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

A Model of the Environment to Avoid Local Learning

Pier Luca Lanzi
Technical Report N. 97.46
December 20th, 1997

A Model of the Environment to Avoid Local Learning

(An Analysis of the Generalization Mechanism of XCS)

Pier Luca Lanzi

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

Technical Report N. 97.46

December 20^{th}, 1997

Abstract

We analyze generalization with the XCS classifier system when the system is applied to animat problems in grid-worlds. The aim of the paper is to give an unified view of generalization with XCS, in order to explain some of the phenomena reported in the literature.

Initially, we extend the results previously presented in the literature applying XCS to three environments. Our results confirm what already reported in the literature, showing that the generalization mechanism of XCS may prevent the system from converging to optimal performance. Accordingly, we study XCS generalization mechanism analyzing Wilson's generalization hypothesis and the conditions under which it may fail.

We draw a hypothesis in order to explain the results we report. We hypothesize that XCS fails to learn an optimal solution when, due to the environment structure and to the exploration strategy employed, the system is not able to explore all the environmental niches frequently. We validate our hypothesis introducing a new exploration strategy which guarantees a frequent exploration of all the areas of the environment, we call it *teletransportation.*

Teletransportation is not introduced as a real solution to the problems we evidenced, since it is not feasible for real applications; rather, we exploit teletransportation as a tool to validate our hypothesis.

Nevertheless, as we subsequently show, the ideas which support teletransportation can be implemented integrating XCS with a model of the environment, learned by experience, in a *dyna* architecture.

We end of the paper discussing another important aspect of generalization in XCS: the conditions under which XCS may fail to produce a compact representation of a learned task. We show this is likely to happen in environments where there is no direct relation between the number of don't care symbols a classifier condition has, and the number of environmental conditions the classifier matches. Accordingly, we discuss the role of subsumption deletion for the problem of evolving a compact representation of the learned task.

1 Introduction

Generalization is the most interesting feature of XCS, the classifier system introduced by Wilson [9]. XCS has in fact been proved to evolve near-minimal populations of classifiers that are accurate and maximally general. Recently, Kovacs [4] has proposed an optimality hypothesis for XCS and showed experimental evidence of such hypothesis with respect to the function representation problem involving multiplexers.

Conversely, for animat problems, [5] presented experimental results showing that XCS may fail to converge to an optimal solution. The author observed that, due to a strong genetic pressure, XCS could be unable to recover dangerous situations in which overgeneral classifiers are likely to corrupt the population. [5] relates this phenomenon with the structure of the environment, specifically with amount of generalizations that the problem allows, and suggests it is more likely to happen in environments which allows few generalizations. However, [5] did not discuss the concept of *environment which allows few/many generalizations* and keep it intuitive.

In order to help XCS to recover from the presence of overgeneral classifiers, [5] introduced a new operator named *Specify* that counterbalances the effect of generalization when such situations occur. Experimental results confirmed that Specify successfully adapts overgeneral classifiers when the system can be prevented to converge.

Wilson [12] observed that the behavior discussed in [5] depends on the amount of random exploration the agent does; specifically, if the agent wanders around too much in between arrivals at the goal it can fail to reach optimal solution. Wilson proposes a different solution in which random exploration, usually employed in XCS, is replaced with *biased* exploration (also pseudo-random exploration according to [1]), which works as follows. When in exploration, the animat decides with a certain probability (P_s) whether to select the action randomly or to select the action which predicts the maximum payoff. Biased exploration reduces the amount of random exploration that the animat performs, therefore the animat concentrates on best policies which will reproduce more.

Experimental results, not presented here, report that XCS with biased exploration successfully solves the simple problem proposed in [5]. Nevertheless, as we show in the first part of this paper, it does not guarantee the convergence to an optimal solution in more difficult environments.

The analysis of XCS's behavior has always been presented without considering the relation between XCS's performance and the environment structure [4, 5]; specifically, it is not clear why an environment is easy to solve while a very similar one can be much more difficult.

The aim of this paper is to answer to such questions in order to get a better understanding of the generalization mechanism of XCS, while giving a unified view of what observed in [5] and [12]. First we extend the results presented in [5] comparing the two solutions proposed to counterbalance generalization mechanism: Specify and biased exploration. The comparison is done in two new environments, `Maze5` and `Maze6`, and subsequently in `Woods14`, the well-known ribbon problem which was firstly introduced by Cliff & Ross in [2]. These comparisons are not intended to show the best strategies to solve the proposed problems, accordingly, all the experiments employ standard parameter settings and not specific ones because, even if these could show better performance, they surely would lack in generality. Results we present evidence that Specify better adapts to all the environments while XCS with biased exploration fails to converge to good solutions as the environment complexity increases.

Although these results are interesting, they only report experimental evidence and do not explain XCS behavior, that is our major goal. We then try to explain XCS's behavior analyzing the assumptions which underlie generalization in XCS and Wilson's *generalization hypothesis* [9]; we study XCS's generalization mechanism in depth and formulate a specific hypothesis.

In short, we hypothesize that XCS fails in learning optimal behavior in those environments where some assumptions on which the generalization mechanism is based do not hold. Specifically, we claim that XCS does not learn an optimal policy when, due to the environment structure and to the exploration strategy employed, the system is not able to visit all the areas of the environment frequently. We validate our hypothesis through a series of experiments. The validation phase is employed to derive some possible solutions for the problems previously discussed.

Specifically, we introduce a novel meta-exploration strategy which can be employed in order to guarantee a uniform exploration of the environment. The strategy we propose, we call it *teletransportation*, is not intended to be a solution to XCS problems discussed in the first part of this paper. The strategy, in fact, cannot be employed in real problems, such as physical autonomous agents; *teletransportation* is rather a way (a tool) we use to validate our hypothesis.

We subsequently present a possible solution, which implements the idea of teletransportation for real problems: the solution consists of integrating XCS in a *dyna* architecture where a model of the environment, built by experience, is employed.

We end the paper discussing another important aspect of generalization with XCS that is, the capability of XCS to evolve a compact representation of the learned task. We show that XCS is not likely to evolve a compact representation in environments where there is not direct relation between the number of don't care symbols in a classifier condition and the number of environmental sensory configuration the classifier condition matches. Most important, in such situations XCS may evolve a redundant representation of the task. Accordingly, we discuss the importance of subsumption deletion [11] in order to evolve a compact representation.

The analysis, we present in this paper, does not concern XCS in general, instead it discusses the generalization mechanism of XCS when animats are applied in grid-worlds. Nevertheless, since our study is mainly based on Wilson's generalization hypothesis, our conclusions appear to be very general and we believe they can be applied to other problems (environments). Our approach to the study of generalization in XCS leads us to two important results; our analysis in fact:

- Gives an effective contribution for understanding generalization in XCS;
- Effectively guides the development of solutions that improve XCS.

The paper is organized as follows. Section 2 briefly overviews XCS according to [9], while Section 3 presents the design of experiments we employed in the paper. XCS with Specify, called XCSS, and XCS with biased exploration are compared in Section 4 using two new environments, `Maze5` and `Maze6`; in Section 5 the same comparison is done using the `Woods14` environment.

The results of the previous sections are discussed in Section 6, where we formulate a hypothesis in order to explain why XCS may fail to converge in certain environments. Different aspects and implications that our hypothesis introduces are discussed in the same section. We validate our hypothesis in Section 7, introducing a novel (meta-) exploration strategy, that we call *teletransportation*. In Section 8 we show how teletransportation can be implemented in real applications.

Section 9 discusses another aspect of generalization in XCS that is, the conditions under which XCS evolves a compact representation of a learned task. Finally, Section 10 ends the papers, drawing some conclusions and the directions for future work.

2 Description of XCS

We now give a brief review of XCS in its most recent version [10]. We refer the interested reader to [9] for the original XCS description or to Kovacs's report [4] for a more detailed discussion for

implementors.

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

At each time step the system input is used to build a match set [M] containing the classifiers in the population whose condition part matches the detectors. If the match set is empty a new classifier that matches the input sensors is created through *covering*. For each possible action a_i the *system prediction* $P(a_i)$ is computed as the fitness weighted average of the classifier predictions that advocate the action a_i in the match set [M]. The value $P(a_i)$ gives an evaluation of the expected reward if action a_i is performed. Action selection can be *deterministic*, the action with the highest system prediction is chosen, or *probabilistic*, the action is chosen with a certain probability among the actions with a not null prediction.

The classifiers in [M] that propose the selected action are put in the *action set* [A]. The selected action is performed and an immediate reward r_{imm} is returned to the system together with a new input configuration.

The reward received from the environment is used to update the parameters of the classifiers in the action set corresponding to the previous time step $[A]_{-1}$. Classifiers parameters are updated as follows.

