A Study of the Generalization Capabilities of XCS

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

Abstract

We analyze the generalization behavior of the XCS classifier system in environments in which only a few generalizations can be done. Experimental results presented in the paper evidence that the generalization mechanism of XCS can prevent it from learning even simple tasks in such environments. We present a new operator, named *Specify*, which contributes to the solution of this problem. XCS with the Specify operator, named XCSS, is compared to XCS in terms of performance and generalization capabilities in different types of environments. Experimental results show that XCSS can deal with a greater variety of environments and that it is more robust than XCS with respect to population size.

1 INTRODUCTION

XCS is a classifier system recently proposed by Wilson (Wilson 1995) which has a strong tendency to evolve near-minimal populations of accurate and maximally general classifiers. Experimental results reported in the literature show that XCS can learn a more compact representation than that learned by tabular Q-learning (Watkins 1989). Generalization in XCS is achieved mainly by the combination of two factors. First, classifier fitness in XCS is based on the accuracy of the classifier prediction instead of the prediction itself as in traditional classifier systems (Holland 1986). Second, the genetic algorithm in XCS acts in environmental niches, as opposed to the traditional panmictical GA. XCS evolves populations of accurate and maximally general classifiers. Because general classifiers match more niches, they reproduce more. But, since the GA bases the fitness upon classifiers accuracy, overgeneral classifiers, that are inaccurate, tend to reproduce less. Evolved classifiers are as general as possible while still being accurate.

Recently, *subsumption deletion*, has been introduced by Wilson (Wilson 1996) to improve generalization. Subsumption deletion acts in the GA and replaces offspring classifiers with clones of their parents if the parents subsume, that is are generalization of, the offspring. XCS with subsumption deletion has a strong tendency to generalization. Unfortunately, experimental results briefly reported in this paper, show that pressure to generalization can prevent the system from learning in very simple environments that do not show regularities, such as repeated patterns, or give very similar sensorial configurations for states in which different actions are required.

In this paper we study the problem of applying XCS in the Maze4 environment in which XCS fails to reach optimal performance. Initially, we briefly analyze the performance of XCS in Maze4. Experimental results presented evidence that generalization capabilities prevent XCS from converging to the optimal solution even if the proposed maze is very small (8×8 grid). Two types of solutions can be proposed for dealing with this problem: a *global* solution and a *local* solution. The global solution consists of reducing the pressure to generalization by modifying the XCS architecture. Unfortunately, our experiments suggest that this solution results in a classifier system which reaches optimal solutions in a large variety of environments but has a strongly reduced generalization capability. Instead, we propose a *local* solution in which an operator, named *Specify*, counterbalances the pressure toward generalization in situations were generalization can prevent learning but is excluded when the system successfully converges to an optimal population. Experimental results presented for XCS with the Specify operator, XCSS for short, show that the proposed system can learn in a greater number of environments than XCS. Moreover a comparison between XCS and XCSS on

woods environments shows that Specify does not eliminate the tendency to generalization of the original architecture, but rather slows the generalization process.

The rest of the paper is organized as follows. Section 2 gives a brief overview of XCS according to the most recent presentation by Wilson (Wilson 1996). Section 3 presents the woods environments, and the design of experiments for the results presented in the paper. Experimental results obtained with XCS in the small Maze4 environment are reported in Section 4, while in Section 5 these results are commented and a possible *global* solution, to the problem of offsetting the generalization mechanism, is discussed. Section 6 isolates the primitive components of generalization, discusses the *Mutespec* operator proposed by Dorigo (Dorigo 1993), and finally defines the *Specify* operator. Experimental results on the Maze4 environment for the proposed XCSS and XCS systems are compared in Section 7. The generalization capabilities of the two systems are evaluated and compared in Section 8. A conclusions section ends the paper.

