Adding Memory to XCS

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

Abstract— **We add internal memory to the XCS classifier system. We then test XCS with internal memory, named XCSM, in non-Markovian environments with two and four aliasing states. Experimental results show that XCSM can easily converge to optimal solutions in simple environments; moreover, XCSM's performance is very stable with respect to the size of the internal memory involved in learning. However, the results we present evidence that in more complex non-Markovian environments, XCSM may fail to evolve an optimal solution. Our results suggest that this happens because, the exploration strategies currently employed with XCS, are not adequate to guarantee the convergence to an optimal policy with XCSM, in complex non-Markovian environments.**

I. Introduction

XCS is a classifier system proposed by Wilson [10] 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. 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 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 and form the most general class of environments. When in non-Markovian 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 in [10] 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 [9] for ZCS the *"zeroth level"* classifier system from which XCS was derived. The proposal was validated for ZCS in [1] 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; while (ii) when size internal memory grows the learning become unstable.

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 memory, XCSM for short, 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. However, as we finally show, XCSM may fail to evolve an optimal solution in complex partially observable environments. Our results suggest that the exploration strategies currently employed with XCS are not adequate to guarantee the convergence to optimal policies in complex problems.

The paper is organized as follows. Section II briefly overviews XCS, while Section III introduces the "woods" environments and the design of experiments. Section IV discusses the performance of XCS in non-Markovian environments. Wilson's proposal and our implementation of XCS with internal memory, we call it XCSM, is presented in Section V. In Section VI, XCSM is applied to two non-Markovian environments, `Woods101` and `Woods102`. The stability of learning of XCSM is then discussed in Section VII, while in Section VIII the previous results are extended applying XCSM to a more difficult environment, that we call `Maze7`. Finally, conclusions and directions for future works are drawn in Section IX.

II. The XCS Classifier System

XCS differs from Holland's classifier system for two main aspects. First, in XCS classifier fitness is based on the accuracy of the prediction instead of the prediction itself. Accordingly, the original *strength* parameter is replaced by three different parameters that are updated using a Q-learning like mechanism [7], [10]: (i) the prediction p_j which gives an estimate of what is the payoff that the system is expected to gain when the classifier is used; (ii) the predic-

tion error ε_j estimating how much precise is the prediction p_j; finally (iii) the fitness F_j that evaluates the accuracy of the prediction given by p_j and therefore is a function of the prediction error ε_j. Second, XCS has a very basic architecture with respect to the original framework. Specifically, XCS has no internal message list, and no other memory mechanisms. XCS works as follows.

At each time step the system input is used to build the 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 payoff expected 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 then 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 [8] using a Q-learning-like technique [10].

The genetic algorithm in XCS is applied to the classifiers in 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*, (micro) classifiers XCS has to deal with.

Since XCS was presented, two genetic operators have been proposed as extensions to the original system: Subsumption deletion [11] and Specify [5]. 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.

III. 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 constant reward; if the destination cell contains a tree, the move does not take place. If the animat has internal memory, 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 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 an empty cell of the environment. Then it moves under the control of the system until enters a food cell, eats the food receiving 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 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 0.3), whether to select the action randomly or to choose the action that predicts the highest payoff. When in exploitation the GA does not act and the animat always selects the action corresponding to the highest prediction. 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.

IV. 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 (also known as McCallum's `Maze` [?]), shown in Figure 1, 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; 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.

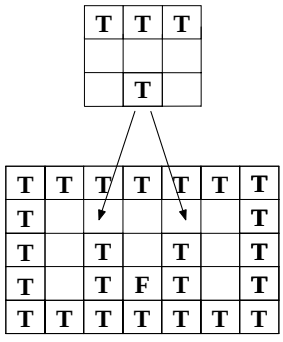

Fig. 1. The `Woods101` environment. Aliasing positions are indicated by the arrows.

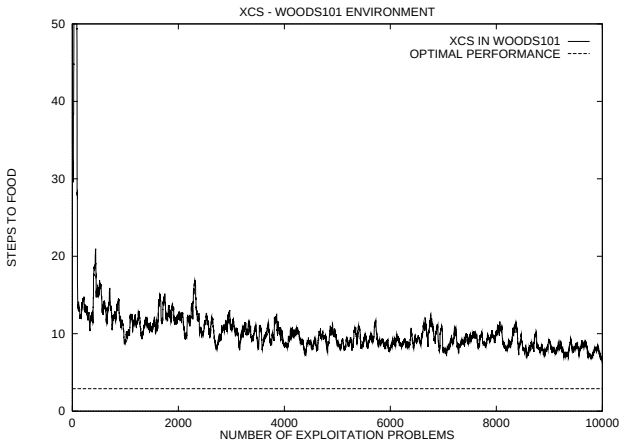

Fig. 2. XCS in `Woods101`.