First, the Q-learning-like payoff P is computed as the sum of the reward received at the previous time step and the maximum system prediction discounted by a factor γ ($0 \leq \gamma < 1$). P is used to update the prediction p_j by the Widrow-Hoff delta rule [8] with learning rate β ($0 \leq \beta < 1$): $p_j \leftarrow p_j + \beta(P - p_j)$. Likewise, the prediction error ε_j is adjusted with the formula: $\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$. Fitness update is slightly more complex. Initially, the prediction error is used to evaluate the classification accuracy κ_j of each classifier as $\kappa_j = exp(\ln \alpha(\varepsilon_j - \varepsilon_0)/\varepsilon_0)$ if $\varepsilon_j > \varepsilon_0$ or $\kappa_j = 1$ otherwise. Subsequently the relative accuracy κ'_j of the classifier is computed from κ_j and, finally, the fitness parameter is adjusted by the rule $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

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

An important innovation, introduced with XCS is the definition of *macroclassifiers*. Whenever

a new classifier has to be inserted in the population, it is compared to existing ones to check whether there already exists a classifier with the same condition/action pair. If such a classifier exists then the new classifier is not inserted but the *numerosity* parameter of the existing classifier is incremented. If there is no classifier in the population with the same condition/action pair then the new classifier is inserted in the population. Macroclassifiers are essentially a programming technique that speeds up the learning process reducing the number of *real*, macro, classifiers XCS has to deal with.

Since XCS was presented, two genetic operators have been proposed as extensions to the original system: Subsumption deletion [10] and Specify [5].

Subsumption deletion has been introduced to improve generalization capabilities of XCS. Subsumption deletion acts when classifiers created by the genetic component have to be inserted in the population. Offspring classifiers created by the GA are replaced with clones of their parents if: (i) they are specialization of the two parents that is, they are "subsumed" by their parents, and (ii) the parameters of their parents have been updated sufficiently. If both these conditions are satisfied the offspring classifiers are discarded and copies of their parents are inserted in the population; otherwise, the offspring classifiers are inserted in the population.

Specify has been proposed to counterbalance the pressure toward generalization in environments that allow few generalizations. Specify acts in the action set when oscillating classifiers are detected; this condition is detected comparing the average prediction error of classifiers in the action set $\varepsilon_{[P]}$ with the average prediction error of classifiers in the population $\varepsilon_{[P]}$. If $\varepsilon_{[A]}$ is twice larger than $\varepsilon_{[P]}$ and the classifiers in [A] have been updated, on average, at least N_{Sp} times then a classifier is randomly selected from [A] with probability proportional to its prediction error. The selected classifier is used to generate one offspring classifier in which each # symbol is replaced with a probability P_{Sp} with the corresponding digit in the system input. The resulting classifier is then inserted in the population and another is deleted if necessary.

3 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`" symbol. An animat placed in the environment must learn to reach food cells. The animat senses the environment by eight sensors, one for each adjacent cell, and 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 and eats the food and receives a reward equal to 1000, otherwise, if the destination cell contains a tree the move does not take place. Each sensor is represented by two bits: `10` indicates the presence of a tree `T`, `11` indicates the food, while `00` stands for and empty cell; classifiers conditions is 16 bits long, while the eight actions are represented with three binary bits.

Each experiment consists of a number of problems that the animat must solve. For each problem the animat is randomly placed in a blank cell of the environment; then it moves under the control of the system until it enters a food cell, eats the food, receiving the reward that is constant. The food immediately re-grows and a new problem begins. We employed the following explore/exploit strategy: before a new problem begins the animat decide with a 50% probability whether it will solve the problem in exploration or exploitation.

We employed two type of exploration strategies, *random* and *biased*. When in random exploration the system selects the action randomly among the ones in the match set. During biased exploration the system decides with a probability P_s whether to select actions randomly or to choose the action which predicts the highest reward (a typical value for P_s is 0.5).

When in exploitation the animat always selects the action which predicts the highest reward and the GA does not act.

Performance of XCS is computed as the average number of steps to food in the last 50 exploitation problems. Every statistic presented in this paper is averaged on ten experiments.

4 XCS in `Maze5` and `Maze6`

A first analysis of the generalization capabilities of XCS for animat problems has been presented in [5] where the authors observed that XCS may fail to converge to an optimal policy, even for simple problems, when, due to a strong genetic pressure, the generalization mechanism is not able to recover from dangerous situations, in which overgeneral classifiers can corrupt the population. The *Specify* operator was thus introduced in order to help the generalization mechanism in recovering from overgeneral classifiers.

Wilson [12] observed that the behavior discussed in [5] also depends on the amount of random exploration the agent does: If the agent wanders around too much in between arrivals at the goal it can fail to converge to optimal performance. Accordingly, Wilson proposes a different solution in

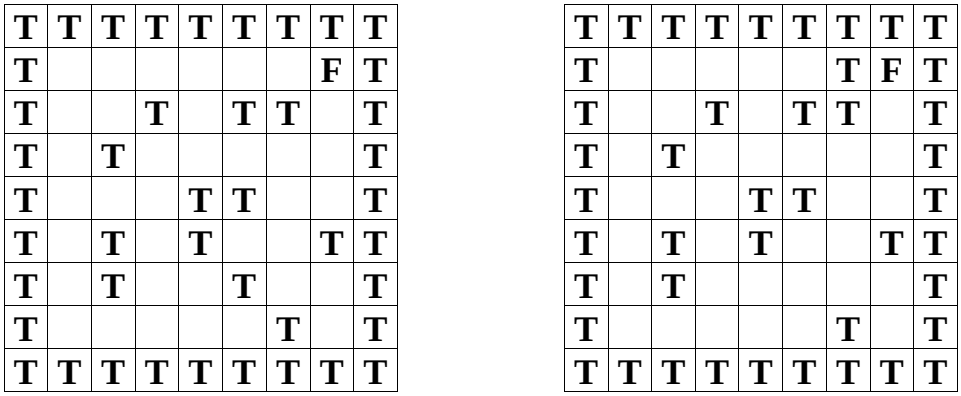

Figure 1: (a) The `Maze5` environment and (b) the `Maze6`environment.

which the amount of random exploration that the agent performs is reduced by replacing random exploration (usually employed in XCS) with biased exploration.

We now extend previous results presented in the literature, and compare the two solutions using two Markovian environments: `Maze5` and `Maze6` (Figure 1.a and Figure 1.b). These environments are very similar since they differ in few positions only, nevertheless `Maze6` is much more difficult to solve than `Maze5`, as our experiments will confirm.

We compare four algorithms for each environment: (i) XCS according to the original definition [9] that is, without subsumption deletion; (ii) XCS without don't care symbols (# are not introduced in the initial population, neither in covering or mutation); (iii) XCS with Specify [5], called XCSS; (iv) XCS with biased exploration [12].

The performances of algorithms (i) and (ii) are two important references. The former in fact indicates what the original system can do when the generalization mechanism works; while the performance of algorithm (ii) defines what are the potential capabilities of XCS, hence it can be considered an upper limit to XCS's performance.

4.1 The `Maze5` Environment

We apply the four algorithms to `Maze5` using a population of 1600 classifiers. General parameters are set as follows:

> β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_{\#}$=0.3, P_I=10.0, ε_I=0.0, F_I=10.0; for algorithm (ii) $P_{\#}$=0.0 and mutation do not insert any # symbol; while in (iii) Specify parameters are set as N_{Sp}=20 and P_{Sp}=0.5; finally, *biased exploration* chooses and action randomly with a probability P_s=0.3.

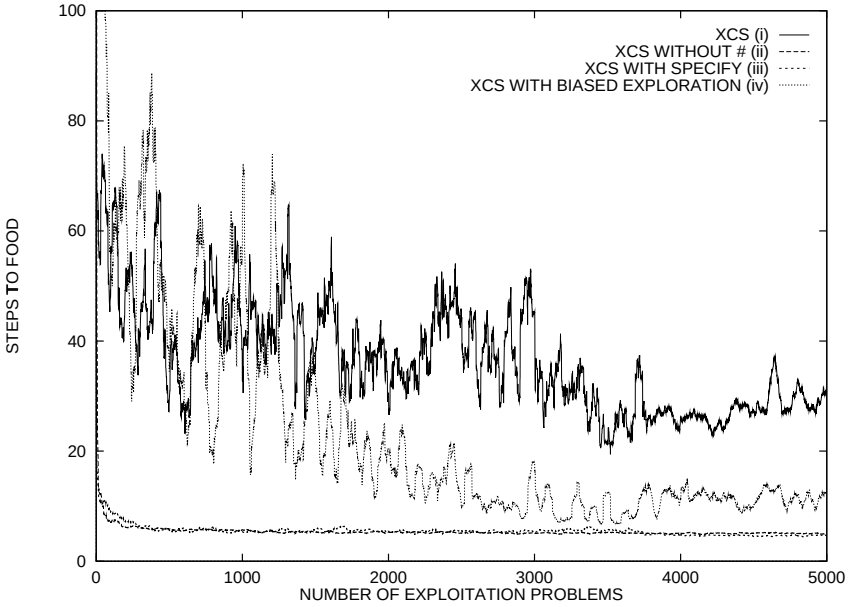

Figure 2: Performance of the four algorithms in Maze5. Curves are averaged over ten runs.

Results for the four algorithms are shown in Figure 2, curves are averaged on ten runs. As the curves show, XCS evolves a suboptimal solution for `Maze5` (algorithm (i)). In contrast, when generalization does not act, i.e. no # are used, the system easily reaches optimal performance (algorithm (ii)). This means that XCS can potentially solve `Maze5`.