2 OVERVIEW OF THE XCS CLASSIFIER SYSTEM

XCS is a classifier system developed by Wilson which differs from the traditional one defined by Holland (Holland 1986) mainly because (i) it has a very simple architecture, (ii) there is no message list, and most important (iii) the traditional *strength* is replaced by three different parameters. In the following we briefly review XCS in its most recent version (Wilson 1996). The original XCS description can be found in (Wilson 1995) or in Kovacs's report (Kovacs 1996) where some original results are duplicated and extended for more complex environments.

Classifier Parameters. 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.

Performance Component. 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 randomly 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.

Reinforcement Component. 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 in the following order: first the prediction, then the prediction error, and finally the fitness.

First, the maximum system prediction is discounted by a factor γ ($0 \leq \gamma < 1$) and added to the reward returned in the previous time step. The resulting quantity, simply named P, is used to update the prediction p_j by the Widrow-Hoff delta rule (Widrow and Hoff 1960) with learning rate β ($0 < \beta \leq 1$): $p_j \leftarrow p_j + \beta(P - p_j)$. Then the prediction error ε_j is adjusted using the delta rule technique: $\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$. 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 $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

Covering. Covering acts when the match set [M] is empty or the system is stuck in a loop. In both cases, a classifier, created with a condition that matches the system inputs and a random action, is inserted in the population while another one is deleted from the population. The situation in which the system is stuck in a loop is detectable because the predictions of the classifiers involved start to diminish steadily. To detect this phenomenon when [M] is created the system checks whether the total prediction of [M] is less than ϕ times the average prediction of the classifiers in the population.

Genetic Algorithm. 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.

Subsumption Deletion. 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.

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. Experimental results reported in (Kovacs 1996) evidence that macroclassifiers do not affect the population of microclassifiers since every procedure is written to take into account the numerosity parameter. The number of macroclassifiers is a useful statistic to measure the degree of generalization obtained by the system. In fact, as XCS converges to a population of accurate and maximally general classifiers, the number of macroclassifiers decreases while the number of microclassifiers is kept constant by the delete/insert procedures.

3 DESIGN OF EXPERIMENTS

Experiments with XCS were conducted in the well-known "woods" environments following the methodology employed in (Wilson 1995). In the next we give a brief overview of the woods environments. Then the design of experiments discussed in the rest of the paper will be introduced.

The Woods Environments. Woods environments are grid worlds in which each cell can be empty, "." symbol, can contain a rock, "`Q`" or "`O`" symbols, or otherwise food, "`F`" or "`G`" symbols. 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, which result in a binary string of 16 digits for environments, with only one type of food and rock, or 24 digits for Environments with two types of food cells and rocks. The animat can decide to move in any of the adjacent cells. If the destination cell is blank then the animat moves; if the cell contains food the animat moves and eats the food, while if the destination cell contains a rock the move does not take place.

```
QQQQQQQQ
Q..Q..FQ
QQ..Q..Q
QQ.Q..QQ
Q......Q
QQ.Q...Q
Q*...Q.Q
QQQQQQQQ
```

Figure 1: The Maze4 Environment.

Experiments. 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 XCS until it enters a food cell, eats the food, and receives a reward equal to 1000. The food immediately re-grows and a new problem begins. We employed the same exploration/exploitation strategy reported in the original XCS papers (Wilson 1995; Wilson 1996). Before a new problem begins XCS randomly decides whether the problem is going to be solved in "pure exploration" or "pure exploitation" with probability 0.5. When in exploration mode, the system selects actions randomly with a probability proportional to their predicted reward[1]. When in exploitation, XCS selects the action with the highest predicted reward. This strategy is simply referred to as 50/50 exploration/exploitation strategy (Wilson 1995).

Two types of statistics are collected for each environment: the performance and the population size in macroclassifiers. Performance is computed as the average number of steps to food in the last 50 exploitation problems. Each statistics presented in this paper is averaged on ten experiments.

