Optimal Classifier System Performance in Non-Markov Environments

Pier Luca Lanzi
Artificial Intelligence & Robotics Project
Dipartimento di Elettronica e Informazione
Politecnico di Milano
lanzi@elet.polimi.it

Stewart W. Wilson
Prediction Dynamics
30 Lang Street
Concord, MA 01742 USA
wilson@prediction-dynamics.com

July 8, 1999

Abstract

Wilson's (1994) bit-register memory scheme was incorporated into the XCS classifier system and investigated in a series of non-Markov environments. Two extensions to the scheme proved important for reaching optimal performance in the harder environments. The first was an exploration strategy in which exploration of external actions was probabilistic as in Markov environments, but internal "actions" (register settings) were selected deterministically. The second was use of a register having more bit-positions than were strictly necessary to resolve environmental aliasing. The origins and effects of the two extensions are discussed.

1 Introduction

The learning capabilities of adaptive agents are related to their perception of the environment. There are cases in which the agent's immediate sensations provide all the information that is necessary to choose the best action in every situation. Such an environment is said to be *completely observable* with respect to the agent's sensors, or *Markov*. When the agent's sensations convey only partial information about the environment, there may be different situations which appear *identical* to the agent but require *different* optimal actions. When this happens the agent cannot decide on the best action by considering only its current sensory inputs. Such an environment is said to be *partially observable* with respect to the agent sensors, or *non-Markov*, and the agent is said to suffer from a *perceptual aliasing problem*.

A very simple example of a situation involving perceptual aliasing is the following (Lin 1993). Imagine a gift packing task in which a gift must be placed in an initially closed box after which the box is wrapped. Initially the box is closed with no gift inside and must be opened and filled. Later, with gift inside and again closed, it must be wrapped. In both cases, the agent's sensors see a closed box, but the actions called for are different. The environment is non-Markov.

Note that the Markov / non-Markov distinction very much depends on the agent's sensors. Environments can be "made" Markov if there are enough sensors. In the gift packing task we might give the agent the ability to weigh the box so that it can sense whether a closed box is empty or not. The

environment becomes Markov. Thus it is not the environment itself that is Markov or non-Markov, but the environment as sensed by the system. In speaking of an environment as Markov or non-Markov, we shall mean the environment as observed through the system's sensors.

However, there are cases in which perceptual aliasing cannot be solved by increasing the agent's sensory capabilities. For instance, in robot navigation tasks, T junctions between corridors may look the same no matter what sensors the agent has. If we cannot make the environment Markov, we can give the agent a memory mechanism to cope with the lack of sensory information. The idea is that, since the agent cannot rely on its current inputs to determine the best action, it can use memory to encode something about the past which can be used in conjunction with current inputs to take correct decisions. For instance, in the gift packing task the agent could use memory to remember whether it had already put the gift inside the box. The environment plus memory would then be Markov.

Holland's classifier system has an internal message list, where the system can in principle store information from previous time-steps which it can reuse on later time steps. Thus, the system should be able to exploit the message list to solve problems which require internal memory, e.g., non-Markov environments. Up to now, however, Holland's system has shown only limited success on non-Markov problems (Robertson and Riolo 1988; Smith 1994).

Wilson (1995) introduced a simplified classifier system model, XCS, that has been shown to reach optimal performance in Markov problems but does not have any form of internal memory. Accordingly, XCS does not learn optimal solutions in non-Markov environments. However, Wilson (1994) suggested that if internal memory were needed, the coupling between posting and reading classifiers would be improved if the memory were embodied in a simple bit register instead of a list of messages. He also observed that in many problems, decisions needing past information often require only one or a few bits.

This paper reports on experiments in which the bit-register memory scheme was incorporated into XCS and investigated in a series of four increasingly complex grid-like non-Markov environments, termed `Woods101`, `Maze7`, `Woods101`$\frac{1}{2}$, and `Woods102`. Direct implementation of the bit-register ("XCSM") gave optimal performance in `Woods101`, but not in `Maze7`. Phenomena observed with `Maze7` led to an extension to the system, *compound exploration*, in which the exploration strategy chose internal actions (register settings) deterministically, while selection of external actions remained probabilistic. The resulting system, "XCSMH", produced optimal performance in `Maze7`. XCSMH was used with the third and fourth environments also, but performance there was again short of optimal, and led to a second extension in which the size of the memory register was made somewhat greater than the number of bits strictly necessary to resolve the aliasing. This *register redundancy* produced optimal performance in `Woods101`$\frac{1}{2}$ and `Woods102`. Conceptually, the two extensions and their results are seen from the point of view of reliably evolving a stable internal coding, or language, to disambiguate aliased states.

The remainder of the paper is organized as follows. Sections 2 and 3 present the perceptual aliasing problem and review related work on it. Section 4 reviews XCS, describes the bit-register implementation XCSM, and the design of experiments. Experiments with XCSM in `Woods101` and `Maze7` are presented in Sections 5 and analyzed in Section 6. Section 7 motivates and introduces compound exploration. The resulting system, XCSMH, is tried on `Maze7` and `Woods101`$\frac{1}{2}$ in Sections 7 and 8. Register redundancy is motivated and introduced in Section 8, where it permits XCSMH to solve `Woods101`$\frac{1}{2}$ to optimality. In Section 9, XCSMH also solves to optimality the most difficult environment, `Woods102`, using sufficient register redundancy. Interpretation and implications of the results are presented in Section 10.

2 Perceptual Aliasing and Internal Memory

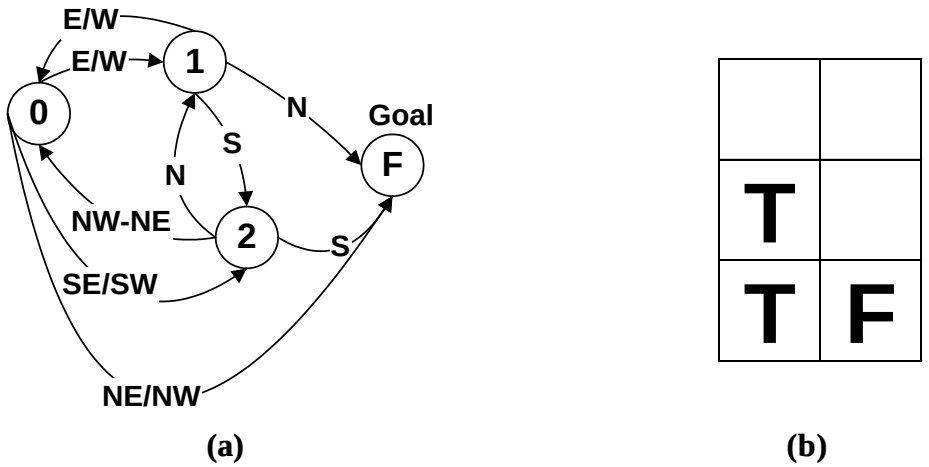
The distinction between Markov and non-Markov problems depends both on the agent's sensory equipment and on the environment. Thus, before we can discuss learning in non-Markov environments we must define what the agent can sense and what the environment looks like.

2.1 Environments and Agents

In the literature two types of environments have been employed to study learning classifier systems: state environments (Riolo 1988; Smith 1994) and "woods" environments (Wilson 1985; Wilson 1994). State environments are described by directed graphs in which nodes represent the agent's sensations (i.e., environmental "states") while edges represent the effect of actions in each state. Edges are usually labelled with the action and sometimes with a transition probability. The agent's task is to learn the shortest path to goal states. An example of a state environment is depicted in Figure 1(a). There are four states (s_0, s_1, s_2, and `F`), the unique goal state is `F`, and there are eight possible actions (`N`, `S`, `E`, `W`, `NE`, `SE`, `NO`, and `SO`). The environment is deterministic since no probability values are associated with edges.

Woods environments are grids in which, typically, each cell can contain an obstacle (a "`T`"), a goal (an "F"), or can be empty. As in state environments, the agent has to learn the shortest path to goal states. The agent senses the environments by means of sixteen boolean sensors (two for each adjacent cell) which tell it the contents of each adjacent cells; the agent can move into any adjacent free cell. Figure 1(b) shows a simple example of a woods environment.

To study learning in non-Markov environments, we will use woods environments because, in our opinion, they are more intuitive and make the problem definition accessible to a larger and more diverse audience. Furthermore, state environments can easily become obscure, even for simple learning problems. In fact, the state environment in Figure 1(a) and the grid in Figure 1(b) represent the same learning problem but the grid representation is surely clearer.



(a) (b)

Figure 1: An example of a state environment (a) and a woods environment (b). The two environments represent the same learning problem.