When a mechanism to counterbalance generalization is added to XCS, we obtain better results: both algorithms (iii) and (iv) in fact converge to optimal performance. Specifically, XCS with biased exploration (algorithm (iv)) converges slower than XCSS (algorithm (iii)). Moreover, experimental results, not reported here, show that sometimes XCS with biased exploration fails to converge even to a suboptimal solution, while XCSS always reaches an optimal solution that is also very stable.

[5] observed that XCSS is very stable with respect to the population size parameter; we thus apply XCS with biased exploration and XCSS to `Maze5` using 800 classifiers only. Results in Figure 3 show that, even with a small population size, XCSS converges to a suboptimal solution in a stable way. In contrast, XCS performance significantly decreases evidencing wide oscillations.

4.2 The `Maze6` Environment

`Maze6` is derived from `Maze5` by covering a small number of free cells with obstacles. The two environments are thus very similar from a topological point of view, although `Maze6` is a much

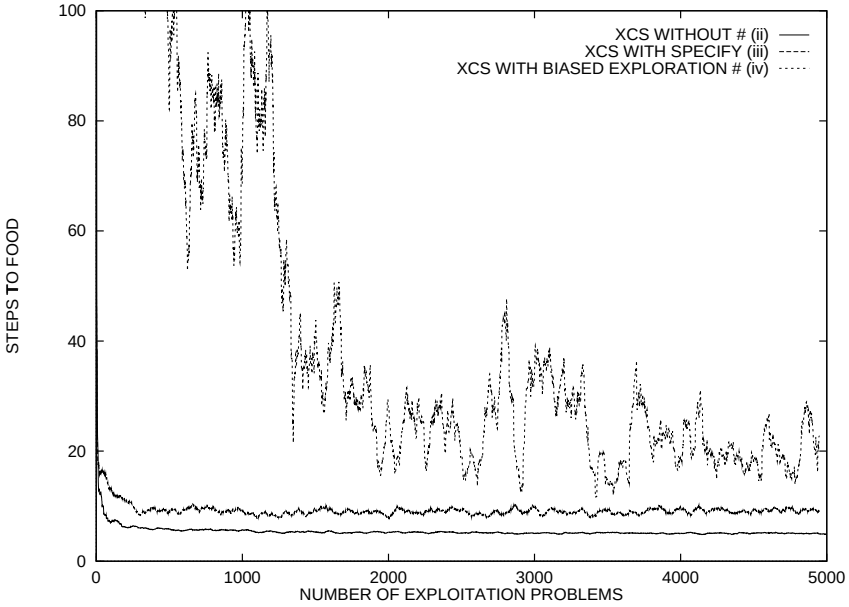

Figure 3: XCS without #, and XCS with Specify, with 800 classifiers. Curves are averaged over ten runs.

more difficult environment for XCS. [1]

We apply the four versions of XCS to `Maze6`. The performances reported in Figure 4 confirm the results we had for `Maze5`. XCS does not converge to an optimal solution when generalization acts, while when # symbols are not employed the system easily reaches optimal performance. Most important, there is almost no difference between the performance of XCS with random exploration and XCS with biased exploration. In contrast, as Figure 4 shows, XCS with Specify easily converges to optimal performance that is also stable.

Comparing the performance of XCS in the two environments (Figure 2 and Figure 4) it is worth noting that although the two environments are very similar, the performance of XCS in `Maze6` is at least five times worse than in `Maze5`. [2]

These results evidence that when the environment becomes more complex, biased exploration is not an acceptable solution anymore, in fact XCS with biased exploration fails to reach even a suboptimal performance. XCSS instead converges to an optimal solution for `Maze6` even if the population size is reduced to 800 classifiers, as shown in Figure 5.

[1] `Maze6` has been built to show how very similar environments can lead to extremely different results, in order to stress the importance of counterbalancing the generalization mechanism of XCS in some applications.

[2] The scale of the ordinate axis in Figure 4 represent values up to 500 steps, while in Figure 2 its values are limited upto 100 steps.

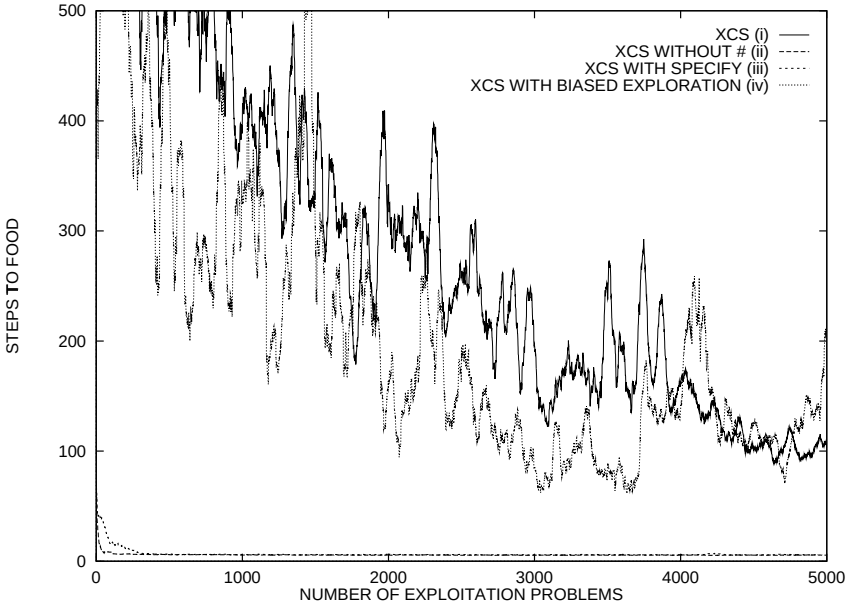

Figure 4: Performance of the four versions of XCS in Maze6. Curves are averaged over ten runs.

4.3 The Specify Operator and Biased Exploration

Results presented in this section confirm the analysis presented in [5]. Specify successfully helps the generalization mechanism to recover dangerous situations which could corrupt the population. On the other hand, biased exploration is successful in simple environments, such as `Maze5`, but it is unfeasible in more complex environments.

In our opinion, this happens because biased exploration is a *global solution* to the problem of balancing the generalization mechanism, while Specify is a *local* solution. [5] in fact observed that XCS's generalization mechanism acts in *environmental niches*, and these should be considered a sort of *fundamental element* for operators in XCS. Specify follows this principle and recovers potentially dangerous situations directly in the niches where they are detected; biased exploration instead acts on the whole population and it must take into account the structure of the entire environment.

5 XCS in `Woods14`

Results previously presented in the literature for ZCS (the system from which XCS was derived) show that the failure in learning an optimal policy, depends on the length of the sequence of actions that are required to reach food: the longer the sequence is, the more difficult the environment.

Our experiments with XCS in `Maze5` and `Maze6` might seem to confirm the results presented

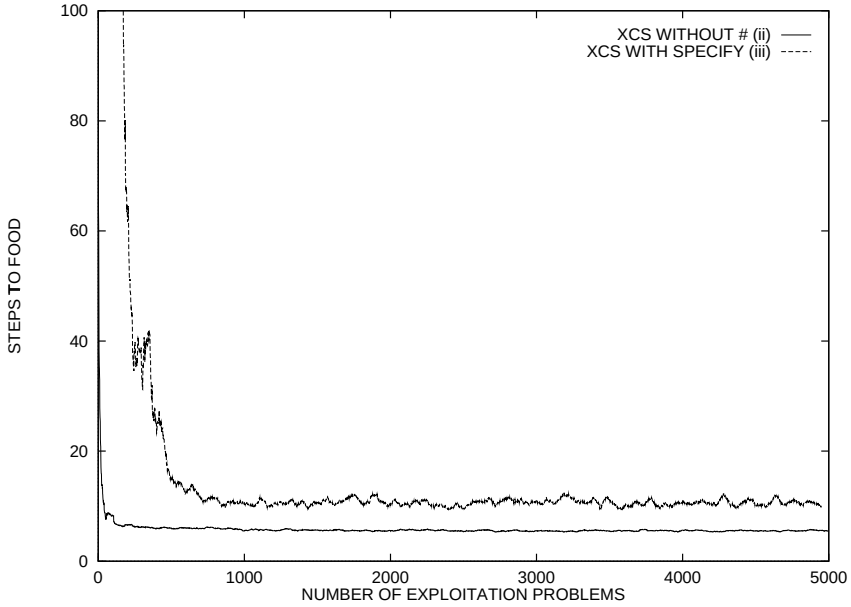

Figure 5: XCS without #, and XCS with Specify, with 800 classifiers in Maze6. Curves are averaged over ten runs.

for ZCS in fact, XCS converge to an optimal solution in `Maze5`, which requires, on the average, 4.75 steps to reach food. XCS performance decreases significantly when the system is applied to `Maze6`, where the animat takes, on the average, 5.05 steps to eat. However, XCS performance in the two environments is significantly different, while the length of the average paths to food in the two environments is very similar.

We now extend the results presented in the previous section, analyzing the performance of XCS in an environment which require long sequence of actions for reaching the goal state. For this purpose, we apply different versions of XCS in the `Woods14` environment [2].