4 XCS IN THE MAZE4 ENVIRONMENT

Results proposed in literature for the XCS classifier system in the woods environments are limited to very regular and periodic environments that is, built repeating a certain pattern indefinitely in the horizontal and vertical directions, such as in Woods1 and

[1] XCS as proposed in (Wilson 1995) selects actions randomly when in exploration mode. Our experiments show no significant difference between the two criteria.

Woods2. Thus, after having duplicated Wilson's results, we experimented the system on a series of non periodic environments. For this purpose we used a number of mazes of increasing complexity based on the woods environments. Starting from trivial mazes (5×5 grids) we increased the complexity of environments uniformly. Here we report the results obtained for XCS in the simple maze, named Maze4, shown in Figure 1. Maze4 is a small maze consisting of 26 distinct free cells, which contains only one type of rock (`Q`) and a unique food cell (`F`). Maze4 was specifically designed to be very different from other environments such as Woods2. First, Maze4 does not have any regularities, consequently each sensory-action pair almost requires one classifier. Second, many sensory configurations that the system receives differ only for few bits and thus is very likely for system to produce overgeneral classifiers. Thus Maze4 does not allow much generalization as other traditional environments do.

Figure 2 shows the performance of the system computed as the average number of steps to food in the last 50 exploitation problems. The XCS system parameters were set as by Wilson for the Woods2 environment in (Wilson 1995): $N = 800$, β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_{\#}$=0.5, P_I=10.0, ε_I=0.0, F_I=10.0 [2].

As it can be noticed, XCS fails to learn an optimal path to food. Experiments, not reported here, evidence that the population size must be set to 1600 classifiers before the system reaches optimal performance, although the Maze4 environment is very simple (it consists only of 26 distinct sensory configurations).

5 ANALYSIS OF THE RESULTS IN THE MAZE4 ENVIRONMENT

To understand the resulting performance of XCS in the Maze4 environment we first removed the generalization mechanism from the system by not allowing don't care symbol in classifiers. Experimental results with this bare version of XCS, not reported, showed that XCS could easily learn how to reach food in the Maze4 environment with 400 classifiers. This result lead us to formulate the hypothesis that the generalization mechanism of XCS needs a very large population to converge in those environments which allow almost no generalization. Unfortunately, the larger the population, the more the time to learn, and therefore the larger the number of problems to solve before the population can converge to a small set of maximally general classifiers. Thus we experimented a modification to the standard XCS that, reducing the pressure towards generalization, could have, on average, better performances with smaller populations on different types of environments.

[2]Some of these parameters have not been presented in the XCS overview but are reported here for completeness. We refer the reader to (Wilson 1995) for a complete discussion of those parameters.

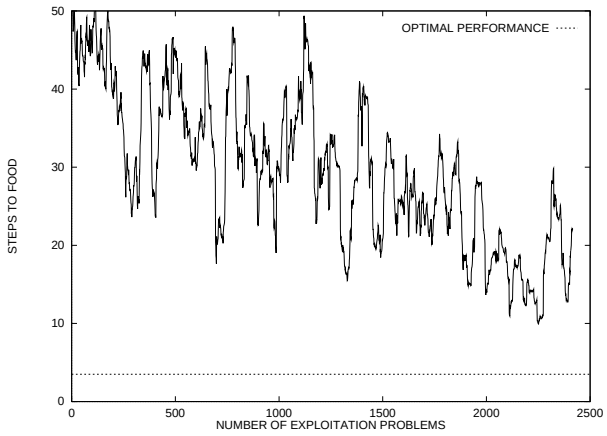

Figure 2: Performance of the XCS system in the Maze4 environment with 800 classifiers. Optimal performance is represented by the horizontal dashed line. The curve is averaged on ten experiments.