2.2 Internal Memory

The `Woods101` environment (Cliff and Ross 1994; McCallum 1996), depicted in Figure 2, is non-Markov since it has two distinct positions, indicated by the arrows, which the agent senses as identical but that require different optimal actions. In the right aliased position the optimal action is "*go south-west*"; in the left aliased position the optimal action is "*go south-east*." When the agent is in one of these positions it cannot decide which is the correct action solely considering its current inputs.

Perceptual aliasing in `Woods101` would be easily solved if the agent could remember from which part of the grid it entered the aliased positions. If the agent enters the aliased position from the left corridor, the correct action will be "*go south-east*;" if the agent enters the aliased position from the right corridor, the optimal action will be "*go south-west*." To behave optimally in `Woods101`, given that we cannot enhance the agent's sensory equipment to make `Woods101` Markov, the agent needs some form of memory to cope with the lack of information provided by its sensors.

We can follow two approaches to add memory to the agent. We can explicitly give the agent a "history window" where previous inputs are stored; the agent can then use the history window to take its decisions. From the agent's point of view decisions are taken like: "I sensed $p_1 \ldots p_n$, now I sense p, therefore the best action is a". The history window extends the agent's input vector by including inputs from previous time steps so that the new problem becomes Markov. But the agent's input space grows exponentially in the size of the history window. Accordingly, specific algorithms are often developed in order to reduce that growing complexity and also to identify the minimum amount of past information that is needed to make the problem Markov (McCallum 1996).

As noted, Holland's idea was that the classifier system would evolve message posting and reading strategies that lead to high performance in situations where past information is required, such as non-

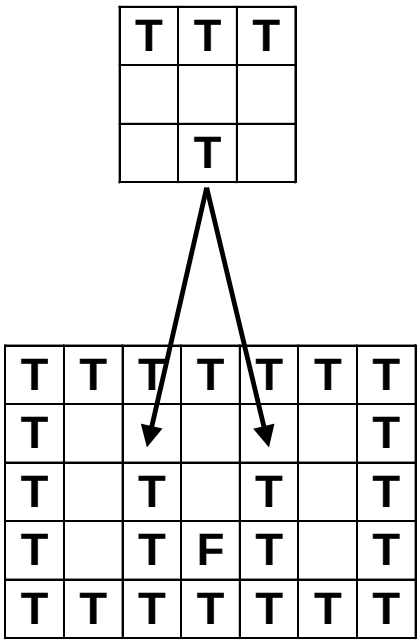


Figure 2: The `Woods101` environment.

Markov environments. Holland's approach is not a history window since past inputs are not explicitly stored. Instead, a Darwinian process is relied upon to evolve appropriate messages and classifiers. Unfortunately, because of the complexity of the message list structure (and also, in our opinion, because classifier fitness was based on predicted payoff, or *strength*), Holland's classifier system has shown only limited success on non-Markov problems (Robertson and Riolo 1988; Smith 1994).

Wilson (1994) suggested that coupling between posting and reading classifiers would be enhanced if the internal state were embodied in a simple bit register instead of a list of messages. The agent is given a register where it can store information—as little as one bit—which might be useful to disambiguate aliased situations. As in the Holland classifier system, past experience is not stored explicitly, as it is in the *history-based* approach, but the agent must, through a Darwinian process, in effect learn to use the memory to solve perceptual aliasing. For example, consider an agent with a *one* bit memory register, R, that must solve `Woods101` optimally; the agent can read the register, and can write in it. A possible optimal policy for `Woods101` using the register R, as depicted in Figure 3, consists of using R to remember from which part of the grid the agent entered the aliased position. If the agent is in the left side of the maze, it sets R to 0; if the agent is in the right side of the maze, it sets R to 1. When entering an aliased position the agent selects the action to perform depending on the value of the register R: if R contains 0 the agent performs the action "*go south-east*;" if R contains 1 the agent performs the action "*go south-west*."

Comparing the history window and memory register approaches it is worth noting that in the former method perceptual aliasing is solved by coupling the learning algorithm (Q-learning in McCal-

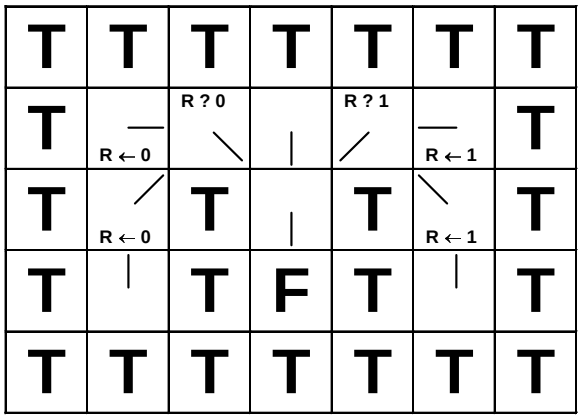


Figure 3: An optimal policy in `Woods101` for an agent with one bit of internal memory. Lines in each position indicate the best action that the policy suggests. The notation "R←..." means that the register is set before the agent moves; "R?..." means that the agent moves only if the register contents are set as specified.

lum (1996)), with specific algorithms responsible for storing history and for identifying the minimum amount of necessary past experiences. In the memory register approach, the *same* learning algorithm (ZCS in Cliff and Ross (1994), XCS in this paper) is intended both to solve perceptual aliasing and learn the optimal paths to the goal state. Perhaps the most intriguing aspect of the memory register—or any classifier system—approach, is that perceptual aliasing is resolved not by storing and/or calculating past information, but by evolving *symbols*, i.e., register settings or messages, that are sufficient to *represent* needed past information without storing it explicitly. As such it may cast light on the broad problem of how symbols can arise in learning systems.

3 Related Work

In the literature many different methods have been proposed to deal with perceptual aliasing. In the simplest case aliasing is simply ignored and the same algorithms, e.g., Q-learning, are applied that would be optimal if the problem was Markov (Chrisman and Littman 1993). Such methods usually perform poorly and moreover Littman (1994) showed that finding the optimal *memoryless* policy for a non-Markov problem is NP-hard. Results can be improved if stochastic policies are considered (Jaakkola, Singh, and Jordan 1995) or if a complete model of the underlying non-Markov problem is available. In this case approximate solutions can be searched (Hansen 1998; Hauskrecht 1998; Meuleau, Peshkin, Kim, and Kaelbling 1999) as well as optimal ones (Kaelbling, Littmann, and Cassandra 1998).

All other methods use some sort of memory of past observations. One way to include memory in a reinforcement learning algorithm like Q-learning is to use a recurrent neural network to represent the Q-values. The resulting network will predict payoff values which implicitly depend on past observations (Lin and Mitchell 1992; Schmidhuber 1991). This approach works well on small problems but

may suffer from local optima in complex problems. Otherwise, as noted previously, the agent can use a finite window of observations to restore the Markov property (McCallum 1995).

In learning classifier systems, past observations are represented by internal messages. Using a very basic non-Markov environment, Smith (1994) carefully analyzed Holland's framework and identified a number of respects in which it can fail on non-Markov environments. Notably, Smith identified a classifier rule and message assignment problem that bears similarities to our "colors" discussion in Section 8.

Wilson's (1994) memory-register proposal was followed by Cliff and Ross (1994) who reported quite good results with Wilson's approach added to ZCS (Wilson 1994), although optimal performance was not obtained. Furthermore, Cliff and Ross (1994) suggested that the approach would not scale up. Optimal performance in non-Markov environments was reported by Lanzi (1998a) when he added the memory register to XCS.

Recently Peshkin, Meuleau, Kim, and Kaelbling (1999) applied the memory-register idea to tabular SARSA(λ), reporting interesting results for simple non-Markov problems.

4 The XCS Classifier System and Internal Memory

Classifiers in XCS have three main parameters: (i) the prediction p, which estimates the payoff that the system expects if the classifier is used; (ii) the prediction error ϵ, which estimates the error of the prediction p; and (iii) the fitness F, which estimates the accuracy of the payoff prediction given by p.

On each time step, the system input is used to build a *match set* [M] containing the classifiers in the population whose condition part matches the current sensory inputs. If the match set does not contain any classifiers, a new classifier which matches the current inputs is created through *covering*. For each possible action a_i in [M], a *system prediction* $P(a_i)$ is computed as the fitness weighted average of the predictions of classifiers which advocate action a_i in [M]. $P(a_i)$ estimates the payoff that the system expects if action a_i is performed. Action selection can be *deterministic*, i.e. the action with the highest system prediction is chosen, or *probabilistic*, i.e. the action is chosen with a certain probability among the possible actions.

Classifiers in [M] which advocate the selected action form the current *action set* [A]. The selected action is then performed in the environment, and a scalar reward r is returned to the system together with a new input configuration.