`Woods14` (Figure 6) is a very simple environment, which consists of a linear path of 18 blank cells to a food cell and has an expected optimal path to food of 9 steps.

T	T	T	T	T	T	T	T	T	T	T	T	T
T	T	3	4	5	T	T	T	T	14	T	T	18
T	2	T	T	T	6	T	T	13	T	15	T	17
T	1	T	T	T	7	T	12	T	T	T	16	T
T	F	T	T	T	8	T	T	11	T	T	T	T
T	T	T	T	T	T	9	10	T	T	T	T	T

Figure 6: The Woods14 Environment. Free cells are numbered according to their distance from the food cell.

Initially, we apply XCS with biased exploration and XCS without generalization to `Woods14`

with a population of 2000 classifiers and the following parameter settings:

β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_\#$=0.3, P_I=10.0, ε_I=0.0, F_I=10.0; generalization is excluded in the second version of XCS, setting $P_\#$=0.0 and inhibiting mutation from inserting any # symbol.

The performance of XCS with biased exploration in `Woods14` is shown in Figure 7, while the performance of XCS when the generalization mechanism does not act, is shown in Figure 8; curves are averaged on ten runs. [3]

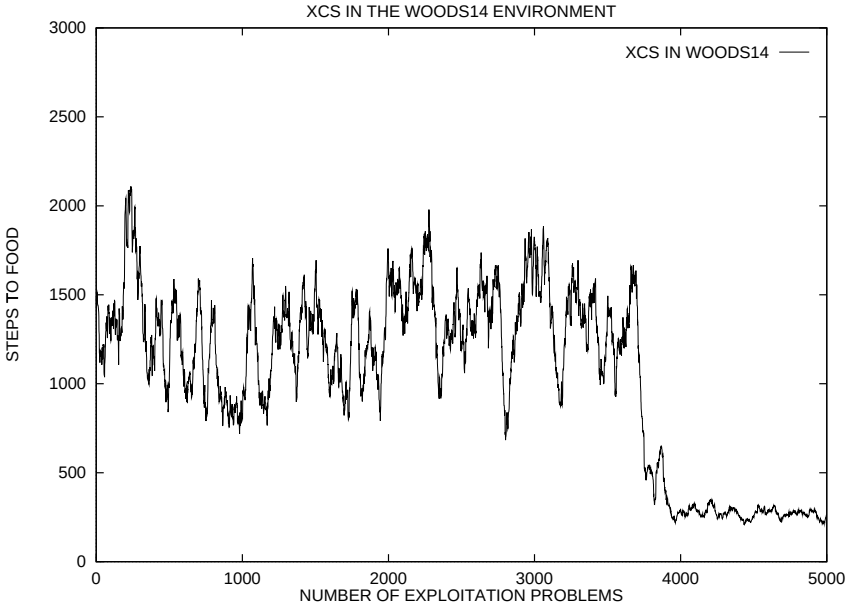

Figure 7: Performance of XCS in Woods14.

Results show that even if biased exploration is introduced, XCS is not able to converge to an optimal policy in `Woods14` when generalization acts; however, when # symbols are not used, XCS easily reaches the optimal performance. The former result could confirm that problems encountered with XCS depend on the length of the expected optimal path to food, but on the other hand, results in Figure 8 also suggest that XCS can solve problems which involve long sequences of actions.

Hence, we conclude that the convergence problems reported for all the three environments, `Maze5`, `Maze6` and `Woods14`, depend on the generalization mechanism of XCS. [4]

In the second experiment, we apply XCSS to `Woods14`. XCSS general parameters are set as for previous experiments, while for the Specify we set: N_{Sp}=20 and P_{Sp}=0.90. Figure 9 reports the

[3] Due to the significant difference of the results is not possible to use the same scale on both plots.

[4] Wilson [12] reports successful convergence for `Woods14` employing a version of XCS specifically developed for solving `Woods14`. Here, we do not discuss Wilson's solution since we are interested in the general issue concerning XCS convergence problems, and not specifically in `Woods14`.

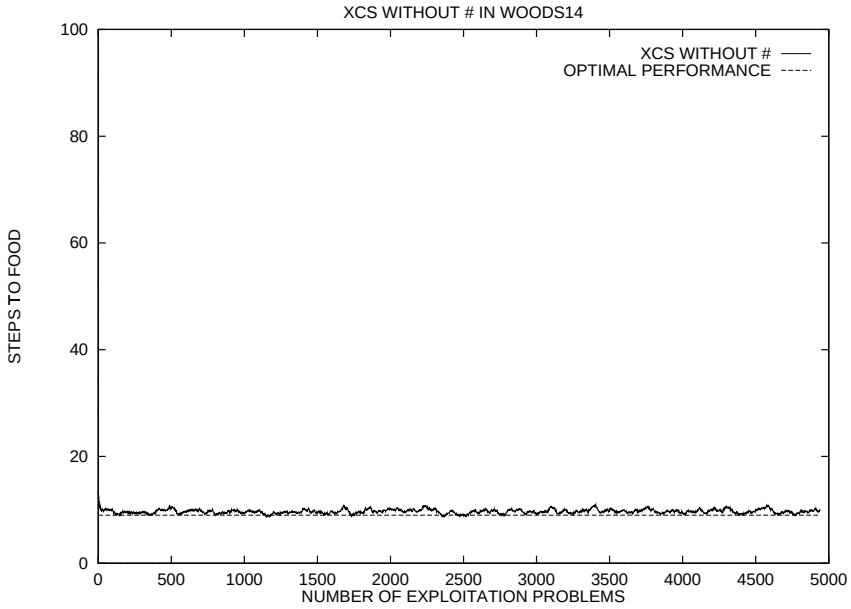

Figure 8: Performance of XCS without Generalization in Woods14.

performance of XCSS in `Woods14`; the curve, averaged over ten runs, show that XCSS can evolve an optimal solution for `Woods14`.

Although these results are interesting, they do not help us to explain the real causes which underlie the results we observed. In order to understand XCS's behavior, we need to study the generalization mechanism of XCS and Wilson's *Generalization Hypothesis* [9]; this is the subject of the next section where we discuss generalization capabilities of XCS and formulate an hypothesis to explain XCS's performance.

6 Generalization with XCS in Animat Problems

6.1 The Generalization Mechanism of XCS

Experimental results show that not all grid worlds are equivalent for XCS: There are environments such as `Woods2` and `Woods1` (see [9]) in which XCS easily produces optimal solutions; others, such as `Maze5`, `Maze6` and `Woods14`, require special exploration policies and/or special operators.

We now analyze the generalization mechanism of XCS in order to understand which factors influence the performance of the system. We start discussing Wilson's generalization hypothesis, which explains the fundamental principles of generalization in XCS as follows:

> "Consider two classifiers C1 and C2 having the same action, where C2's condition is a generalization of C1's. That is, C2's condition can be generated by C1's by changing

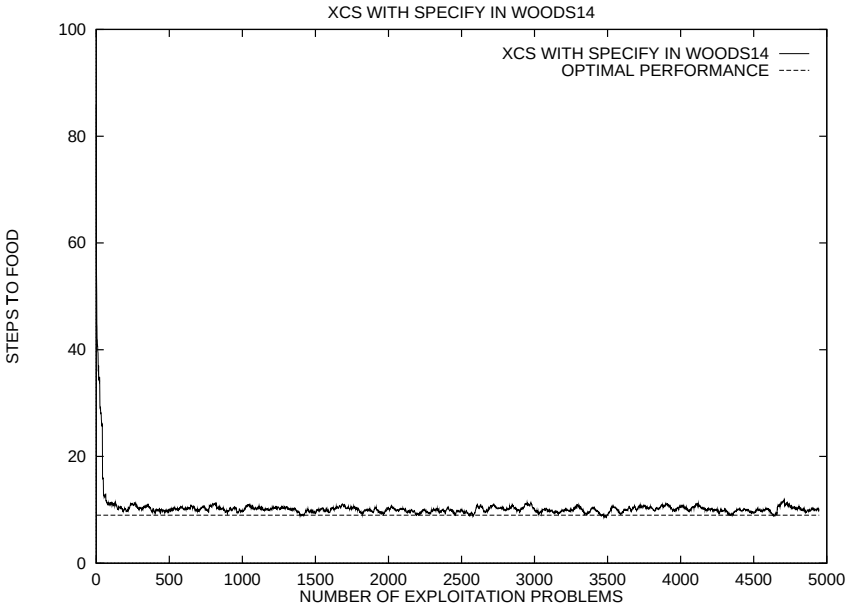

Figure 9: Performance of XCS with Specify in Woods14.

one or more of C1's specified (1 or 0) alleles to don't cares (#). [...]

Every time C1 and C2 occur in the same action set, their fitness values will be updated by the same amount. However, because is a generalization of C1 it will tend to occur in more match sets than C1, and thus probably (depending on the action-selection regime) in more action sets. Because the GA occurs in action sets, C2 will have more reproductive opportunities and thus its number of exemplars will tend to grow with respect to C1's [...]. Consequently, when C1 and C2 next meet in the same action set, a larger fraction of the constant fitness update would be "steered" toward exemplars of C2, resulting via the GA in yet more exemplars of C2 relative to C1. Eventually, it was hypothesised, C2 would displace C1 from the population." (Wilson 1995)

Wilson's hypothesis explains how XCS develops a tendency to evolve maximally general classifiers. But what happens when an overgeneral classifier appears in the population, is another useful key in order to understand why XCS sometimes fails to converge to an optimal solution:

Overgeneral classifiers are such that, due to the presence of some don't care symbols, they match different niches with different rewards and thus they will become inaccurate. But since in XCS the GA bases the fitness upon classifiers accuracy, overgeneral classifiers, that are inaccurate, tend to reproduce less. Evolved classifiers are thus as general as possible while still being accurate.

