesis we test is that XCSM is able to learn a stable and optimal policy for large b values even in an environment that does not require internal state. XCSM parameters are set as follows: N = 800, β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_\#$=0.5, P_I=10.0, ε_I=0.0, F_I=10.0, $P_{I\#}$=0.3. Figure 14, reports the results of the experiments for XCS, XCSM1 and XCSM2; curves are averaged over ten runs. XCSM learns how to reach food in an optimal way even when three bits of memory are employed. It is worth noticing that even if XCSM is applied to search spaces of very different sizes, due to the generalization over internal memory, there is almost no difference in the convergence rate between XCS and XCSM2. The analysis of final populations reveals that the slight degradation in performance when b increases mainly depends on external null actions that sometimes the system performs. Null actions are "useless" in `Woods1` since the environment would not require any operation on the internal state and consequently any null action. But XCS tends to evolve complete input/output mappings and allocates resources to all the possible external actions, even to the null action. Accordingly, some null action is chosen during random exploration and gets system resources.

The second experiment consists of applying XCSM1, XCSM2 and XCSM3 to `Woods101`. Parameters are set as for the previous series of experiments except for the following ones: N = 1600, $P_\#$ = 0.2, $P_{I\#}$ = 0.3. Results for `Woods101`, reported in Figure 15, are similar to the ones for `Woods1`. Generalization mechanism of XCS, and consequently of XCSM, successfully generalizes over internal memory bits leading to almost no difference in performance when more bits of internal memory are employed.

Results for both environments confirm that XCSM learns a stable and optimal policy even when a larger number of internal memory bits is employed. At this point is worth noticing that even three bits of internal memory may appear only a few, most of the environments presented in literature requires only one or two bits to disambiguate aliasing situations [14, 1].

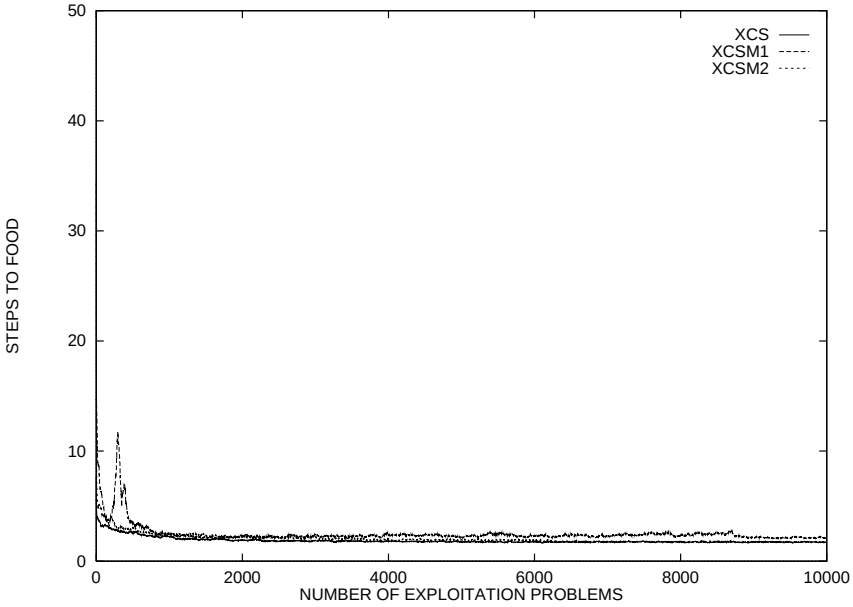

Figure 14: XCS, XCSM1, XCSM2 and XCSM3 in Woods1.

9 Sequences of Actions in the Internal Memory

In the previous sections we applied XCSM to environments in which the optimal solution requires the agent to visit at most one aliasing state before it reaches the food, and the goal state is very near to the aliasing cells. The optimal policy for such type of environments is quite simple, and it usually requires only one action on the internal memory in order to reach the goal state. We now want to test XCSM in an environment where (i) the animat has to evolve an optimal strategy to visit more aliasing positions before it can eat; and (ii) longer sequences of actions must be taken to reach the goal state. The optimal solution for this type of environment can be far more complex than that for environments previously presented in the literature of classifier systems. Because of (i) the animat have to perform a *sequence* of actions in the internal memory instead of a single action; while, as shown in [1], the longer the sequence of action that the agent needs to reach the goal state is, the more difficult is the problem to solve.