Classifier parameters are updated on each time-step. In sequential problems such as those considered here, the updates occur in the action set $[A]_{-1}$ from the *previous* time-step, which is saved for the purpose. First, a Q-learning-like payoff P is computed: $P = r_{-1} + \gamma max_a P(a)$, where r_{-1} is the reward on the previous time-step, $P(a)$ are the system predictions for the current time-step, and γ is a *discount factor* ($0 \leq \gamma < 1$). Then, each classifier in $[A]_{-1}$, is updated as follows. The prediction p is updated using the Widrow-Hoff delta rule (Widrow and Hoff 1960) with learning rate β ($0 \leq \beta \leq 1$):

$p \leftarrow p + \beta(P - p)$. The prediction error ϵ is updated with the formula: $\epsilon \leftarrow \epsilon + \beta(|P - p| - \epsilon)$. The fitness update is slightly more complex. Initially, the prediction error is used to calculate the *accuracy* κ of each classifier as $\kappa = 0.1(\epsilon/\epsilon_0)^{-n}$ for $\epsilon > \epsilon_0$, else $\kappa = 1$. Then, each classifier's *relative accuracy* κ' is computed: $\kappa' = \kappa / \sum_{[A]_{-1}} \kappa$. Finally the fitnesses are adjusted: $F \leftarrow F + \beta(\kappa' - F)$.

The genetic algorithm is applied to $[A]_{-1}$, though not usually on every time-step. It selects two classifiers with probability proportional to their fitnesses, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. The resulting offspring are inserted into the population and two classifiers are deleted. See Wilson (1995) and Wilson (1998) for further XCS details.

4.1 Adding Memory to XCS

Adding memory to XCS following Wilson's (1994) proposal, and as implemented by Cliff & Ross (1994) on ZCS, is straightforward. An internal register with b bits is added to the XCS architecture; classifiers are extended with an *internal condition* and an *internal action* that XCS uses respectively to *examine* and to *modify* the contents of the internal register. Internal conditions and internal actions consist of b characters in the ternary alphabet {0,1,#}. For internal conditions, the symbols retain the same meaning they have for the 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; don't care symbols (#) leave the corresponding bits unmodified. There are nine possible external actions: eight moves and one *null* action that the agent can use to remain stationary in the environment while performing only internal actions. The nine actions are encoded using two symbols in the alphabet $\{0, 1, \#\}$. Internal conditions and internal actions can be initialized randomly or by the covering operator.

XCS with internal memory, "XCSM", works basically like XCS. At the beginning of each problem, the internal register is initialized setting all b bits to zero. At each time step, the match set [M], the system predictions, and the action set [A] are formed essentially as in XCS. The differences are that in XCSM internal conditions must also match when building [M], and each combination of an external and an internal action results in a distinct system prediction. The action set [A] is created as in XCS, but all classifiers in it have the same external / internal action combination. The external action and the internal action are performed in parallel: the former is sent to the environment, the latter modifies the internal register. In XCSM the classifier parameters are updated and the genetic algorithm works as in XCS.

In the rest of the paper, we refer to XCS with an internal memory of b bits as XCSMb, to XCSM when the discussion is independent of the size b of the register, and finally to XCS(M) when considering both XCS and XCSM models. Later we shall introduce a further modification called XCSMH.

4.2 Design of Experiments

Each experiment consisted of a number of problems that the agent must solve. For each problem the agent is randomly placed in an empty cell of the environment; then the agent moves under the control of the classifier system until it reaches the goal, receives a constant reward, and the problem ends.

Each problem is either a *learning* problem or a *test* problem. In a learning problem, the agent selects actions (external/internal action combinations) randomly [1] from those having system predictions (i.e., those represented in the match set). During a learning problem, the system is said to solve the problem in *exploration* (Wilson 1995). In a test problem, actions are selected deterministically: the external/internal action combination with the highest prediction is selected. This is also termed solving the problem in *exploitation*. The genetic algorithm is in operation during learning problems, but it does *not* operate during test problems. At the beginning of a new problem, the agent decides with probability 0.5 whether it will solve a learning problem or a test problem. Thus learning and test problems approximately alternate.

The agent's performance is computed as the average number of steps to the goal position in the past 50 test problems. After some number of problems, learning is usually turned off and all further problems are solved as test problems. The period before learning is turned off is termed the *learning period*; the period afterward is called the *testing period*. Use of test problems during the learning period allows monitoring of learning progress. Test problems afterward evaluate the final attained performance. All statistics in this paper are averaged over ten experiments.

5 Experiments in Simple Non-Markov Environments

In the first experiment, we applied XCSM with one bit of memory, XCSM1, to `Woods101` (Figure 2) which was employed by Cliff and Ross (1994) to test the extension of ZCS with internal memory. XCSM1 used a population size, N, of 800 classifiers. Parameters were set as follows: β=0.2, γ=0.71, θ= 25, ϵ_0=0.01, χ=0.8, μ=0.01, and ϕ=0.5, following settings in Wilson (1995). During the last 2000 problems, learning was turned off to test the final solution evolved. Recall that during the learning period the genetic algorithm is in operation and actions are selected randomly; during testing, the genetic algorithm *is not* in operation and the best actions are always selected.

The results depicted in Figure 4 show that XCSM1 converged to an optimal policy during the testing period. During learning (first 6500 problems) the performance oscillated slightly above the optimum. Examination of classifier populations during the learning period revealed that an optimal policy was often present. But, in non-Markov environments, as further explained in Section 6.1, exploration of internal actions can cause an optimal policy to be lost, at least temporarily. Once exploration stopped in Woods101, the solution immediately converged to the optimum.

[1] Later, using compound exploration, the selection will be only partly random.

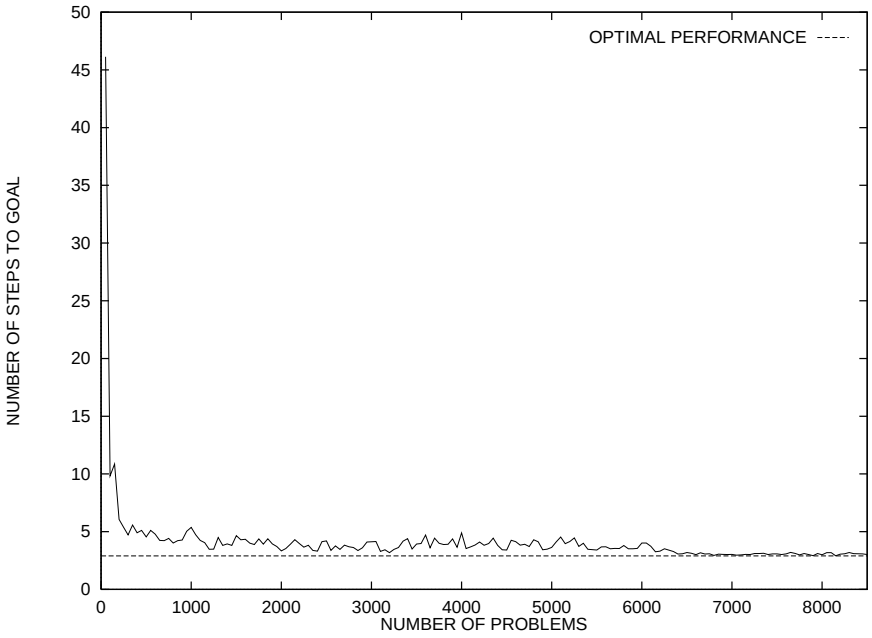


Figure 4: Performance of XCSM1 in `Woods101`. Population size is set to 800 classifiers. The optimum average performance is at 2.90 steps per problem.

`Woods101` is a simple problem in that the optimal solution requires the agent to visit at most one aliased position before reaching a nearby goal state. We introduced `Maze7` (Lanzi 1998a) to test XCSM in a more challenging environment where the agent had to visit more aliased positions and perform longer sequences of actions before it reached the goal. `Maze7`, depicted in Figure 5, is a path of nine cells with a goal at the end; it has two aliased positions, indicated by the dashed circles. We applied XCSM1 to `Maze7` using a population of 1600 classifiers; parameters were set as in the previous experiment. Results are reported in Figure 6; as in the previous experiment we turned off learning during the last 2000 problems to test the final solutions evolved.

Figure 6 shows that during learning the agent did not converge to an optimal solution. But when the final population was tested, indicated by the arrow at beginning of the peak, XCSM1 eventually evolved a near optimal solution for `Maze7`. Examination of populations showed that during learning the agent was unable to evolve an optimal policy that could reach the goal from positions near the end of the corridor. Therefore, XCSM performance would drop whenever a problem began in one of those positions, so that the overall performance oscillated heavily. When learning stopped (at the beginning of the peak in Figure 6) the performance fell sharply indicating that the final policy caused the agent to get stuck (loop) in some areas of the environment. XCSM detects this situation because the predictions of the classifiers involved dramatically decrease (Wilson 1995), and it starts replacing them through covering.[2] Thus the final solution was formed almost completely of classifiers created by the covering operator. Quite remarkably, it was close to optimal.