In the next section, we are going to analyze the generalization mechanism in details, in order to show why it may fail under certain conditions.

6.2 Are Overgeneral Classifiers Inaccurate?

The generalization mechanism is sound and thus it is not clear why it fails in certain environments. [5] observes that generalization in XCS is achieved through evolution; hence in case there is a strong genetic pressure, the generalization mechanism can be too slow to delete overgeneral classifiers, and these have enough time to corrupt the population. On the other, hand Wilson [12] observed that XCS performance can be severely compromised if overgeneral classifiers appear in an empty environmental niche. Both statements above are correct, nevertheless they do not explain the phenomenon that has been observed in previous sections.

We believe that Wilson's generalization hypothesis is correct accordingly, we claim that XCS fails in learning a certain task when some terms of the hypothesis do not hold. First, we observe that:

> **First Remark**
>
> *For overgeneral classifier to be "deleted", i.e. to reproduce less and then to be deleted, they must become inaccurate; however, this happens only if overgeneral classifiers are applied in distinct environmental niches.*

In other words, we suggest that in XCS it is not always true that an overgeneral classifier will become inaccurate, in fact, due to the parameter update, it becomes inaccurate only it receives different rewards. However, this only happens when the classifier is applied in different situations, i.e. environmental niches. Unfortunately, there are applications in which, due to the structure of the environment and to the exploration policy, the animat does not visit all the niches with the same frequence, but rather it stays in an area of the environment for a while and then moves to another one. In such situations, Wilson's generalization hypothesis fails, and overgeneral classifiers which should be inaccurate are evaluated as accurate. The fact we observed, that we are going to discuss with an example, greatly influences the performance of XCS:

- Since in XCS fitness is based on accuracy, it may happens that overgeneral classifiers, that are evaluated as accurate and accordingly should be deleted from the population, are instead reproduced.

- Moreover, if subsumption deletion is employed, overgeneral classifiers may get more resources than specific and non overgeneral classifiers.

As a first example, consider an overgeneral classifier that matches two niches which belong to two different areas of the environment. While the system stays in the area to which the first niche belongs, its parameters will be updated accordingly to the reward it receives and thus, as long as the animat does not visit the second niche, the classifier is accurate even if it is globally overgeneral. Therefore, the overgeneral classifier is selected for reproduction and accordingly the system is going to allocate resources, i.e. copies, to it. When the animat moves to the other area of the environment the classifier starts becoming inaccurate, since the reward it predicts is no longer correct. At this point, two things may happen. If the classifier did not reproduce sufficiently in the first niche, the classifier may be deleted because it has become inaccurate; consequently, the animat "forgets" what it learned in the other area. Otherwise, if the overgeneral classifier reproduced sufficiently when in the initial niche, the (macro) classifier survives enough to adjust its parameters in order to become accurate with respect to the current niche. Therefore, the overgeneral classifier continues to reproduce and mutate in the new niche, and it can produce even more overgeneral offspring. This behavior can be summarized as follows:

Second Remark

XCS usually learns a global policy however, if the environment is not, or cannot be, visited frequently, it tends to learn a local policy that can corrupt the information about other areas of the environment.

A more specific example can be presented using `Woods14`. For simplicity's sake, the example does not consider some possible interactions between classifiers in the population and some secondary factors that can influence evolution. The example is in fact presented in order to discuss only the main principles which underlie the phenomenon we observed. Consider the classifier `(A)` with condition `1010001010001010` and action `101` i.e. *go south-west*, that matches the the 3^{rd} cell of the environment. Suppose also `(A)` has been experienced sufficiently and therefore it is very accurate. A classifier `(B)` with condition `1010001010#010#0` and with the same parameters and the same action as `(A)` is inserted in the population after the GA has been applied to classifier `(A)`. The classifier `(B)` is overgeneral, since it matches two distinct positions in the environment, the 3^{rd} and 9^{th} cell, but its action, in each cell, should predict different rewards. Instead, the predicted reward of `(B)` is accurate when in position 3, and incorrect for position 9.

According to Wilson's hypothesis, classifier (B) should become inaccurate and finally it should be deleted from the population. But, due to the structure of the environment and to the exploration strategy, [5] the animat does not explore the environment uniformly, instead it spends more time in the central part of the corridor. Accordingly, during exploration, the animat will stay more time in the environmental niche corresponding to position 9 than in the one corresponding to position 3. As pointed out previously, two things may happens. If (B) has not reproduced sufficiently when in position 3, it can become inaccurate and all its copies are deleted from the population; in such case the animat loses what it learned in position 3. If (B) has been reproduced sufficiently, it survives adapting its parameters according to the reward received in the new niche; (B) therefore continues to reproduce in the new niche and in case it may produce offspring that are more overgeneral. The first hypothesis we formulate is thus that:

Our Hypothesis

XCS fails to learn an optimal policy in environments where: (i) XCS is not very likely to explore all the environmental niches uniformly; and (ii) overgeneral classifiers which match only a few niches are very likely to be produced.

The former statement (i) concerns the capability of the agent to explore all the environment in a uniform way, and thus is related to the environment structure and to the exploration strategy employed. Since the exploration strategies usually employed within XCS in animat problems select actions randomly, (i) is directly related to the average random walk to food: The smaller it is, the more likely the animat will be able to visit all positions in the environment uniformly; The larger the average random walk is, the more likely the animat is to visit more frequently a certain area of the environment. Statement (i) therefore explains why in certain environments XCS with biased exploration performs better than random exploration. When using biased exploration, the animat performs a random action only with a certain probability, otherwise it employs the best action. Accordingly, the animat is not likely to spend much time in a certain area of the environment but, following the best policy it learned, the animat moves to another area of the environment. Nevertheless, when the environmental niches are more distant, such as in Maze6 and Woods14, the animat is unable to change niche as frequently as it would be necessary in order to evolve an

[5] For sake of simplicity, we do not discuss a particular exploration strategy. Our conclusions applies to both the exploration strategy usually employed with XCS: random exploration and biased exploration.

optimal policy. This explains why XCS with biased exploration fails to converge to an optimal solution in "more complex" environments.

Statement (ii) is instead related with the environment structure only. We now discuss (ii) using `Woods14` as an example. In `Woods14`, each free position admits only two blank cells, therefore the optimal action in each cell can always be determined using two cells. A classifier condition can thus may have a lot of don't care symbols without being overgeneral; as an example consider the two conditions `1010001010001010` and `####0#####0#####` that match only the 3^{rd} cell in `Woods14`. A classifier condition becomes overgeneral only when at least one of the bits corresponding to an adjacent empty cell is replaced by a #. Hence, an overgeneral classifier in `Woods14` will match at most two niches [6] and thus, as the number of free cells in the environment grows, the probability of testing an overgeneral classifier in different environmental niches decreases. The two statements above can explain many phenomena reported in the literature concerning the performance of XCS.

6.3 Discussion

We presented a hypothesis in order to characterize the situations in which the generalization mechanism of XCS can prevent the system from learning an optimal policy.

The hypothesis we formulated concerns the concept of *environmental niche* and suggests that XCS can fail to converge to a *globally* optimal policy if the environmental niches are not explored frequently. We observe that the system should not explore some areas of the environments for a long time, instead it should frequently change environmental niche. Otherwise, XCS will start to learn locally, evolving classifiers which are correct with respect to a specific area but are inaccurate in some other area.

Our hypothesis is not a matter of the environment or of XCS alone but it depends upon the interaction between them. An environment in which the animat can easily visits all the possible areas will be easily solved by XCS with the usual random exploration strategy. However, as we are going to discuss in the next section, if the environment is complex it can be solved by XCS if a more adequate exploration strategy is employed.

At the end of this section, we want to point out that, although the approach we followed to study the behavior of XCS regards a specific kind of environments, i.e. grid-worlds, the conclusions we draw appear to be very general and therefore they can be extended to other environments as well.

[6] We do not discuss the trivial case of the classifier that matches all the niches, with a condition containing only don't care symbols.

7 A Meta-Exploration Strategy to Avoid Local Learning

In the previous section, we presented a hypothesis in order to characterize the situations in which the generalization mechanism of XCS eventually prevents the system from learning an optimal policy.

We now present an empirical validation of the hypothesis while we propose a possible solution which consists in a new (meta-) exploration strategy, see Section 7.1. The strategy we introduce can be applied to any exploration strategy previously employed with XCS, accordingly we refer to it as a meta-exploration strategy rather than an exploration strategy. The section ends with a discussion of the proposed solution and the possible extensions.