We first experimented with a version of XCS in which the tendency to generalize was reduced. This was obtained as follows: (i) the GA acted in the match set as in the first Wilson's proposal (Wilson 1995); (ii) classifiers were selected for reproduction with probability proportional to the product of the prediction and the fitness ($p_j \times F_j$) instead of the fitness alone; (iii) during exploration problems actions were randomly selected using the Boltzmann distribution. The underlying idea was to reduce the generalization pressure, by means of point (i) above, and to diminish the exploration of less rewarded condition/action pairs, by means of points (ii) and (iii). Results, presented in Figure 3, show that this *reduced* version of XCS learns to reach food in a few trials but, as Figure 4 confirms, has a very reduced tendency to generalization. This version of XCS, in fact, tended to keep a population of macroclassifiers almost as large as the population of microclassifiers, which indicates that generalization is not operating.

6 POSSIBLE SOLUTIONS

We consider the reduced version of XCS proposed in the previous section an inacceptable solution to the difficulty that XCS has to learn in environments which,

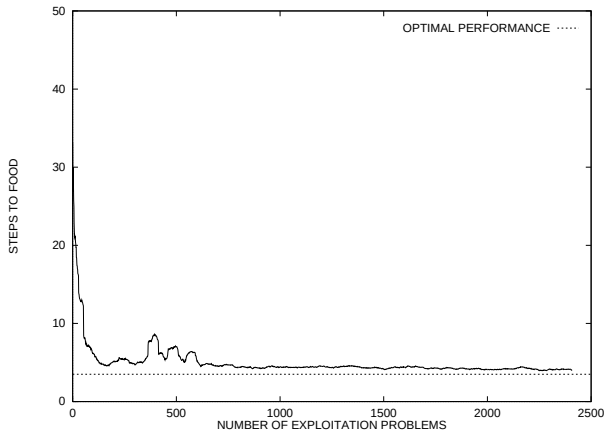

Figure 3: Performance for the reduced version of XCS in the Maze4 environment with 800 classifiers. Optimal performance is represented by the horizontal dashed line. The curve is averaged on ten experiments.

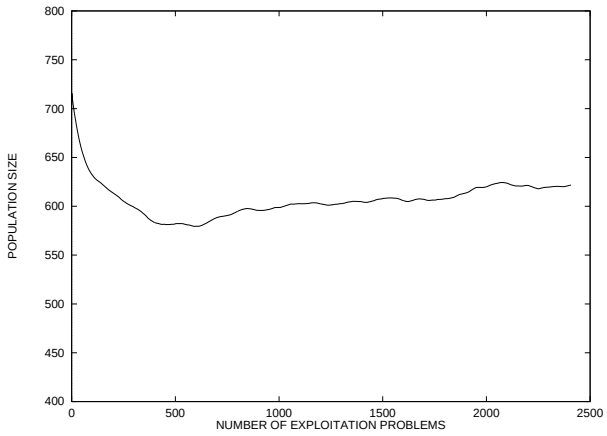

Figure 4: Population size in macroclassifiers for the reduced version of XCS in the Maze4 environment with 800 classifiers. The curve is averaged on ten experiments.

like Maze4, allow only a few generalization. First, generalization, one of the most interesting features in XCS, is almost eliminated. Second, the reduced version of XCS gives a *global* solution to the problem. Generalization is, in fact, reduced on the whole environment. This is not desirable since many environments could have zones in which generalization is desirable and areas where generalization is impossible. A *local* solution to the problem evidenced in the previous section, would be more desirable.

Classifier systems introduce generalization by means of the don't care symbols (#) in the condition part of the classifiers. Specifically, the presence of a # indicates that the classifier condition can match sensory inputs in which the bit corresponding to the # is either a 1 or a 0. In XCS, # symbols are introduced, in three places:

- in the initial population, where an allele is set to # with probability $P_{\#}$;
- in covering, when a new classifier that matches the input condition is created, don't care symbols are randomly introduced;
- in the mutation, when the alleles of a classifier are randomly changed.

The first two cases can be regarded as *initial conditions* of the system, while mutation is a main component of generalization in XCS. Thus we devised a mechanism to contrast mutation in situations that do not allow much generalization. In literature Dorigo (Dorigo 1993) already presented an operator, named *Mutespec*, for this kind of problems.