[2]The covering operator was enabled during both learning and testing.

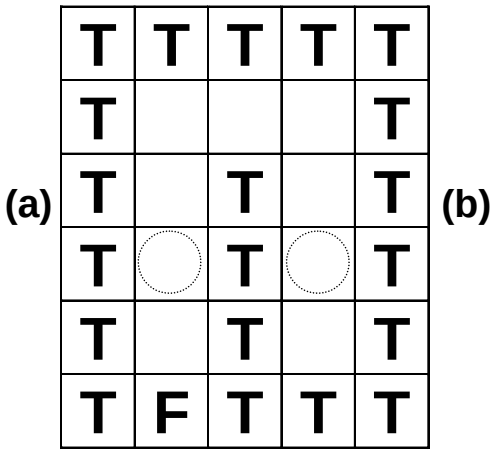


Figure 5: The `Maze7` environment. The two aliased positions, (a) on the left and (b) on the right, are indicated by dashed circles.

This analysis suggests that XCSM1 can evolve a near optimal solution for `Maze7` just using covering. We verified this by applying XCSM1 in `Maze7` in an experiment with no genetic algorithm and completely deterministic action selection. Note that under this regime XCSM1 implements a *greedy* search of the optimal policy for `Maze7`. In fact, the system performs only a little search through covering, while the system chooses the most promising search direction by always selecting the highest predicting action.

Figure 7 reports the average performance of this greedy XCSM1 with a solid line. The two dashed lines represent respectively the best and the worst performances over the ten runs. The analysis of single runs showed that in many cases greedy XCSM converged very close to the optimum (lower dashed line in Figure 7) but sometimes evolved an unstable solution (upper dashed line in Figure 7).

These results may suggest that a merely greedy version of XCSM would be capable of solving general non-Markov problems. But a series of experiments (not reported) showed that greedy XCSM is likely to converge to a highly suboptimal solution as the complexity of the environment increases. Finally, since generalization in XCS(M) is achieved through exploration and use of the genetic algorithm, this version of XCSM effectively throws away all generalization capability.

6 Analysis of the Results

6.1 Explaining XCSM Behavior in `Maze7`

Returning to "regular" XCSM, consider the two aliased positions in `Maze7`, indicated in Figure 5 by the dashed circles. An optimal policy which disambiguated the two aliasing positions (a) and (b), might include two classifiers which advocate respectively 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). Suppose that

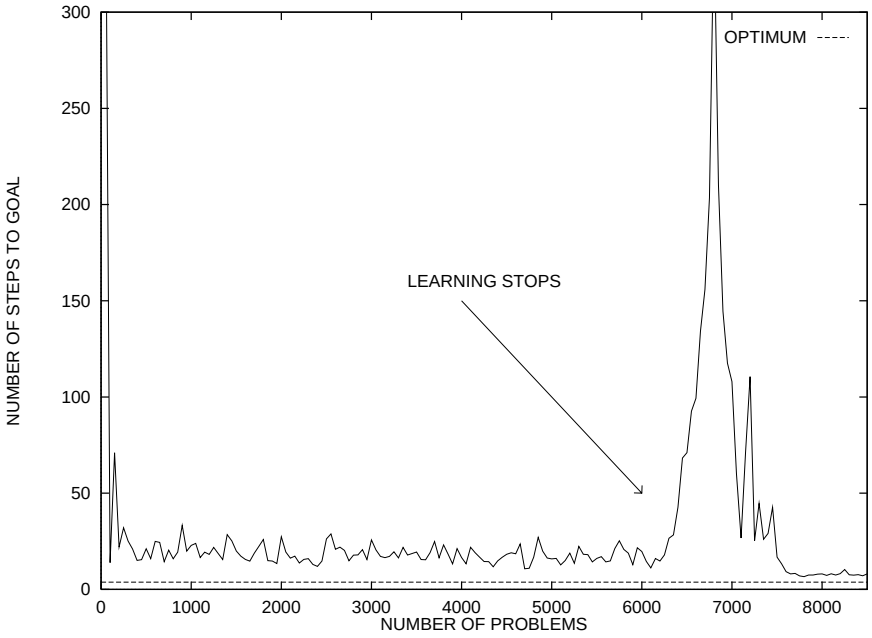


Figure 6: Performance of XCSM1 in `Maze7`. The curve is averaged over ten experiments. The optimum is 4.33.

XCSM has successfully evolved such a policy and that, at this point, the system starts exploration. Since during exploration the system selects actions randomly, it may happen that the agent enters position (b) with the internal register set to `1`. The agent can now select any of the actions which appear in the match set and, for example, it may try the action "*go-south*". If this occurs, the classifiers that match position (b) are the same ones which predict the optimal action in position (a). But in these two positions, the action "*go-south*" has different payoffs. Therefore, classifiers that are optimal in (a) become inaccurate when tried in (b) because they are applied in *the same situation* with different payoff levels. Accordingly, since in XCSM classifier fitness is based on accuracy, their fitnesses decrease. As a consequence, they reproduce less and may be selected for deletion through the evolutionary process.

6.2 Verification of the Hypothesis

To verify whether our explanation of XCSM's suboptimal behavior in `Maze7` is correct we applied a version of XCSM in which the action with the highest payoff was *always* selected (both during learning and during testing), while the genetic algorithm was in operation, as usual, during learning. This experimental regime was identical to that of Figure 6, except that action-selection was deterministic during learning. Our aim was to verify that the suboptimal behavior we observed earlier depends on the action selection strategy—namely on exploration of internal actions—and not on the genetic algorithm.

We applied this version of XCSM in `Maze7` with the same settings we used in the previous experiments. The results reported in Figure 8 show that with these settings XCSM1 converged almost

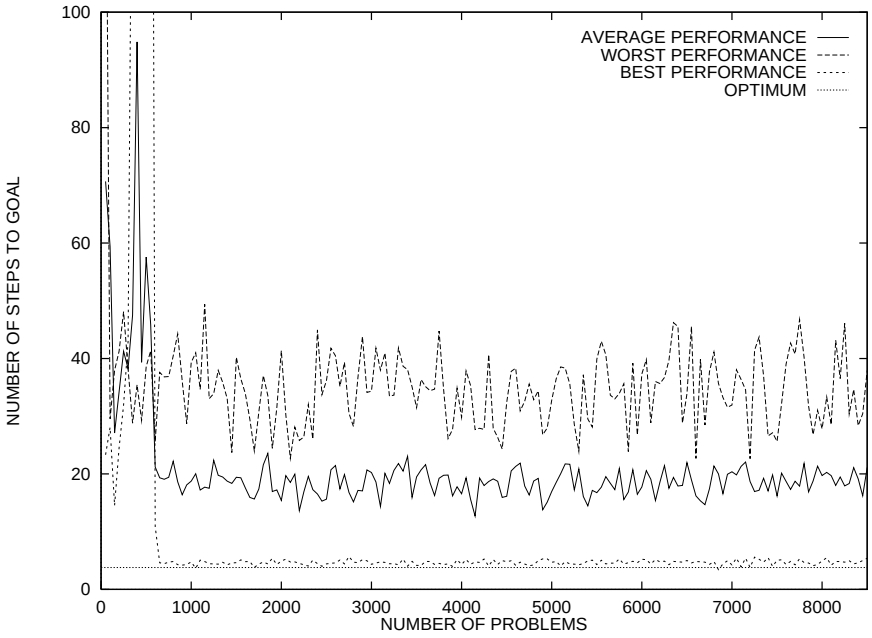


Figure 7: Performance of XCSM1 in `Maze7` when the system works only in exploitation: the genetic algorithm is not operating and the best action is always selected. The solid curve represents the average performance over ten runs. Lower dashed line and upper dashed line are respectively an example of stable near optimal performance and of unstable performance.

immediately to an essentially stable optimal solution, supporting our explanation that in non-Markov environments, exploration of internal actions can interfere with learning. Separately, we note that the regime in this experiment differed from that of the experiment with greedy XCSM *only* in that the GA was active here. Yet the optimum was reliably reached in this experiment whereas it was not reached with greedy XCSM. This suggests that, even without exploration of actions, the search performed by the genetic algorithm itself is sufficient for convergence to an optimal solution in `Maze7`.