7.1 Teletransporting the Animat

According to the hypothesis presented in previous section, XCS fails to converge to optimal solutions in those environments where the system is not likely to explore all the environmental niches frequently. If our hypothesis is correct, an exploration strategy which guarantees a frequent exploration of all the environmental niches should solve all the problems evidenced for XCS.

In order to validate our hypothesis, we introduce a meta-exploration strategy which guarantees a frequent exploration of all the possible environmental niches. We call this strategy *teletransportation*, and it works as follows. When in exploration, the animat is placed randomly in a blank cell of the environment. Then it moves following one of the possible exploration strategies proposed in the literature, random or biased. If the animat reaches a food cell by a maximum number M_{es} of steps, the exploration ends; otherwise, if the animat does not find food by M_{es} steps, it is *teletransported* to another blank cell and the exploration phase is restarted. The proposed exploration strategy guarantees, for small M_{es} values, that the animat visits all the possible niches with the same frequence.

We apply XCS with teletransportation, we call it XCST, to the environments previously discussed (`Maze5`, `Maze6` and `Woods14`) using the same parameters settings employed in the original experiments. Figure 10 compares the performance of XCST and XCS with biased exploration in `Maze5`, when a population of 1600 classifiers is employed and M_{es} parameter is set to 20 steps.

Results show that, in `Maze5`, XCST converge to optimal solution more rapidly than XCS. Moreover, as Figure 11 shows, XCST performance is still optimal when only 800 classifiers are employed in the population. Nevertheless, since the genetic pressure is much stronger with 800

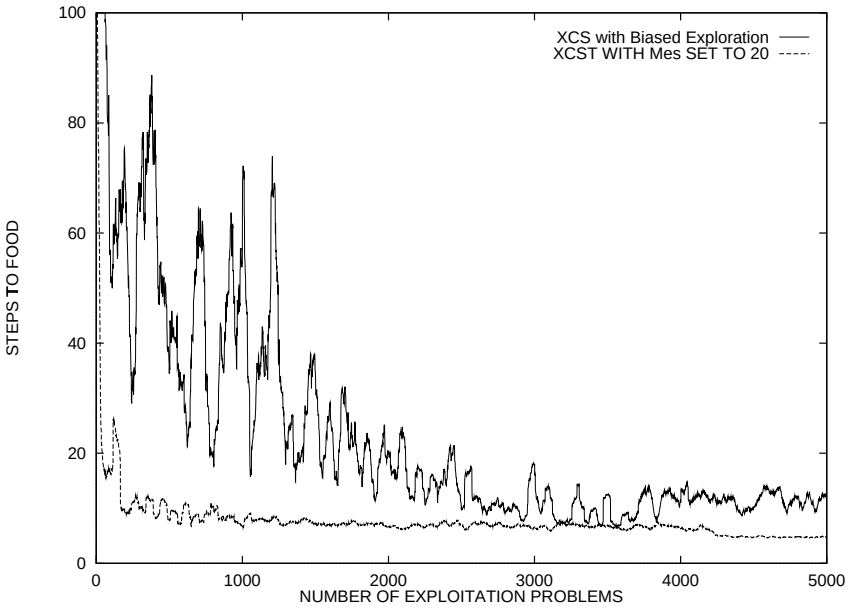

Figure 10: XCS with biased exploration and XCST in Maze5 with 1600 classifiers. M_{es} is set to 20 steps.

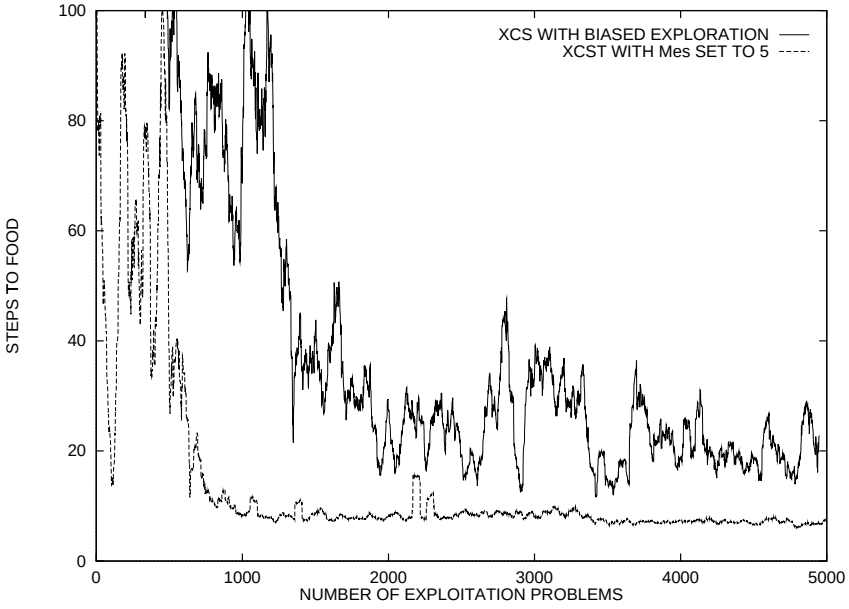

Figure 11: XCS with biased exploration and XCST in `Maze5` with 800 classifiers. M_{es} is set to 5 steps.

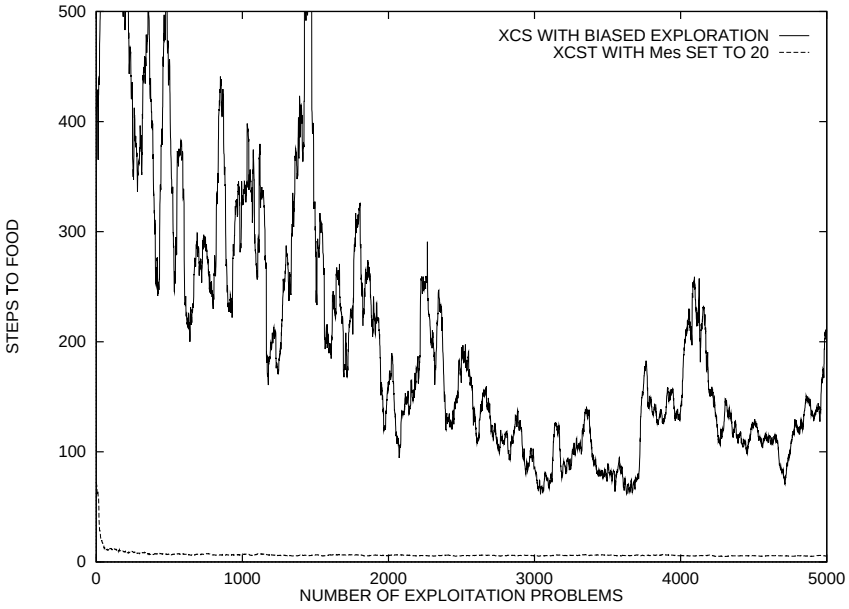

Figure 12: XCS with biased exploration and XCST in `Maze6`. M_{es} is set to 20.

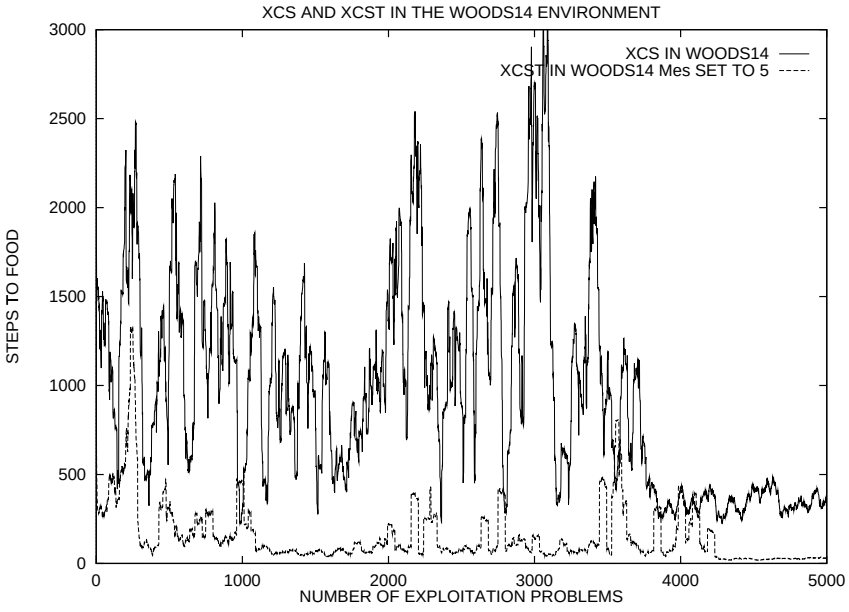

Figure 13: Typical performance of XCS with biased exploration and XCST in `Woods14`. Parameter M_{es} is set to 5 step.

classifiers, the M_{es} parameter has to be set to 5. Accordingly, the animat changes niches very often and explores all environmental niches very frequently.

Same results are shown when XCST is applied to `Maze6`, see Figure 12. A comparison of the performance for XCST and XCS shows that XCST converges to an optimal solution while XCS with biased exploration, for the same parameter settings, reaches a highly suboptimal performance. Nevertheless, experimental results, not reported here, show that if less than 1600 classifiers are employed XCST tends to converge very slowly to a suboptimal performance. This happens because teletransportation still applies evolution in order recover from overgeneral classifiers hence, teletransportation is still sensible to genetic pressure and therefore to the population size parameter.