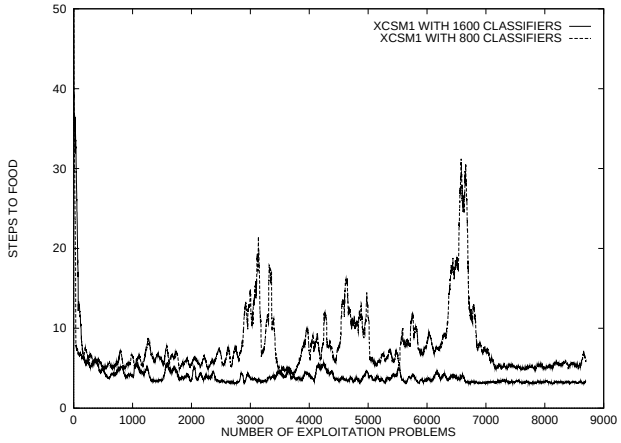

Fig. 4. XCSM1 in `Woods101` with 1600 and 800 classifiers.

Figure 2 compares the performance of XCS in `Woods101`, solid line, with the optimal performance, 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 found for the same environment with ZCS [1].

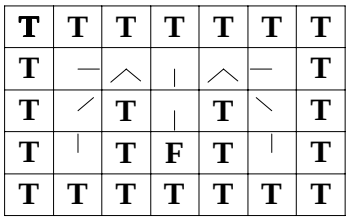

Fig. 3. Vector field for the policy in `Woods101`.

In order to evolve an optimal solution in `Woods101`, XCS needs some sort of memory mechanism. Optimal policy for `Woods101` can in fact 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 bit to 0, when it arrives from the right, the agent sets the bit to 1. Accordingly, when in the aliasing state, the agent is able to choose the action *go south-east* or *go south west* if the memory bit contains 0 or 1 respectively.

V. 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, which are 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, 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. At each time step, the match set [M], the prediction array, and the action set [A] are build as in XCS. The only difference is that in XCSM the internal condition is considered when building [M], and the internal action is used to build 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.

VI. 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 apply XCSM to the `Woods101` environment, seen in Section IV, 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, Specify does not act. Results reported in Figure 4 show that XCSM1 learns an optimal policy with a population of 1600 classifiers while with 800 classifiers the system converges to a slightly suboptimal policy. But `Woods101` is a very simple environment consisting only of 10 sensory configurations and we would expect 800 classifiers to be enough to evolve an optimal policy. However, a limited population size may increase the genetic pressure toward more general classifiers that, as

noticed in [5], may prevent the system from converging to optimal performance. Specify has been introduced in [5] to counterbalance generalization mechanism when such type of situations occur. Accordingly, when we apply XCSM1 with Specify to `Woods101` using a population of 800 classifiers, the system converges to an optimal solution, as Figure 5 reports.

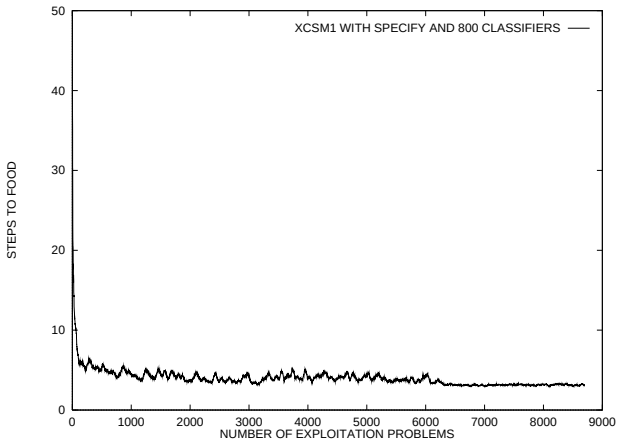

Fig. 5. XCSM1 with Specify in `Woods101` with 800 classifiers.

As a second experiment, we test XCSM in `Woods102` [1], a more difficult environment shown in Figure 6(a). `Woods102` has two types of aliasing states. The former, see 6(b), is encountered in four different positions in the environment; the latter, see 6(c), is at one of two different positions in the environment. An internal state with two bits, giving 4 distinct internal states, should be sufficient to disambiguate the aliasing states in order to converge to an optimal policy. XCSM2 and XCSM2 with Specify are applied to `Woods102` with 1600 classifiers. Experimental results reported in Figure 7 show that XCSM2 (solid line) 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. On the contrary, XCSM2 with Specify successfully evolves an optimal solution for `Woods102`.

Results presented in this section, confirm that XCS with the internal memory mechanism proposed by Wilson is able to converge to optimal solutions in non-Markovian environments. Moreover, they also confirm the early results presented in [5] where the authors observed that a strong genetic pressure can prevent the system from converging to an optimal solution. Accordingly, Specify has to be employed in order to guarantee the convergence to an optimal performance.