6.1 THE MUTESPEC OPERATOR

Dorigo (Dorigo 1993) presents the Mutespec operator to reduce reward variance in *oscillating classifiers*. These are classifiers that, due to the presence of some don't care symbols, match different conditions with different rewards. Consequently they are, on average, activated too often in situations in which they are not useful and too seldom in situations in which they would be useful. Dorigo used the Mutespec operator to specialize an oscillating classifier into two offspring classifiers: When the reward variance of a classifier C_1 is K times greater than the average of the reward variance in the population then Mutespec is applied. Mutespec selects the classifier with the largest reward variance in the population. Then it generates two offspring classifiers from the original one replacing a randomly selected # symbol respectively with the 0 and 1 symbols.

Comparing the way in which the operator proposed by Dorigo is applied, with other operators in the XCS system it is worth noting that:

- Mutespec operates on the whole population acting *panmictically* while the genetic operators in XCS are executed in niches, i.e. in the action set [A];
- Mutespec introduces a new statistics, the reward variance, that should be treated differently from the other classifier parameter in XCS;
- Mutespec selects a classifier deterministically, in fact, it is always applied to the classifier with the highest variance value.

6.2 THE SPECIFY OPERATOR

We propose *Specify*, a specification operator which replaces don't care symbols in the classifiers with a criterion different from Mutespec. First, Specify acts in the action set, as the other genetic operators in XCS do, and uses the prediction error ε_j to detect oscillating classifiers and to select them. Finally Specify, as the GA in XCS, is executed only under certain conditions.

Specify works as follows. At each cycle the average prediction error $\varepsilon_{[A]}$ in action set [A] is compared to the average prediction error $\varepsilon_{[P]}$ in the population [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. Finally, 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.

Specify is thus a sort of "covering" operator in that it tries to correct an oscillating classifiers using the raw sensory configuration. Parameters for the new *specified* classifier are initialized as the offspring classifiers in the GA (Wilson 1995; Kovacs 1996). In the following, XCS with the *Specify* operator will be called XCSS.

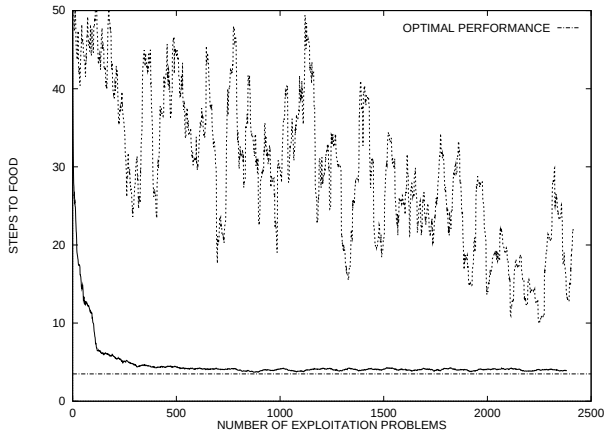

Figure 5: Performance for the XCSS, solid line, and XCS, dashed line, classifier systems in the Maze4 environment. Optimal performance is represented by the horizontal dashed line. Curves are averaged on ten experiments.

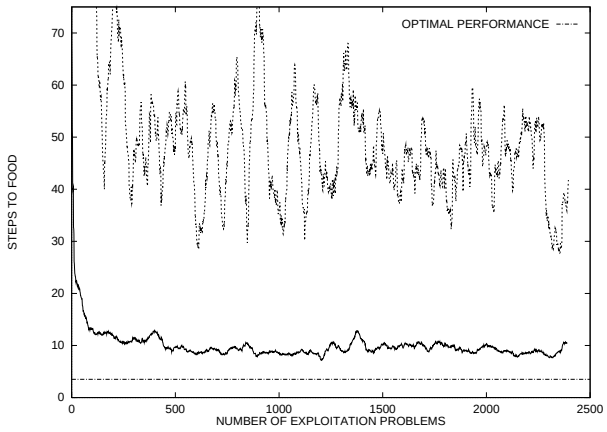

Figure 6: Performance for the XCSS, solid line, and XCS, dashed line, classifier systems on the Maze4 environment for a population of 400 classifiers. Optimal performance is represented by the horizontal dashed line. Curves are averaged on ten experiments.