`Maze7` is a simple environment, see Figure 16, which consists of a linear path of nine cell to food and it has two aliasing cells, indicated by two dashed circles. Nevertheless, `Maze7` is more difficult than the previous ones in that: (i) it has two positions, at the end of the corridor, from which two aliasing states must be visited to reach the food cell; moreover (ii) it requires a long sequence of action to reach food. We apply XCSM1 with Specify operator to `Maze7` with a population of 1600 classifiers. Results are reported in Figure 17; as in the previous experiments we presented, during

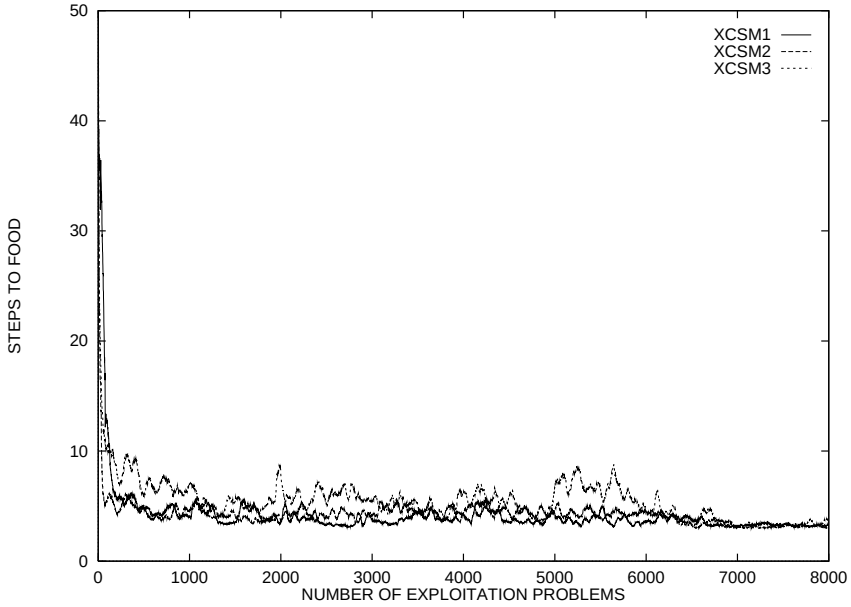

Figure 15: XCSM1, XCSM2 and XCSM3 in Maze.

T	T	T	T	T
T				T
T		T		T
T		T		T
T		T		T
T	F	T	T	T

Figure 16: The Maze7 Environment.

the last 2500 problems exploration is turned off. Figure 17 shows that while exploration acts the system cannot converge to an optimal solution, but when the final population is evaluated turning off exploration, at beginning of the peak, XCSM1 evolves an optimal solution to the problem. The analysis of the population dynamic shows two main facts. First, when exploration acts the system is not able to learn an optimal policy for some of the positions at the end of the corridor. Thus, the system learns a policy that is not globally optimal, hence the performance of XCSM1 drops when an experiment starts in positions for which the optimal policy is not evolved and the overall performance oscillates. Second, when exploration stops most of the classifiers evolved during the previous phase are eliminated, the final policy is thus formed by classifiers created by the covering operator. Accordingly, if we apply XCSM1 to `Maze7` only in exploitation, that is the GA does not work and always the best action is selected, XCSM1 easily converges to a suboptimal solution for `Maze7`, see the solid line in Figure 18. The analysis of single runs shows that in many cases

XCSM1 converges to the optimal performance, lower dashed line, while seldom the performance is suboptimal, upper dashed line.