Figure 13 shows a typical performance of XCS with biased exploration (solid line) and XCST (dashed line) when they are applied to `Woods14`. The immediate impression is that XCST performance is not very stable and, moreover it is only suboptimal. However, to understand Figure 13 truly, we have to analyze how XCST learns. When in exploration, XCST continuously moves in the environment in order to visit all the niches frequently. Accordingly, the animat does not learn the optimal policy in the usual way, by "trajectories", i.e. starting in a position and exploring until a goal state is reached.

XCST's policy instead *emerges* from a set of experiences of a limited number of steps the animat has collected while it was in exploration. The system thus immediately converges to optimal policies for positions near the food cells, then the policy is extended during subsequent explorations in the other areas of the environments. Accordingly, in `Maze6` the policy is extended very rapidly since the environment is quite simple, while in `Woods14` the analysis of single experiments evidences that XCST almost immediately learns an optimal policy for the first eight positions. Then the policy converges also for the subsequent next eight positions. At the end, the performance is still slightly suboptimal because for the last two position of `Woods14`, the most difficult ones, the optimal policy is not determined yet.

The experiments with XCST in `Woods14` therefore evidence a limit of teletransportation: Since the environment is explored uniformly the positions for which is difficult to evolve a optimal solution and would require more experience, converge very slowly to optimal performance.

8 Exploration, Generalization, Models and Animats

The teletransportation strategy, that we introduced in previous section, cannot be considered a solution to the problems that the generalization mechanism of XCS has evidenced. Teletransportation has been a way (a tool) we employed to validate our hypothesis concerning generalization in XCS. Teletransportation does not represent an effective solution since it cannot be employed in general problems, such as physical autonomous agents; teletransportation would in fact require the presence of a trainer that, every M_{es} steps, should pick up the agent and take it to another area of the environment. However, the idea underlying teletransportation can be employed to develop a more complete classifier system.

8.1 Related Works

As we pointed out previously, XCS usually learns a global policy but, it tends to evolve local policies in those environments where the agent is not able to visit all the areas with the same frequence. This problem is not novel in the area of reinforcement learning. In fact, many reinforcement learning algorithms, in order to converge to an optimal policy, require that the environment is visited uniformly. Moreover, when neural networks are employed, all the areas of the environment have to be explored with the same frequence, otherwise the neural network may overfit locally.

Solutions to this kind of problems for reinforcement algorithms have already been proposed in literature by Sutton [7] and by Lin [6]. Sutton proposed the *Dyna* architecture which integrates the learning algorithm with a model of the environment that is built by experience. The model is then employed to simulate exploration in other areas of the environment or for planning. Another solution is the one proposed by [6] that introduces the idea of *experience replay*: Past experienced trajectories to goal states are memorized and subsequently used to reproduce past experience in order to avoid local overfitting.

8.2 Dyna Architecture for XCS

Teletransportation can be implemented for real problems integrating XCS with a model of the environment that is built during exploration. Therefore, the analysis of the generalization mechanism of XCS, we present in this paper, motivates why it may be useful to integrate XCS with a model of the environment, developed by experience. The model can be employed as in [7] to simulate exploration in other areas of the environment, while the agent is actually exploring one specific

environmental niche, or otherwise for planning. In the following, we give a brief overview of a very simple version of dyna architecture we integrated in XCS that we are currently experimenting.

The simplest way to develop a model of the environment in a discrete state/action space, like grid-worlds, is to memorize the past experience as quadruples of the form (s, a, s', r), where: s is current sensory input, a is the action the agent selected when perceived s, s' is the sensory input returned when the agent perceiving s has performed a, finally r is the immediate reward the agent received for performing a. Such model is easily integrated with XCS; the overall system, we call it Dyna-XCS, works as follows. When in exploration, the animat is placed randomly in a blank cell of the environment and then it moves under the control of XCS using one the usual exploration strategy, random or biased. [7] If the animat reaches a food cell by M_{es} steps, the exploration ends. Otherwise, if the animat does not find food "in time" the system stop exploring the environment and start using the model of the environment in order to simulate an exploration experiment on the model. Accordingly, the current sensor configuration is memorized, and a new exploration starts on the model. Exploration on the model is very similar to the one the agent performs in the environment. First, the initial position is determined randomly among the states that appears in the first position of the quadruples that have been experienced. Then exploration continues on the model until S_{es} steps have been done, or the animat has reached a food cell, in the model. At this point the animat ends exploration in the model and restarts the exploration in the environment.

8.3 Discussion

The Dyna-XCS system we implemented is very basic, and the system is still under experimentation. As it can be noticed there are a number of issues that have to be discussed when integrating XCS in a Dyna architecture. First, the representation of the model we implement is very simple and, as the exploration in the environment proceeds, complete description of the environment is obtained; this means that such solution is feasible only when the environment is quite small.

Moreover, before a model can be employed to simulate the environment, a sufficient number of experiences, i.e. quadruples, must be collected. Finally, the two parameters M_{es} and S_{es} which indicate how much real exploration has to be done before starting simulated exploration in the model must be determined.

[7] We are currently experimenting a version of Dyna-XCS that employs a biased exploration strategy.

9 Evolving a Compact Representation

In the previous sections, we discussed how the generalization mechanism of XCS and the structure of the environment influence performance. Another important aspect of generalization in XCS concerns the capability of XCS to evolve a compact representation of a learned task.

9.1 Generalization and Task Representation in XCS

Results reported in the literature show that XCS can evolve near minimal populations of accurate and maximally general classifiers [11]. Recently, Kovacs [4] has proposed an optimality hypothesis which states that XCS tends to evolve the minimal population with respect to the boolean function representation problem.

We now discuss, with respect to animat problems, how XCS develops a tendency to evolve near minimal populations and we show how XCS may, in certain environments, fail to evolve a compact representation and, on the contrary, can produce redundant representations of a certain task.

Consider again `Woods14` (Figure 6), as noticed in Section 5, every position in `Woods14` is uniquely determined by the position of the two adjacent free cells. Therefore, in each classifier condition only two bits are sufficient to characterize a specific environmental niche. Since classifiers in `Woods14` are 16 bits long, for each niche there are 2^{14} possible classifiers that belongs to that niche only. According to Wilson's hypothesis, general classifiers should reproduce more than specific ones, since general classifiers appears in more match sets.

Unfortunately, this is not always true. Consider as an example the two conditions `1010001010001010` and `####0#####0#####`, although the second has much more don't care symbols both the conditions match only the 3^{rd} free position in `Woods14`. Wilson's hypothesis therefore fails in such a case, since two classifiers with those conditions will always match the same environmental niche. [8]

Since the pressure toward more general classifiers is lost, action sets in `Woods14` will consequently tend to contain a great number of classifiers which have different conditions, but almost all match the same niche. Therefore, each niche will be represented by a large number of (macro)classifiers that are equivalent. In such a case, the representation does not tend to be compact anymore, instead it tends to become very redundant. In XCS this situation can be easily detected since the number of macroclassifier and the number of (micro)classifiers in the action sets will tend to be

[8] This phenomenon was already noticed by Wilson in [9] where, discussing the generalization produced by XCS in `Woods2`, Wilson evidenced that XCS produced classifiers which matched the same niches but contained different amount of # symbols

the same.

Third Remark

XCS is not likely to evolve a compact representation in those environments where there is no direct relation between the number of # symbols in classifier conditions and the number of niches that classifier matches. Moreover, as the phenomenon becomes more noticeable, such as in `Woods14`*, the representation becomes highly redundant.*

As an example, we apply XCST to `Maze6` with a population of 1600 classifiers; Figure 14 reports the number of macroclassifiers in the population, the curve is averaged over ten runs. We notice that the number of macroclassifiers grows immediately and then reaches an equilibrium value which depends on the genetic pressure. The analysis of final populations shows that only few macroclassifiers represent more than one (micro)classifier. Hence, each niche is represented by more classifiers that are equivalent.

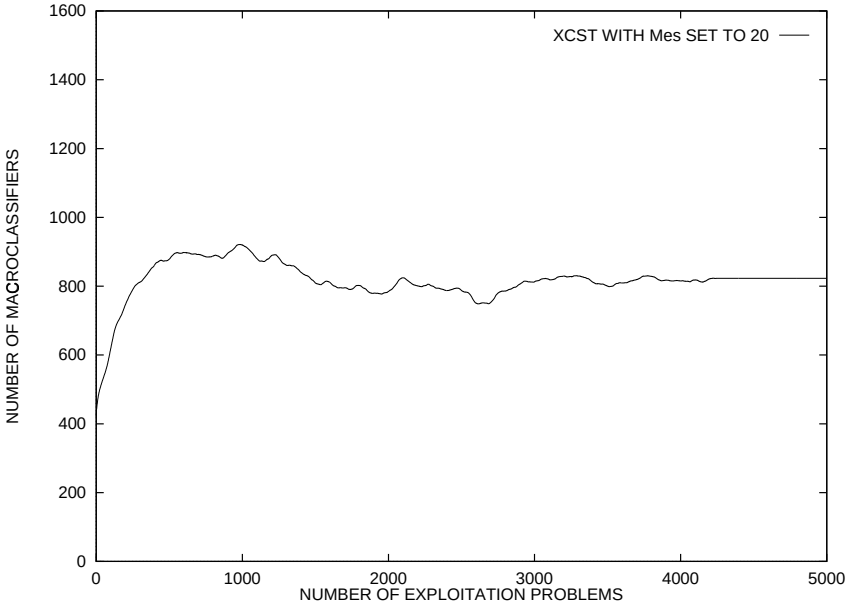

Figure 14: Number of Macroclassifiers for XCST with M_{es} set to 20 in Maze6.