7 XCSS IN THE MAZE4 ENVIRONMENT

The first experiment employs XCSS to learn how to reach food in the Maze4 problem with a population of 800 classifiers. The parameters for the experiments are set as for the previous experiment on the XCS with the same environment while parameters for the *Specify* operator are set as follows:

- Specify acts when the classifiers in the action set have been updated on average at least 20 times (N_{Sp}=20);
- Each # symbol in the selected classifier is replaced by the corresponding sensory input with probability 0.5 (P_{Sp}=0.5).

Figure 5 compares the performance of XCSS, solid line, with that of XCS, dashed line, in the Maze4 environment. The curves evidence that XCSS rapidly learns how to reach food in the maze while XCS performance is very unstable. A second experiment tests the robustness of the Specify operator against the population size. The hypothesis we test is that the Specify operator, contrasting the generalization mechanism, determines a better performance even with smaller populations. XCSS and XCS are applied, with the same parameter settings of the previous experiment, to Maze4 with a population of 400 classifiers. Our hypothesis is that in XCS the pressure to generalization leads to worse performance but that in XCSS the worsening is reduced by the presence of the specification operator.

Results for the performance of the two systems, shown in Figure 6, confirm our hypothesis. As the population size diminishes, the generalization mechanism in XCS prevents the system from learning the shortest path to food. On the contrary, in XCSS, the Specify operator contrasts the tendency to generalization and the system reaches a good solution almost immediately. It is worth noting the oscillating performance of XCSS, that evidences the presence of the contrasting generalization/specification operators.

8 XCSS IN THE WOODS2 ENVIRONMENT

The Woods2 environment, shown in Figure 7, has been introduced by Wilson in (Wilson 1995) to study the generalization mechanism in XCS. It contains two types of food cells (`G` and `F`) and two types of rocks (`Q` and `O`). The right and left edges of the grid are connected and so are the top and bottom edges.

```
.............................
.QQF..QQF..OQF..QQG..OQG..OQF.
.OOO..QOO..OQO..OOQ..QQO..QQQ.
.OOQ..OQQ..OQQ..QQO..OOO..QQO.
.............................
.............................
.QOF..QOG..QOF..OOF..OOG..QOG.
.QQO..QOO..OOO*.OQO..QQO..QOO.
.QQQ..OOO..OQO..QOQ..QOQ..OQO.
.............................
.............................
.QOG..QOF..OOG..OQF..OOG..OOF.
.OOQ..OQQ..QQO..OQQ..QQO..OQQ.
.QQO..OOO..OQO..OOQ..OQQ..QQQ.
.............................
```

Figure 7: The Woods2 Environment with animat ("`*`"). Empty cells are indicated by "`.`". Two types of rocks ("`O`" and "`Q`"). Two types of food ("`F`" and "`G`").

We apply XCSS and XCS to this environment and compare the results. The goal is to evaluate the generalization capabilities of XCSS with respect to XCS or equivalently how much generalization is lost when XCSS is used in environments which allow generalization. XCS parameters are set as in (Wilson 1996) for the same experiment: N = 800, β=0.2, γ=0.71, θ= 25, ε_0=0.01, α=0.1, χ=0.8, μ=0.01, δ=0.1, ϕ=0.5, $P_{\#}$=0.5, P_I=10.0, ε_I=0.0, F_I=10.0. Parameters for the Specify operator are set as for the Maze4 environment.

Figure 8 shows the performance of the two systems: XCSS, solid line, and XCS, dashed line. The two curves evidence that the systems reach the same performance and that XCSS converges a little bit faster than XCS. Analyzing the population size in macroclassifiers for the two systems, see Figure 9, it can be noticed that XCS has better generalization capabilities; nevertheless, the Specify operator slows the generalization process but does not eliminate it.