Our analysis appears to be general, thus it should apply to all the environments. On the other hand, in Section 5 we showed that XCSM1 solves `Woods101` optimally so it might appear that our explanation is contradicted by those positive results. However, if we analyze the classifiers in the population during single runs for XCSM1 in `Woods101` we note that also in `Woods101`, classifiers that are optimal in one of the two aliased positions eventually become inaccurate because they are tried in both the aliased positions. But, since in `Woods101` these positions are near to the goal state, they are visited frequently. Moreover, since these positions are symmetric with respect to the goal, their payoff levels in both positions are similar. Thus, the classifiers which advocate the optimal actions in the two aliased positions are able to survive, even if they may be applied to situations that have different payoff levels. It is true that in XCS(M) inaccurate classifiers are deleted through evolution (i.e., they tend to reproduce less and consequently to be deleted). But this evolutionary mechanism is slow so that in `Woods101` its effects are counterbalanced by the frequent exploitation of those classifiers which form

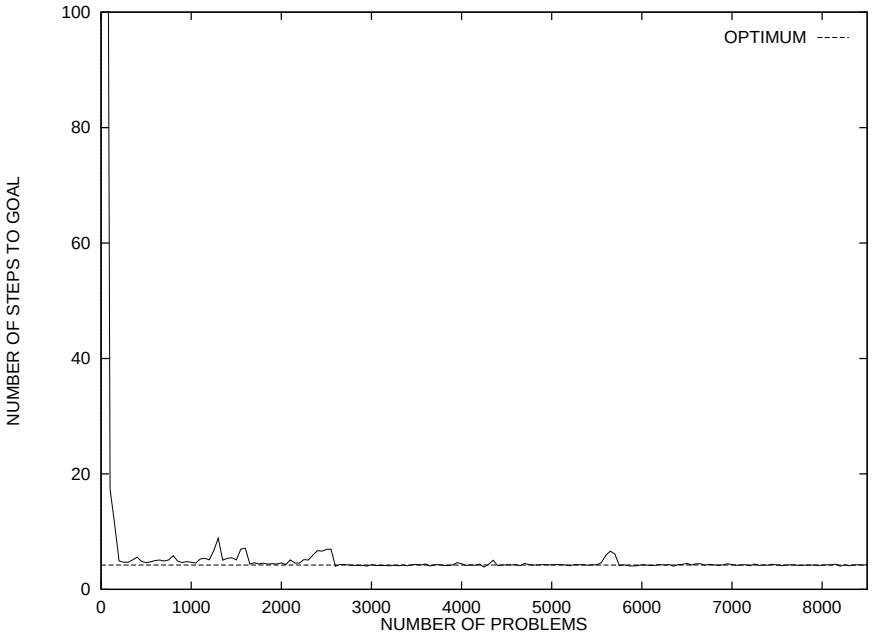


Figure 8: Performance of XCSM1 in `Maze7` with deterministic action selection. The genetic algorithm is operating during learning, but the best action is selected both in learning and in testing.

the optimal policy. In contrast, in `Maze7`:

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

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

The reasons for the difference in XCSM performance between `Maze7` and `Woods101` are now clear. Because of (i), in `Maze7` the classifiers that are optimal in a certain aliased position are more likely to become inaccurate when they are tried in the other aliased position. Because of (ii), those classifiers that are optimal in (a) tend to survive because they are tried frequently in (a) and seldom in (b). At the same time, classifiers that are optimal in (b) tend to be selected for deletion because they are often tried 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 associate the content of the internal memory to the actual agent position in the environment, so as to solve aliased situations. We showed that in XCSM, since during exploration actions are selected randomly, the agent may enter the same position with different memory settings. When this happens, the memory mechanism becomes useless with respect to the perceptual aliasing

problem, because the contents of internal memory are no longer related with the absolute agent position.

The analysis we presented reveals that the strategy XCSM follows to use the internal memory register 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 increase XCSM's learning capabilities.

7.1 Description of XCSMH

We now introduce an extension to XCSM, "XCSMH", which appears capable of learning an optimal policy in much more difficult partially observable environments. XCSMH retains the major characteristics of XCSM, while it differs from it in two major respects:

- In XCSMH internal actions are performed only if the associated external action results in a change in the sensory inputs; that is, if the agent perceptions have changed.
- During learning, XCSMH employs a *compound* action selection strategy: the system first selects the internal action deterministically; then, it selects the external action randomly as usual.

These changes are closely related to the analysis of XCSM behavior we presented in the previous sections. In the following, we discuss each change in detail.

Internal Actions. In the previous section we observed that, during exploration, the main problem of XCSM is that internal actions act independently of the agent position in the environment. In particular, the internal memory can be modified even if the agent does not change its position. In contrast, XCSMH associates the execution of an internal action with a definite change in the agent perceptions. I.e., an internal action is performed only if the corresponding external action has caused the agent to receive a new sensory input. Thus in XCSMH internal memory is associated with the sequence of past agent sensations. Internal memory does not represent the result of the execution of an *internal program* anymore (Cliff and Ross 1994) because the register cannot be updated independently from what the agent experiences in the environment. Rather, in XCSMH, internal memory is intended to give a compact representation of the agent's past sensory experience. As an immediate consequence of this memory update policy, XCSMH no longer employs *null* external actions.

Compound Action Selection. The second change we introduce in XCSMH concerns the agent's action selection strategy during learning. As we discussed previously, random action selection may cause the agent to enter the same aliased position with different configurations of internal memory. Consequently, XCSM can fail to associate the current content of the internal memory to the actual position in the environment. In XCSMH, we replace the usual random action selection strategy by a *compound*

action selection strategy to limit the set of internal actions that the agent can perform in each position. As a result, XCSMH tends to associate a specific internal memory configuration with each position in the environment. The strategy we propose works as follows:

- First, the agent selects the internal action which corresponds to the internal/external action pair predicting the highest payoff, i.e., to the highest system prediction.
- Then, the agent selects the external action randomly from among the action pairs having the selected internal action.

This compound action selection strategy selects the internal action deterministically so that, in each environmental niche, the classifiers that advocate the best internal action will tend to survive; the external action is selected randomly as usual to guarantee an adequate exploration of the environment.

7.2 XCSMH vs. XCSM

We compared XCSMH and XCSM in `Maze7` using the same parameter settings used in the previous experiments. Results reported in Figure 9 show that compound action selection strategy does improve XCSM performance. In fact XCSMH (solid line) performs significantly better that XCSM1 (dashed line), and rapidly converges to an optimal policy for `Maze7`.[3] Furthermore, if we compare XCSMH1 and XCSM1 in `Woods101` (not reported) we find that the performance of the two systems is almost identical (Lanzi 1998b).

7.3 A Linguistic Interpretation of the Results

To solve a non-Markov problem, XCSM has to develop a strategy to associate the content of internal memory to the agent's actual position in the environment. This strategy is formed by (i) classifiers which properly set the content of internal memory before the system enters an aliased position; and (ii) classifiers which exploit those settings to behave optimally *in* the aliased situations.

These classifiers can be viewed as agents trying to develop a common lexicon (Steels 1996). There are agents, the classifiers in (i), that try to develop a "language" whose symbols are used to *mark* aliased situations. There are agents, the classifiers in (ii), which try to associate a "meaning" to those symbols, i.e., what actual position in the environment corresponds to a specific internal memory configuration. To use the internal memory effectively implies that XCSM must both create an appropriate internal language of feasible memory configurations, and learn to associate a meaning to that language by binding each memory configuration to a specific aliased position.

In this perspective the experiments with `Maze7` suggest that for XCSM it is difficult to develop both the language and the meaning at once. In fact our analysis shows that incoherence can arise in the

[3] The vertical scale of Figure 9 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 performance goes off-scale.

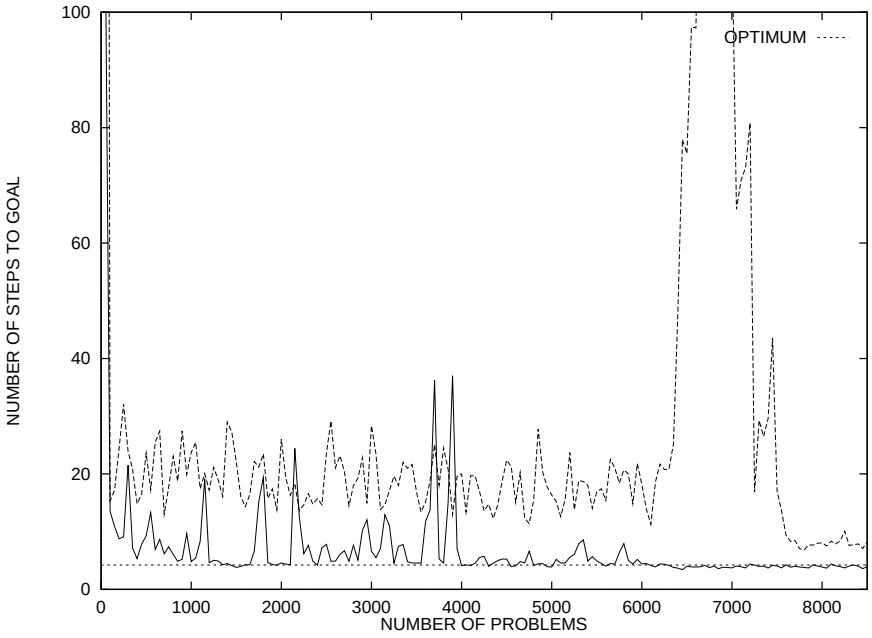


Figure 9: Comparison of the new XCSMH1 (solid line) and XCSM1 (dashed line) in `Maze7`.

meaning of a specific language symbol because the same symbol (i.e., memory configuration) might be associated with different meanings (i.e., aliased positions). With the compound action selection strategy in XCSMH we try to fix the internal language by limiting its exploration and letting the system focus on the evolution of a correct meaning. In fact, in XCSMH the internal language is searched by the genetic algorithm alone[4] while the meaning is searched as usual both through random exploration and through evolution.