VII. Stability of Learning with XCSM

Results presented in [6] for ZCS with internal memory showed increasing instability in performance for increasing memory sizes. We now apply XCSM to `Woods101` using different sizes of internal memory to test the stability of the system. The hypothesis we test is that the generalization

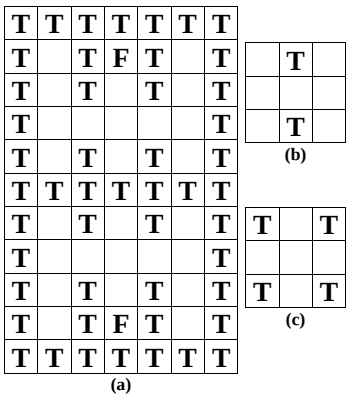

Fig. 6. The `Woods102` environment (a) with the corresponding aliasing states (b) and (c)

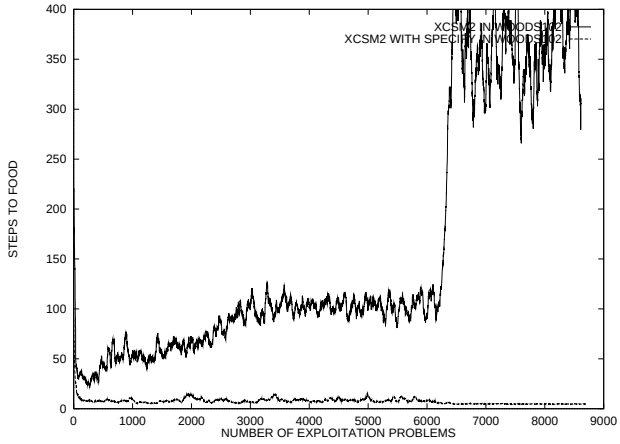

Fig. 7. XCSM2 in `Woods102` without Specify (upper solid line) and with Specify (lower dashed line).

mechanism of XCS can lead to a stable and optimal policy even if redundant bits of internal memory are employed. We apply XCSM1, XCSM2 and XCSM3 to `Woods101` using 1600 classifiers. Results reported in Figure 8 show that 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 between the final solutions evolved.

We have extended these results in [4], where we have applied XCSM with increasing sizes of internal memory to other environments. Results, not reported here for the lack of space, confirm that XCSM is able to learn a stable and optimal policy even when a redundant number of internal memory bits is employed. Finally, we wish to point out that, even if an internal state consisting of three bits may appear very small, most of the environments presented in the literature require only one or two bits of internal memory in order to disambiguate aliasing situations [1].

VIII. A More Difficult Environment

In the previous sections we applied XCSM to environments in which the optimal solution requires the agent to

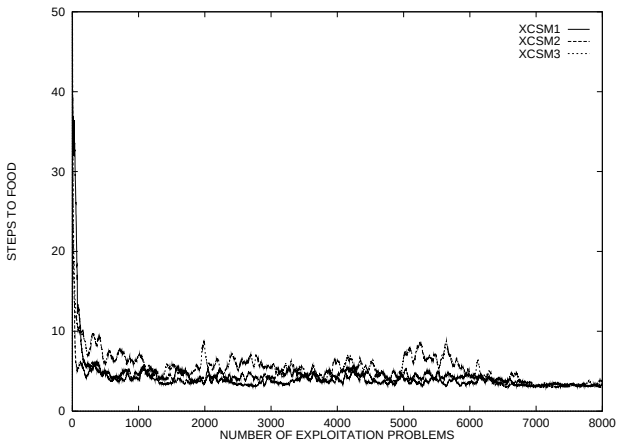

Fig. 8. XCSM1, XCSM2 and XCSM3 in `Woods101`.

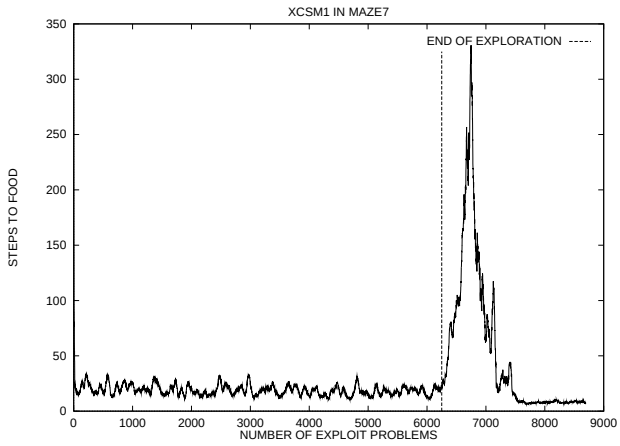

Fig. 10. XCSM1 with Specify in `Maze7`.