9.2 How to Compact the Representation

We now face the problem of how the representation of the task, that the agent is learning, can be compacted in environments, such as `Woods14`, where a pressure toward more general classifiers cannot be developed. The obvious solution to this kind of problem consists in defining an operator which, acting in environmental niches, tries to produce a more compact representation.

However, before such an operator can be defined, we have to discuss what kind of generalization we are trying to evolve. We may decide that "more general" simply means "with more don't cares", accordingly the operator should collapse specific classifiers into more general ones. We define this type of generalization an "*a priori*" generalization, since # are inserted in the classifiers not because they are necessary in order to match more niches, rather the system inserts as much # as possible so that the classifier is not overgeneral. This is the type of generalization to which XCS tends to evolve when the subsumption operator is employed. Another possible solution would consist in devising an operator which tends to evolve a generalization that is maximally specific, in which # symbols are inserted only if they are necessary in order to match more niches.

The two types of generalizations might seem equivalent but it is worth noticing that the former tends to produce a population which is very likely to be overgeneral in case the environment is extended, as for example if a new area of the environment is discovered [12]. This happens because, with an "*a priori*" generalization, the system tends to produce classifiers which apply in much more conditions than the agent experimented. On the contrary, a generalization which is maximally specific produces classifiers that are as general as possible and cover only the sensory configurations that the agent has experimented.

According to the classification of generalization types we present, the *subsumption deletion* operator compacts the representation of a task learned according to an "*a priori*" generalization. Hence, the system can produce solutions that are likely to result overgeneral when new areas are discovered in the environment.

Wilson [12] suggests that in such cases the Specify operator can be a useful way to recover from classifiers that are overgeneral in the new areas.

However, as pointed out in Section 6, subsumption deletion relies upon the accuracy parameter and therefore it may corrupt the population when overgeneral classifiers are evaluated as accurate. We developed a series of experiments in which XCS with biased exploration was applied to the environments previously presented. Results, not reported here, show that the performance of the system is highly decreased when subsumption deletion is used. The same kind of result was reported by Wilson [12].

After we introduced teletransportation we repeated the set of experiments in order to test whether the decrease of performance when subsumption was used, depended on the presence of overgeneral classifiers that were evaluated as accurate. We applied XCST with Subsumption deletion to `Maze6`.

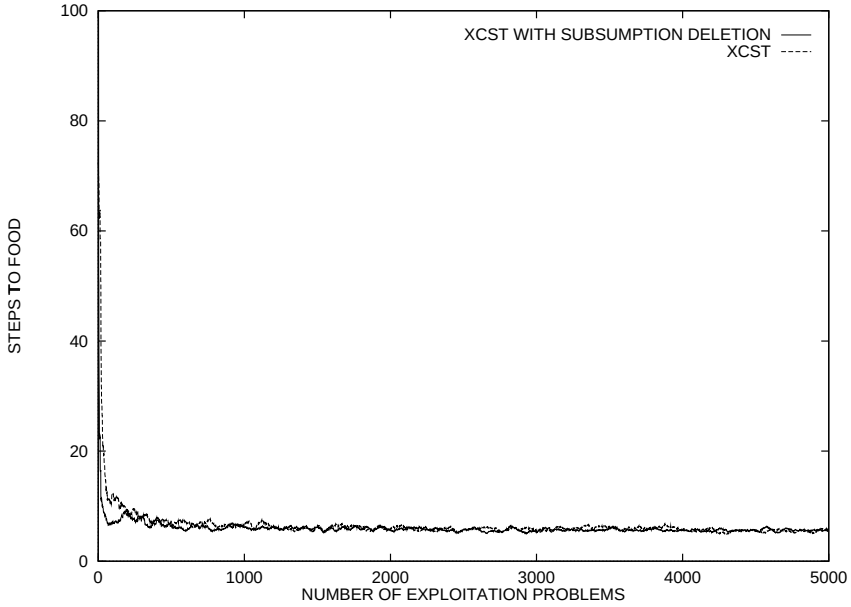

Figure 15: Performance of XCST with (solid line) and without subsumption deletion in `Maze6`. M_{es} is set to 20.

As Figure 15 shows, performance of XCST is not decreased when subsumption deletion is employed, [9] while the comparison of the population size in macroclassifiers for the two systems, see Figure 16, shows that subsumption deletion compact the representation resulting in a smaller number of macroclassifiers. As it can be noticed, the number of macroclassifiers does not decrease rapidly, instead it evidence a tendency to diminish.

10 Conclusions

We presented a study of the generalization mechanism of XCS, in order to explain some of the results previously reported in the literature. Accordingly, we analyzed Wilson's generalization hypothesis which explains how generalization in XCS works, and then we drew a hypothesis which suggests that XCS cannot converge to an optimal solution when, due to the structure of the environment and to the exploration strategy, the system is not able to visit all the areas of the environment frequently. We validated our hypothesis, introducing a novel (meta-) exploration strategy which guarantees frequent exploration of the environment. The strategy we introduced, called *teletransportation*, has not been proposed as a solution to the problems that XCS may evidence, rather *teletransportation* is a tool for validating our hypothesis. However, the idea

[9] Nevertheless, experimental results, not reported here, show that for a smaller parameter value, such as 20, the system does not converge to an optimal solution.

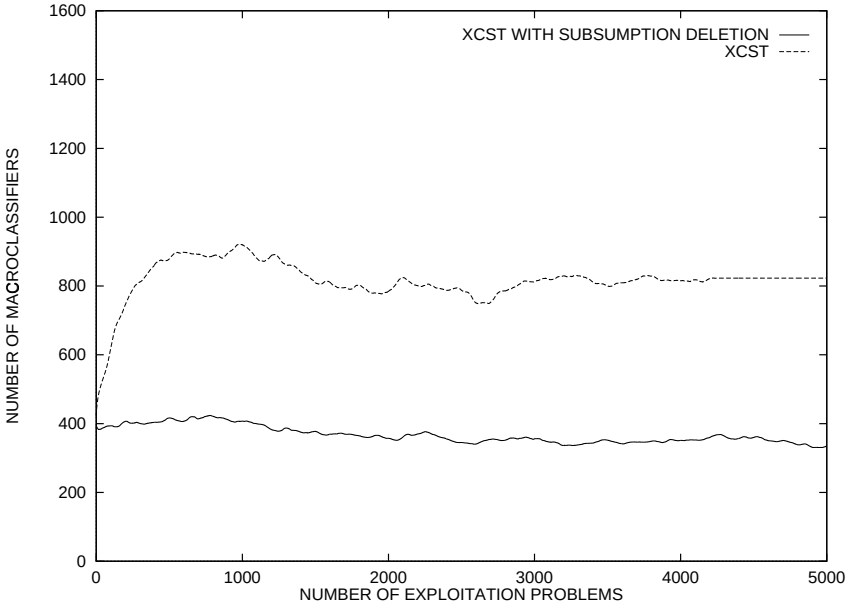

Figure 16: Population size in macroclassifiers for XCST with (solid line) and without subsumption deletion in `Maze6`.

underlying *teletransportation* can be implemented for real application integrating XCS with a model of the environment, learned by experience, in a *dyna* architecture. A possible implementation of such architecture, that at the moment we are experimenting, was then discussed.

Finally, we analyzed the conditions under which XCS may fail to evolve a compact representation of the learned task. We showed this is likely to happen in environments where there is no direct relation between the number of don't care symbols a classifier condition has, and the number of environmental configurations it matches. In such cases, there is no pressure toward generalization and the system can evolve a redundant representation of the learned task. Accordingly, operators are required in order to develop a pressure to more general classifiers. We thus analyzed how subsumption deletion can be successfully employed to compact the evolved solutions.

Acknowledgments

I wish to thank Stewart Wilson for reviewing an early version of this paper. Many thanks also to Marco Dorigo for the many afternoons spent discussing the generalization mechanism of XCS while I was in Brussels. Last but not least, Marco Colombetti who always encourages my work and discusses the many doubts I always have.

References

[1] P.V.C. Caironi and Marco Dorigo. Training and delayed reinforcements in q-learning. *International Journal of Intelligent Systems*, 12(10), 1997.

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

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

[4] Tim Kovacs. Evolving optimal populations with XCS classifier systems. Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U.K., 1996. Available from the technical report archive at http://www/system/tech-reports/tr.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] L.-J. Lin. Reinforcement learning for robots using neural networks. Technical Report CMU-CS-93-103, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1993.

[7] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, Austin, TX, 1990. Morgan Kaufmann.

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

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

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

[12] Stewart W. Wilson. Personal communication. 1997.