`Maze7` is a simple problem for XCSM, indeed it is solved using a very basic version of XCSM. Unfortunately, a system which only relies on covering and exploitation is unacceptable since it does not employ any search procedure and thus, in general, cannot guarantee the convergence to an optimal policy; moreover it throws away all the generalization power of XCS. A further analysis of the behavior of XCSM in `Maze7` suggests that the exploration strategies currently employed in the study of XCS are too simple for XCSM. Exploration in XCS is only done "*in the environment*" and relies both on the structure of the environment and on the strategy employed. Instead in XCSM exploration is also done "*in the internal memory*"; accordingly, the exploration of the possible policies for updating the internal register only depends on the agent strategy, that in XCS(M) is random. Consequently, as the sequence of internal actions that must be performed to reach the goal state becomes more complex, the probability that a specific policy is experimented dramatically decreases. We thus conclude that more adequate exploration strategies should be investigated in order to develop better classifier system.

Another set of experiments for XCSM with two and three bits of memory, not reported here for lack of space, shows that a greater memory size reduces the peak, see Figure 17, which separates the explore/exploit phase from the exploitation phase, although the final performance is still slightly sub-optimal. These results can be easily explained considering how the problem changes when more bits of memory are employed. As the size of the internal register increases, the state/action space becomes larger but also there are more admissible optimal policies, which means that there are more possible solutions the system can experiment. This suggest an important aspect of XCSM that will need further analysis: in environments which require complex policies for the internal memory, XCSM is able to use redundant bits of internal memory in order to evolve more stable solutions.

10 Conclusions

We have implemented and tested XCS when the memory mechanism proposed by Wilson is added. We applied XCS with internal memory, we call it XCSM, with different sizes of internal memory to non-Markovian environments with two and four aliasing positions. The results we present show that, in a simple environment, such as `Woods101`, XCSM converges very easily to an optimal solu-

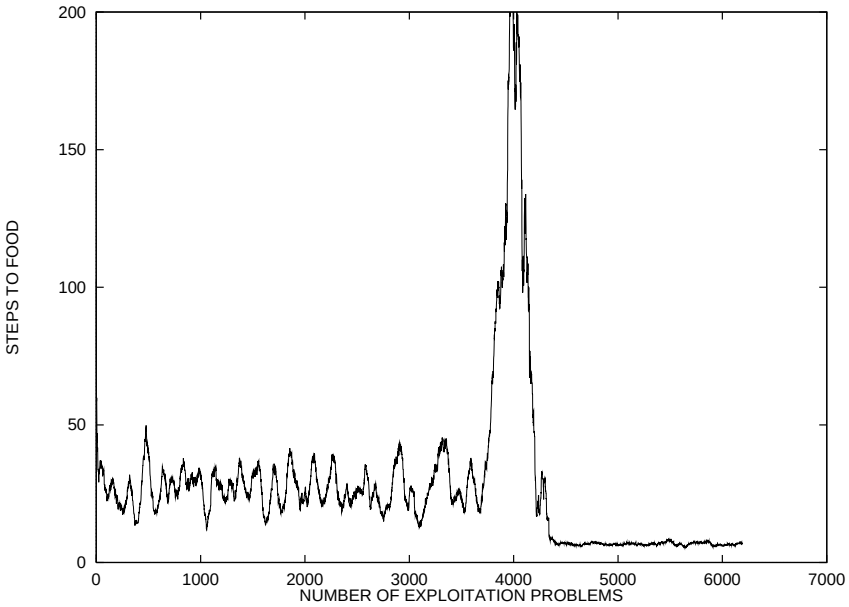

Figure 17: XCSM1 with Specify in Maze7.