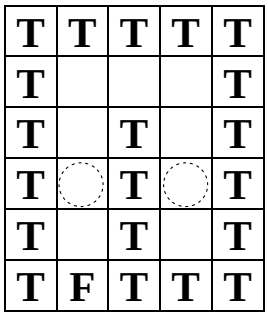

Fig. 9. The `Maze7` Environment. Aliasing positions are indicated by dashed circles.

visit at most one aliasing state before it reaches the food, and the goal state is very near aliasing cells.

The optimal policy for such type of environments is usually quite simple. Accordingly, we now want to test XCSM in a more difficult environment in that (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 environments can be far more complex. Since the animat visits more aliasing cells before it reaches the goal state, it may need to perform sequences of actions in the internal memory. Moreover, as shown in [1], the longer the sequence of action the agent must perform to reach the goal state is, the more difficult is the problem to solve.

`Maze7` is a simple environment, see Figure 9, 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 environment previously considered 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 10; as in the previous experiments we presented, during the last 2500 problems exploration is turned off. Figure 10 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 evidences that, when exploration acts, the system is not able to learn an optimal policy to reach the goal state from 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, see the vertical dashed line in Figure 10, 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 [10]. Accordingly, XCSM starts replacing such low predictive classifiers through covering. The final policy, at the end of the peak, is thus built by classifiers created by the covering operator.

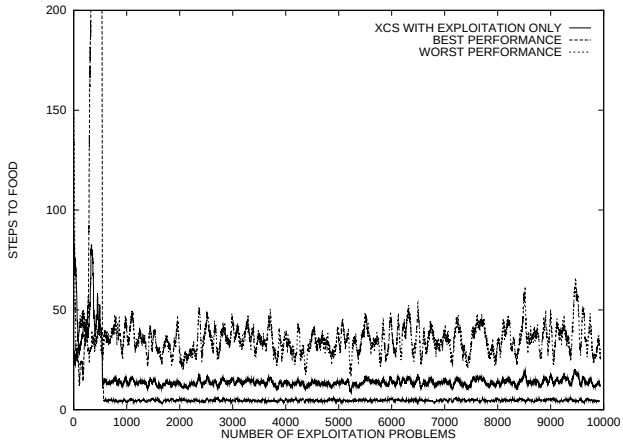

Fig. 11. XCSM1 with Specify in `Maze7` working in exploitation only.

Therefore, we apply XCSM1 to `Maze7` only in exploitation, that is the GA does not work and always the best action is selected. XCSM1 performance is reported in Figure 11 with a solid line, while the two dashed lines show the worst and the best performance over the ten runs. Results show that XCSM1 easily converges to a suboptimal

solution for `Maze7` when all the problems are solved in exploitation. The analysis of single runs also shows that in many cases XCSM1 converges to the optimal performance, lower dashed line, while seldom the performance is suboptimal, upper dashed line. These results suggest that `Maze7` is a simple problem for XCSM, indeed it is solved using a very basic version of XCSM. However, the results for XCSM working in exploitation only suggest that the exploration strategies currently employed with XCS are too simple for XCSM. In XCS in fact, exploration is done "*in the environment*," and relies on both the structure of the environment and on the strategy employed. Conversely, in XCSM, the exploration is also done "*in the memory*." This type of exploration only relies on the agent's exploration strategy, accordingly, if the strategy is not adequate it cannot guarantee that the animat will be able to evolve a stable an optimal solution for complex problems.

IX. Conclusions

We have implemented and tested XCS when internal memory is added. XCS with internal memory, we call it XCSM, has been applied with different sizes of internal memory to non-Markovian environments with two and four aliasing positions. Experimental results we present show that, in simple environments XCSM converges to an optimal solution, even if redundant bits of memory are employed. Most important, experiments with `Maze7` show that in complex problems the XCSM's exploration strategy currently employed, is not adequate to guarantee the convergence to an optimal solution. Therefore other strategies should be investigated in order to develop better classifier systems.

Acknowledgments

I wish to thank Marco Colombetti and Stewart Wilson for the many interesting discussions and for reviewing the early versions of this paper. Many thanks also to to the three anonymous reviewers for their comments.

References

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

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

[3] Leslie Pack Kaelbling, Michael L. Littman, and Andew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[4] Pier Luca Lanzi. Experiments on adding memory to XCS. Technical Report N. 97.45, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1997. Available at http://www.elet.polimi.it/ lanzi/listpub.html.

[5] Pier Luca Lanzi. A Study on the Generalization Capabilities of XCS. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

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

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

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

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

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

[11] Stewart W. Wilson. Generalisation in evolutionary learning. In *Proc. Fourth European Conf. on Artificial Life (ECAL97)*, 1997.