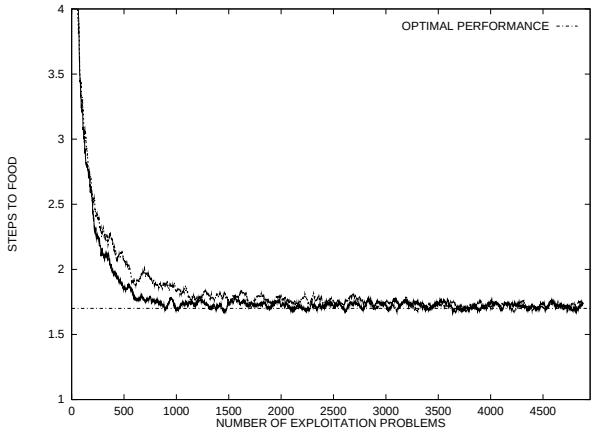

Figure 8: Performance in the Woods2 environment for XCSS, solid line, and XCS, dashed line. Optimal performance is represented by the horizontal dashed line. Curves are averaged on ten experiments.

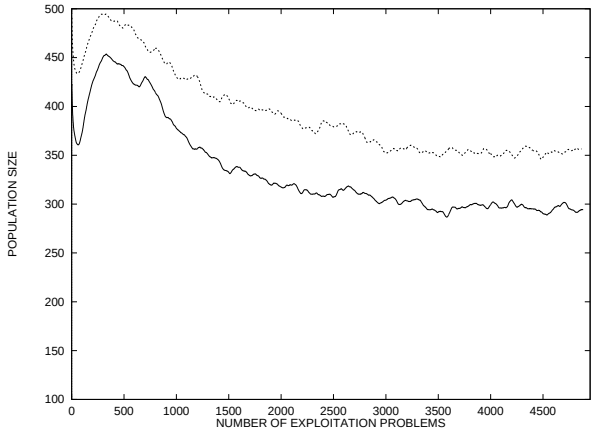

Figure 9: Population, in macroclassifiers, for XCSS, solid line, and XCS, dashed line, in the Woods2 environment. Curves are averaged on ten experiments.

9 CONCLUSIONS

This paper presents a modification of the XCS system to counterbalance the effects of its generalization mechanism in environments that allow only few generalizations. Experimental results have shown that the generalization mechanism of XCS can prevent the sys-

tem from learning simple tasks in such type of environments. Two solutions to the problem were discussed: a *global* solution and a *local* solution. In the former, the XCS architecture was modified to reduce the overall generalization mechanism of the system; unfortunately, this solution results in a classifier system with almost no generalization capabilities.

In the latter, a genetic operator, called *Specify*, counterbalances the effects of the generalization mechanism in environmental niches that do not allow much generalization. The proposed operator acts in environmental niches only when an unstable situation is detected and replaces some don't care symbols with the corresponding sensory input digits. Experimental results reported in the paper show that the proposed system can deal with a larger number of environments and, most important, is more robust than XCS with respect to the population size parameter.

Acknowledgments

I wish to thank Marco Colombetti and Marco Dorigo for the many interesting discussions and for the support in reviewing the last version of this paper. Many thanks also to Stewart Wilson and to two anonymous reviewers for their comments on an earlier version of this paper.

References

Dorigo, M. (1993). Genetic and non-genetic operators in ALECSYS. *Evolutionary Computation 1*(2), 151–164.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In M. Michalski, Carbonell (Ed.), *Machine Learning, Volume 2*, Chapter 20, pp. 593–623. San Mateo, CA: Morgan Kaufmann.

Kovacs, T. (1996). 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. Avaiable from the technical report archive at http://www/system/tech-reports/tr.html.

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

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

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

Wilson, S. W. (1996). Generalization in XCS. Unpublished contribution to the ICML '96 Workshop on Evolutionary Computing and Machine Learning. Avaiable at http://netq.rowland.org.