8 More Than Two Aliased Positions

Both `Woods101` and `Maze7` have only two aliased positions; accordingly, they can be solved by an agent with one bit of internal memory. In this section we extend those results by applying XCSMH in an environment, `Woods101`$\frac{1}{2}$, for which an optimal policy requires more than one bit of memory. `Woods101`$\frac{1}{2}$, depicted in Figure 10(a), has four different positions that the agent perceives as identical, as shown in Figure 10(b), but which require four distinct optimal actions. To disambiguate these positions the agent needs at least two memory bits, to represent four distinct memory configurations. A possible optimal solution for `Woods101`$\frac{1}{2}$ using a memory register R having two bits is depicted in Figure 11. As with `Woods101` (Figure 3) the optimal policy for `Woods101`$\frac{1}{2}$ consists of properly setting register R before entering an aliased position. Then, when the agent enters an aliased position, it selects the best action according to the content of R.

We applied XCSMH with two bits of memory, XCSMH2, in `Woods101`$\frac{1}{2}$ in order to test whether

[4]Because internal actions are always selected deterministically.

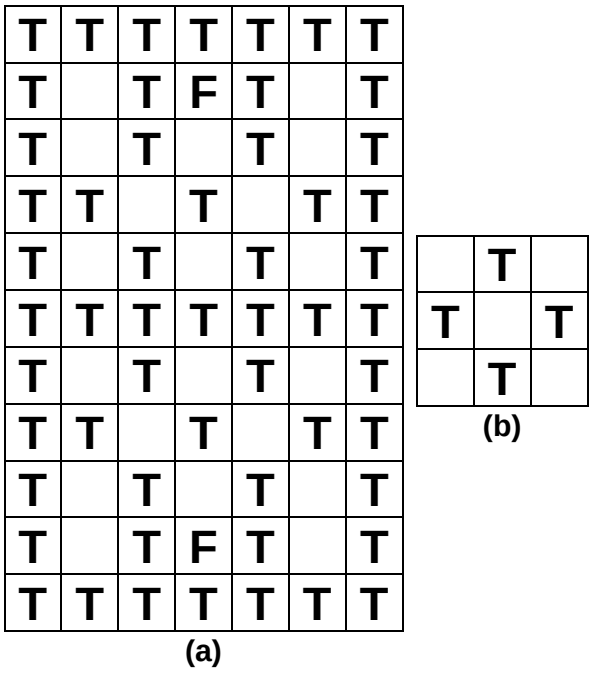


Figure 10: The Woods101$\frac{1}{2}$ environment.

XCSMH can exploit internal memory to disambiguate the four aliased positions. The population size was set to 2800 classifiers. General parameters were set as in the initial experiments with XCSM1; during the last 2000 problems we turned off learning to test the final policy.

The results reported in Figure 12 (upper solid curve) show that in Woods101$\frac{1}{2}$, XCSMH2 converges to a policy that is close to, but not optimal. Analysis of individual runs indicated that the system could not evolve a policy capable of solving all four aliased situations at once. The final policies would usually disambiguate at most three out of the four aliased positions. These results suggested that the approach based on the internal memory register might not be scalable to more complex problems. However, let us consider how the agent exploits the internal memory to cope with perceptual aliasing.

To disambiguate the four aliased positions in Woods101$\frac{1}{2}$ the agent must evolve a policy in which before entering one of those positions the internal register is properly *stamped* so that when the agent faces an aliased situation it can in effect determine its actual position by looking at the register. The agent solves perceptual aliasing by taking decision like: "Register R is set to 01 therefore I should be in the aliased position at *south-east*, and the best action is *go south-west*" (see Figure 11). From this perspective, the different configurations of internal memory can be seen as *colors* that the agent can use to *mark* the different aliased positions. When the environment can be solved with one bit of internal memory, just two colors are needed. When the environment requires two bits of internal memory, four colors are needed. Note that there is a direct relation between the number of colors that an environment requires and the number of distinct optimal solutions: the more the colors, the more possible assignments of

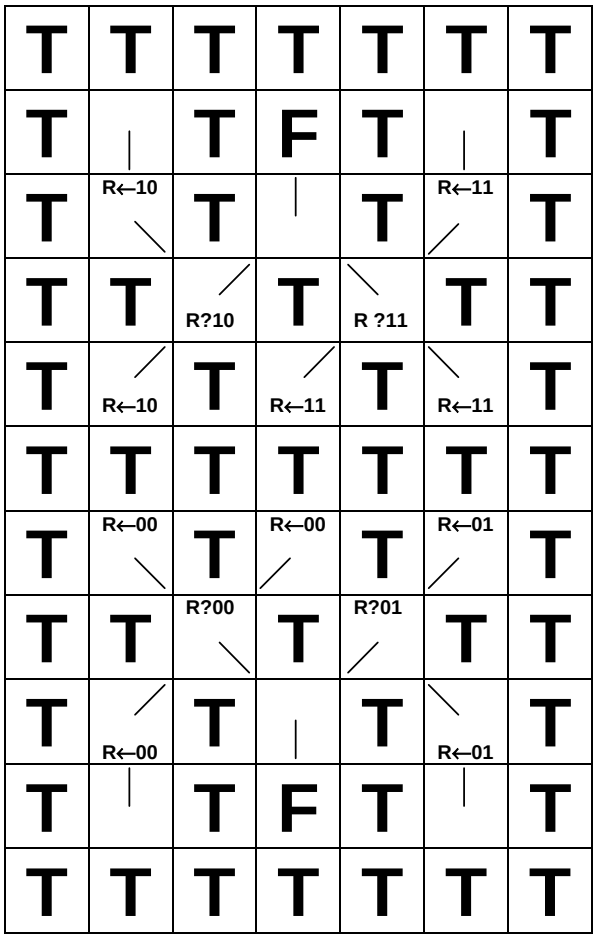


Figure 11: An optimal policy for Woods101$\frac{1}{2}$ with an internal register, R, of two bits. See Figure 3 for notation.

colors to aliased states. In particular, in Woods101 and in Maze7 there are only two strategies that can be used to disambiguate the two aliased situations, while in Woods101$\frac{1}{2}$ there are $4!$ different strategies. This may be considered a positive aspect of Woods101$\frac{1}{2}$ since more admissible solutions means that the system has more chances to find an optimum. Still, Woods101$\frac{1}{2}$ is harder than Woods101 to begin with.

To solve Woods101$\frac{1}{2}$, XCSMH must evolve a policy which associates a distinct color (i.e., memory configuration) to each aliased position. Note that the agent associates a color to a particular aliased position *without* knowing which colors have been already associated to the other positions. The agent takes a "local" decision and lets evolution "decide" which policy is best. Accordingly, when there are many different aliased position that must be solved at once, as in Woods101$\frac{1}{2}$, it may be difficult for the system to evolve a globally optimal policy. Let us illustrate this problem with an example.

Consider Woods101$\frac{1}{2}$ and suppose that XCSMH2 has evolved a policy to disambiguate three of the four aliased situations. When the agent enters the last aliased position it should build a proper association between the memory and the position so that the overall policy becomes globally optimal. Note that the system has three possibilities out of four of associating the current position with a color that has been already used. Thus XCSMH2 is most likely to choose a color (i.e., a memory configuration) which is already associated with another position. When this happens those classifiers that match the two positions associated to the same color will become inaccurate because they apply in aliased situations with different payoff. Consequently, they are likely to be deleted; that is, part of the solution evolved is corrupted and the system "forgets" what it has learned. There is, however, a certain probability that in entering the last aliasing situation the system will associate the correct memory configuration with that position. Accordingly, we should see from time to time that an optimal policy is evolved even if only temporarily. But this was not observed in our experiments. As a possible explanation, we note that in solving an aliased situation the system must associate the correct color with *all* the positions that are adjacent to the aliased situation to guarantee a complete solution. So in Woods101$\frac{1}{2}$ all the classifiers matching in three different positions around an aliased cell must associate their internal action with the correct memory configuration. Therefore, although there is a finite probability that the system will make the correct association once, the chances that the same association is selected for all the three adjacent positions is quite small.