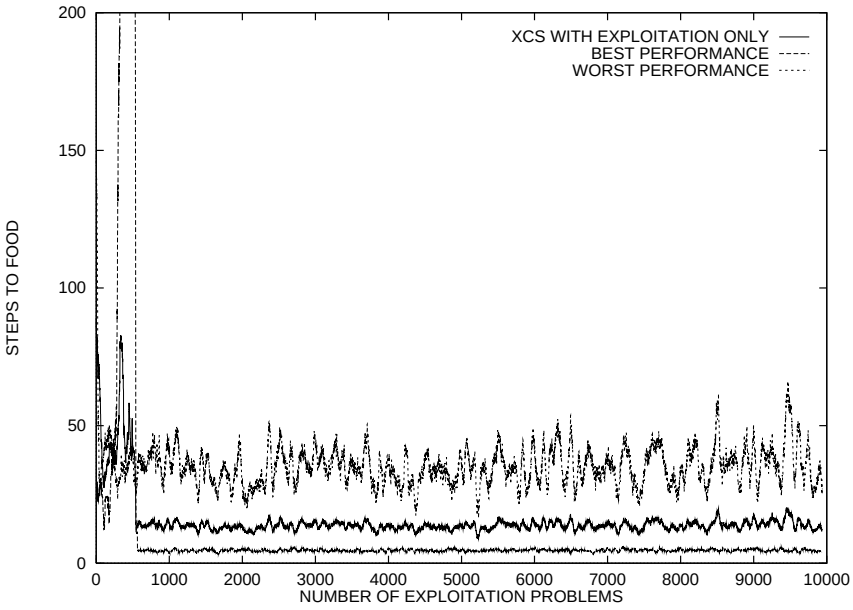

Figure 18: XCSM1 with Specify in Maze7 working in exploitation only.

tion that has also a compact representation. When the system faces more complex environments, special operators or different exploration strategies are needed to guarantee convergence. We then analyze the behavior of XCSM in an environment for which the optimal policy requires sequences of actions in the internal register; experimental results show that, even if the problem is simple, XCSM is not able to evolve an optimal policy for all the positions in the environment. Our analysis suggests that the exploration strategies currently employed in the study of XCS are too simple to guarantee the convergence of XCSM to a policy that is optimal for all the areas of the environment. Accordingly we suggest that better exploration strategies should be developed in order to obtain better classifier systems. Finally, we show that XCSM performance is very stable with respect to the size of the internal register employed; hence, in environments which require complex policies, XCSM is able to use redundant bits of memory in order to evolve more stable solutions.

Acknowledgments

I wish to thank Stewart Wilson for reviewing preliminary versions of this paper and for the many interesting discussions. Many thanks also to Marco Colombetti and Marco Dorigo for encouraging my work.

References

[1] Cliff, D., and Ross, S. Adding memory to ZCS. *Adaptive Behaviour 3*, 2 (1994), 101–150.

[2] Holland, J. H. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[3] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996).

[4] Kovacs, T. Evolving optimal populations with XCS classifier systems. Tech. Rep. CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Avaiable from the technical report archive at http://www/system/tech-reports/tr.html.

[5] Lanzi, P. L. A Model of the Environment to Avoid Local Learning with XCS in Animat Problems (An Analysis of the Generalization Mechanism of Wilson's Classifier System). Tech. Rep. N. 97.46, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1997. Available at http://www.elet.polimi.it/ lanzi/listpub.html.

[6] Lanzi, P. L. A Study on the Generalization Capabilities of XCS. In *Proceedings of the Seventh International Conference on Genetic Algorithms* (1997), Morgan Kaufmann.

[7] Littman, M. L. An optimization-based categorization of reinforcement learning environments. In *In From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior* (1992), J.-A. M. H. Roitblat and S. W. eds., Eds., The MIT Press/Bradford Books.

[8] McCallum, R. A. 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) 26*, 3 (1996).

[9] Ross, S. Accurate reaction or reflective action? experiments in adding memory to wilson's ZCS. University of Sussex, 1994.

[10] Watkins, C. Learning from delayed reward. PhD Thesis, Cambridge University, Cambridge, England, 1989.

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

[12] Wilson, S. W. The animat path to AI. In *From Animals to Animats: Proceedings of the First International Conference on the Simulation of Adaptive Behavior* (1991), The MIT Press/Bradford Books.

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

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

[15] Wilson, S. W. Generalization in XCS. Unpublished contribution to the ICML '96 Workshop on Evolutionary Computing and Machine Learning. Avaiable at http://netq.rowland.org/sw/swhp.html, 1996.