At this point we may think that a solution for this type of problem will be hard to find. Nevertheless, if our analysis is correct, the reason XCSMH2 does not evolve an optimal policy in Woods101$\frac{1}{2}$ is simply that it does not have *enough* colors to differentiate the four aliased positions. More precisely, it is difficult to solve Woods101$\frac{1}{2}$ having only the exact number of colors that are sufficient to distinguish the different aliased positions, but a larger number of color "choices" might make the problem easier.

In a second experiment we gave more "colors" to the agent to test whether that would lead to an optimal policy for Woods101$\frac{1}{2}$. We applied XCSMH with four bits of internal memory, XCSMH4, and a

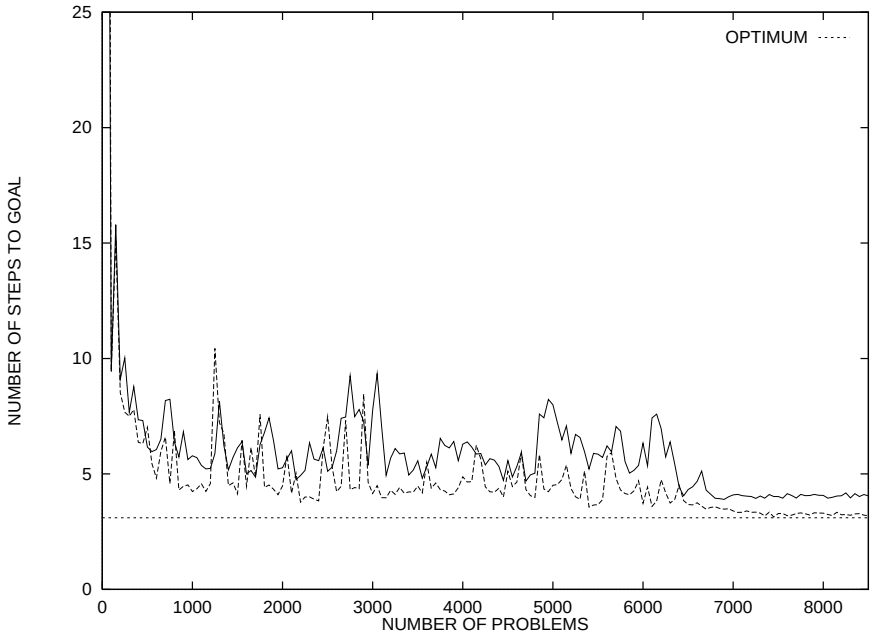


Figure 12: Comparison of the performance of XCSMH2, solid line, with that of XCSMH4, dashed line, in `Woods101`$\frac{1}{2}$. Population size is set to 2800 classifiers. Curves are averaged over ten experiments. The optimum is at 3.10.

population of 2800 classifiers in `Woods101`$\frac{1}{2}$; during the final 2000 problems exploration was turned off to test the final solutions. Our hypothesis was that with more bits the system would have more chances to associate distinct memory configurations with distinct aliased positions. Note that an increase in the number of memory bits brings an exponential increase of the search space; accordingly, the system needs more classifiers to converge. But while the search space complexity grows exponentially, we found that the required population size grows almost linearly. This supports Wilson's (1998) hypothesis which suggests that in XCS, the complexity of the learning process grows as a low polynomial of the complexity of the learning task instead of with the complexity of the search space as in other learning techniques such as neural networks and nearest-neighbor.

Figure 12 compares the performance of XCSMH2 with 2800 classifiers (solid line) with that of XCSMH4 and the same population size (dashed line) in `Woods101`$\frac{1}{2}$. XCSMH4 with four bits of internal memory performed better than XCSMH2, supporting our hypothesis. All the ten XCSMH4 runs in fact converged to optimal policies. These results tend to confirm the "color" analysis. They also recall work of Teller (1994) who used an internal memory register in a genetic programming approach to a robot task and found that adding redundant positions to the register improved the results.

9 A More Difficult Problem

We now present a final set of experiments using a more difficult environment. `Woods102`, depicted in Figure 13 was first introduced by Cliff and Ross (1994). `Woods102` has two aliased situations. One,

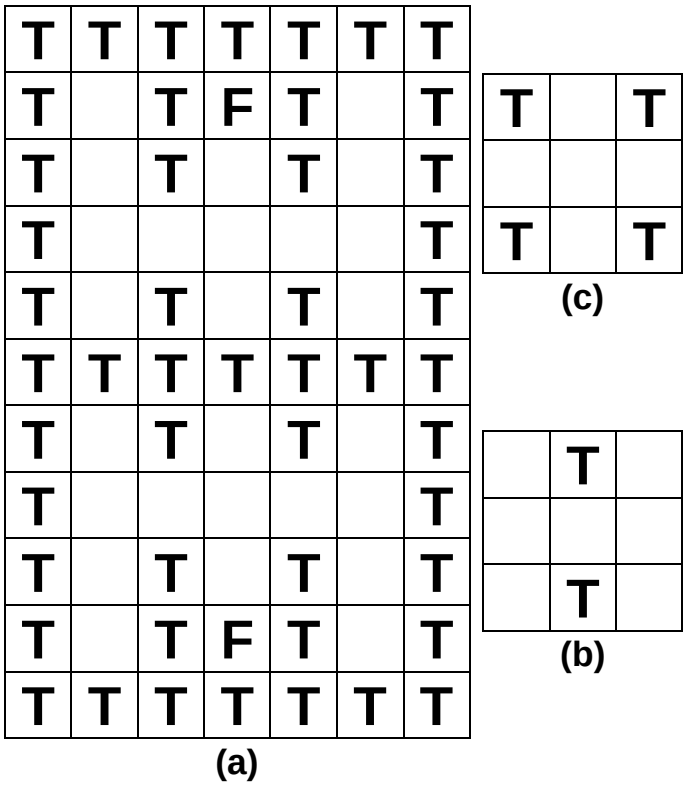


Figure 13: The `Woods102` environment.

shown in Figure 13(b), is encountered in four positions of the environment. The second, shown in Figure 13(c), is encountered in two other positions. From this it would appear that three memory-register bits are required to resolve perceptual aliasing. However, since the two situations occur in separate parts of the environment, there is the possibility that an optimal policy could evolve in which certain register bits are used for more than one situation, thus requiring fewer bits in all. It is therefore not clear how large a bit-register is strictly necessary.

We therefore performed experiments using XCSMH with 2-, 4-, and 8-bit memory registers. The optimum performance in `Woods102` is 3.23 steps. Performance for XCSMH2 and XCSMH4 reached 4 and 3.7 steps, respectively (results not shown). Results for XCSMH8 are shown in Figure 14, where it is seen that XCSMH8 reliably converged to an optimal policy. Since it seems clear that the number of register bits strictly required to solve `Woods102` is definitely less than eight, these results further confirm the importance of register redundancy.

10 Summary and Implications

The work reported in this paper contains the first examples of a learning classifier system achieving optimal performance in non-Markov environments. Schematically, the final system, XCSMH, consisted

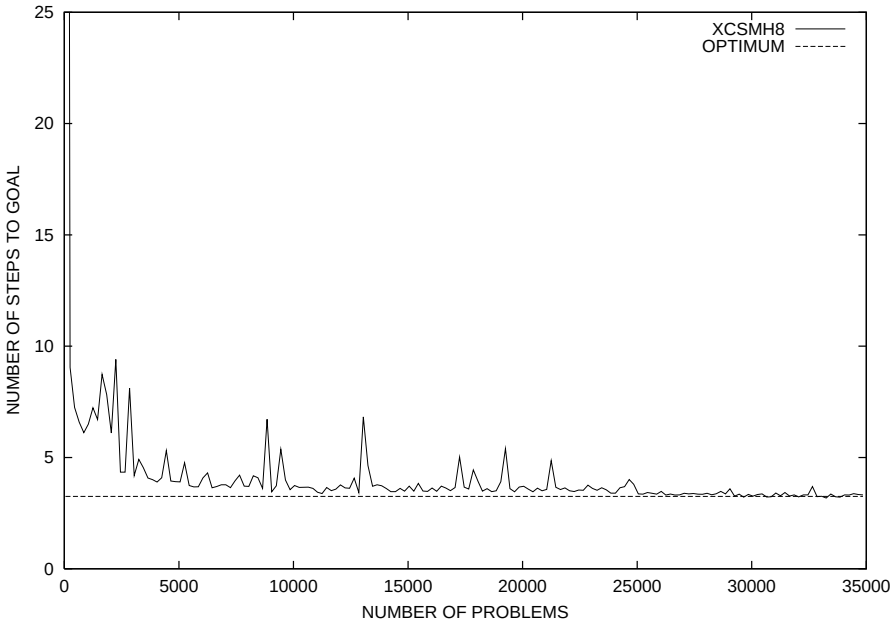


Figure 14: Performance of XCSMH8 in `Woods102`. Population size is set to 6000 classifiers. Curve is averaged over ten experiments. The optimum is at 3.23

of three components: (1) the XCS classifier system; (2) a memory register written into and read by classifier ”internal actions”; and (3) a ”compound” action-selection strategy in which internal actions were selected deterministically. The fact that XCSM—which did not contain component (3)—reached optimality on `Woods101`, whereas ZCSM of Cliff and Ross (1994) did not, suggests that use of XCS was required for optimality. The essential difference between XCS and ZCS lies in the definition of classifier fitness: in XCS it is based on prediction accuracy; in ZCS and traditional classifier systems, on strength, which resembles the prediction itself. Thus our results suggest that XCS's way of defining fitness may be necessary for optimal classifier system performance in non-Markov environments.

XCSMH (and XCSM) are internal-state classifier systems but they use a memory register instead of a message list. The memory register is much simpler conceptually and in implementation than a message list. It is possible that the memory-register architecture can be applied successfully to much more complex non-Markov problems than those here, and may be superior generally to the list. It is also possible that if the fitness definition of message-list classifier systems were changed to that of XCS, optimal performance would be obtained. Both directions should be pursued.

Success of XCSMH on the harder non-Markov problems in this paper resulted from discoveries and observations that increased our understanding of the nature of internal actions. Unlike external actions, which have well-defined effects, the effect of an internal action depends on the classifiers that match the resulting register setting. This has positive and negative consequences. On the positive side, more than one—and possibly a great many–coherent *sets* of classifiers setting and reading the register will be consistent with optimal performance in a given environment, which can ease the search problem. On

the negative side, the requirement for coherence means that exploration of internal actions in a manner typical of external actions can readily upset good sets of classifiers before they are complete.

XCSMH's compound action-selection strategy abandoned probabilistic internal action selection in favor of deterministic selection, in effect leaving the exploration of internal actions to the genetic algorithm. This appeared to provide sufficient stability to the "symbol choosing" side of the internal "language" so that the meanings of the symbols could be evaluated—via external action selection—without disruption. With that accomplished, the positive consequence above—the multiplicity of consistent "languages"—came to the fore, especially when the memory-register size was larger than strictly necessary.

Scale-up is an important issue for learning systems. The question is: how rapidly does the learning time or system size grow as the problem size grows? Cliff and Ross (1994) worried that systems using a memory register would not scale up well, because the amount of needed exploration of internal actions would grow exponentially with the register size. They assumed that the space of internal actions would need to be explored as thoroughly as the space of external actions. Our results indicate otherwise. XCSMH reached optimality using deterministic internal action-selection and GA search; in effect, it was not necessary to explore all possible settings of the register. Many (sets of) settings will be interpretable to yield high or optimal performance; it is only necessary to find one of them.

Nevertheless, we did observe that system size and learning time grew with problem difficulty. To give a crude overall picture, we can compare the experiment with XCSM1 and `Woods101` (Figure 4) to the experiment with XCSMH8 and `Woods102` (Figure 14). The learning time to reach near-optimal performance goes from about 500 problems to 5000 problems, the population size goes from 800 to 6000, and the number of possible internal actions goes from $3^1 = 3$ to $3^8 = 6561$. Clearly, the time and resources required are not rising as fast as the space of possible internal actions, as discussed above, but they do rise considerably.

We are not able at this point to be more precise about scale-up. However, note that an environment like `Woods102` is "tricky" in that aliased positions are numerous and close together. Natural, or real-world synthetic environments may be considerably larger, but the density of aliased situations will probably be considerably lower. In addition, spaced aliased situations may permit the system to evolve schemes for re-using register bits. As a result the degree of learning complexity *added* by aliasing to that already present in the Markov parts of the problem may not be major.

Future work will include implementation in a robotic environment with aliasing due to sensor limitations. However, much remains to be done to understand more deeply the mechanisms of XCSMH and the specific policies it evolves to overcome aliasing, and to obtain definitive results on complexity.

References

Chrisman, L. and M. Littman (1993). Hidden state and short-term memory. Presentation at Reinforcement Learning Workshop, Machine Learning Conference.

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

Hansen, E. (1998). *Solving POMDPs by searching in policy space*. Ph. D. thesis, Department of Computer Science – University of Massachusetts at Amherst.

Hauskrecht, M. (1998). *Planning and Control in Stochastic Domains with Imperfect Information*. Ph. D. thesis, Massachusetts Institute of Technology, Cambridge MA.

Jaakkola, T., S. P. Singh, and M. I. Jordan (1995). Reinforcement learning algorithm for partially observable markov decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems 7*, pp. 345–352. The MIT Press.

Kaelbling, L. P., M. L. Littmann, and A. R. Cassandra (1998). Planning and acting in partially observable stochastic domains. *Artificial Intellingence 101*(1-2), 99–134.

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

Lanzi, P. L. (1998b). An Analysis of the Memory Mechanism of XCS. In J. K. et al (Ed.), *Proceedings of the Third Annual Genetic Programming Conference*, Madison (WI), pp. 643–651. Morgan Kaufmann San Francisco (CA).

Lin, L. (1993). Reinforcement learning for robots using neural networks. Technical Report CMU-CS-93-103, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

Lin, L. and T. Mitchell (1992). Memory approaches to reinforcement learning in non-Markovian domains. Technical Report CMU-CS-92-138, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA.

Littman, M. L. (1994). Memoryless policies: Theoretical limitations and practical results. In D. Cliff, P. Husband, J.-A. Meyer, and S. Wilson (Eds.), *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pp. 238–245. The MIT Press – Cambridge, MA.

McCallum, A. K. (1995, December). *Reinforcement Learning with Selective Perception and Hidden State*. Ph. D. thesis, University of Rochester, Rochester, NY.

McCallum, R. A. (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) 26*(3).

Meuleau, N., L. Peshkin, K. Kim, and L. Kaelbling (1999). Learning finite-state controllers for partially observable environments. In *Fifteenth Conference on Uncertainity in Artificial Intelligence*. AAAI. (to appear).

Peshkin, L., N. Meuleau, K. Kim, and L. Kaelbling (1999). Learning policies with external memory. In *Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann. (to appear).

Riolo, R. L. (1988). Cfs-c / fsw1: An implementation of the cfs-c classifier system in a domain that involves learning to control a markov decision process. Technical report, Logic of Computer Group, Division of Computer Science and Engineering, University of Michigan, Ann Arbor. Avaiable from the technical report archive at ftp: / / ftp.cs.bham.ac.uk / pub / tech-reports / 1996 / CSRP-96-17.ps.gz.

Robertson, G. G. and R. L. Riolo (1988, October). A tale of two classifier systems. *Machine Learning 3*(2 / 3), 139–159.

Schmidhuber, J. (1991). Reinforcement Learning in Markovian and non-Markovian environments. In R. P. Lippman, J. E. Moody, and D. S. Touretzky (Eds.), *Advances in Neural Information Processing Systems 3*, pp. 500–506. The MIT Press.

Smith, R. E. (1994). Memory exploitation in learning classifier systems. *Evolutionary Computation 2*(3), 199–220.

Steels, L. (1996). Emergent adaptive lexicons. In P. Maes, M. J. Mataric, J. A. Meyer, J. Pollack, and S. W. Wilson (Eds.), *From Animals to Animat 4. Proceedings of the Simulation of Adaptive Behavior Conference*, pp. 562–567. The MIT Press.

Teller, A. (1994). The evolution of mental models. In *Advances in Genetic Programming*. The MIT Press / Bradford Books.

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

Wilson, S. W. (1985). Knowledge growth in an artificial animal. In L. E. Associates (Ed.), *Proceeding of the First International Conference on Genetic ALgorithms and Their Applications*, pp. 16–23.

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

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

Wilson, S. W. (1998). Generalization in the XCS classifier system. In J. K. et al (Ed.), *Proceedings of the Third Annual Genetic Programming Conference*, Madison (WI), pp. 665–674. Morgan Kaufmann San Francisco (CA).