Empirical Analysis of Generalization and Learning in XCS with Gradient Descent

Pier Luca Lanzi[†‡], Martin V. Butz[⋆], David E. Goldberg[‡]
[†]Artificial Intelligence and Robotics Laboratory, Politecnico di Milano, I-20133, Milano, Italy
[⋆]Department of Cognitive Psychology, University of Würzburg, 97070 Würzburg, Germany
Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign, Urbana, IL 61801, USA
pierluca.lanzi@polimi.it, butz@psychologie.uni-wuerzburg.de, deg@uiuc.edu

ABSTRACT

We analyze generalization and learning in XCS with gradient descent. At first, we show that the addition of gradient in XCS may slow down learning because it indirectly decreases the learning rate. However, in contrast to what was suggested elsewhere, gradient descent has no effect on the achieved generalization. We also show that when gradient descent is combined with roulette wheel selection, which is known to be sensitive to small values of the learning rate, the learning speed can slow down dramatically. Previous results reported no difference in the performance of XCS with gradient descent when roulette wheel selection or tournament selection were used. In contrast, we suggest that gradient descent should always be combined with tournament selection, which is not sensitive to the value of the learning rate. When gradient descent is used in combination with tournament selection, the results show that (i) the slowdown in learning is limited and (ii) the generalization capabilities of XCS are not affected.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics Based Machine Learning, Learning Classifier Systems

General Terms

Algorithms, Performance, Generalization.

Keywords

LCS, XCS, RL, Gradient Descent, Generalization.

1. INTRODUCTION

Learning classifier systems combine temporal difference learning and genetic algorithms to solve problems online. They have been successfully applied to several domains [19, 5, 3, 2]. However, their application in sequential decision making tasks is usually restricted to small problems [1, 15], though some successes in complex tasks have been reported [18]. Recently, Butz et al. [6, 8] extended XCS with gradient descent to improve its performance in sequential decision making tasks. Their results show that the addition of gradient descent improves the learning performance of XCS in small though challenging sequential decision making problems [1, 15, 8, 7]. This improvement was explained by noting that the gradient acts as an adaptive learning rate that slows down the update of classifier prediction in low fitness classifiers [6, 8]. As a consequence, the prediction of inaccurate classifiers, which usually oscillates (see e.g. [11]), as well as the classifier prediction error can be estimated more accurately, increasing their chance of being deleted. Drugowitsch and Barry [13], however, suggested a different explanation: they argued that the reported improvements [6, 8] may be mainly due to an additional support of overspecific classifiers, which will result in very little generalization [13, pag. 20]. So far, however, empirical validation was presented neither in [6, 8] nor in [13]. Thus, it is still unclear, if gradient descent has any effect on XCS's generalization capabilities.

The purpose of this paper is to investigate the effect of gradient descent on the generalization and learning capabilities of XCS in a principled manner. The analysis in [6, 8] focused on sequential decision making problems [15, 1], which allow little generalization. Since our focus is on *generalization* and learning, in this work we considered simpler problems that allow many generalizations: the Boolean multiplexer and the `Woods2` environment [25]. Initially, we compare XCS with and without gradient descent on the Boolean multiplexer. We show that XCS with gradient learning is slower because the gradient actually decreases the learning rate. We also show that when gradient descent and roulette wheel selection are used together, such a slowdown can be dramatic in larger problems. We argue that this effect is not surprising and it is basically due to a combination of two effects. First, in large problems gradient descent can result in very small learning rates. Second, when roulette wheel selection is used with small learning rates, the performance of XCS may be impaired–as Butz et al. [9] already showed. However, tournament selection [9] does not suffer from the same shortcoming. In fact, when we repeat the same set of experiments with tournament selection we find that (i) when gradient descent is used in combination with tournament selection the slowdown in learning is limited and (ii) gradient descent has still no effect on the generalization capabilities of XCS. Overall, our results suggest that although

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

gradient descent and roulette wheel selection can work well together in sequential decision making tasks (e.g., [8] and here in Section 5.2), overall the results suggest that gradient descent should be usually combined with tournament selection. This extends the previous analysis in [6] where it was noted that tournament selection did not provide any advantage over roulette wheel selection on the set of problems considered.

2. REINFORCEMENT LEARNING

In reinforcement learning an *agent* learns to perform a task through *trial and error interaction* with an unknown *environment* [21]. At time t, the agent is in state s_t and performs an action a_t in the environment. Depending on the state s_t, on the action a_t performed, and on the effect that a_t has in the environment, the agent receives a *scalar reward* r_{t+1} and a new state s_{t+1}. The agent's goal is to *maximize* the discounted long term reward it receives from the environment, often termed the *return* [21]. The agent achieves this by learning an action-value function $Q(s_t, a_t)$ that maps state-action pairs into the corresponding expected payoff. For instance, Q-learning [21] starts from a random $Q(\cdot, \cdot)$ and, at time t, it updates the current payoff estimate $Q(s_t, a_t)$ as,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta(\hat{Q}(s_t, a_t) - Q(s_t, a_t)), \quad (1)$$

where, β is the *learning rate* ($0 \leq \beta \leq 1$) and $\hat{Q}(s_t, a_t)$ is the new estimate of $Q(s_t, a_t)$ computed based on the current experience as "$r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a)$" with a *discount factor* γ ($\gamma \in [0, 1)$). In large problems, $Q(s, a)$ is usually approximated by a function f parametrized with a vector $\boldsymbol{\theta}$, i.e.,

$$Q(s, a) = f(\boldsymbol{\phi}(s, a), \boldsymbol{\theta}), \quad (2)$$

where $\boldsymbol{\phi}(\cdot)$ is an input mapping function that translates the state-action space to a feature space [20, 21]. The problem of learning $Q(s, a)$ thus translates into the problem of estimating the parameter vector $\boldsymbol{\theta}$ which at time t minimizes a given error function E_t. This is usually solved with methods of gradient descent. At time t, the parameter vector $\boldsymbol{\theta}$ is modified following, with step β_t, the *direction* that reduces the error E_t in f, i.e.,

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \beta_t \frac{\partial E_t}{\partial \boldsymbol{\theta}}. \quad (3)$$

Linear methods are probably the most important class of approximators used in reinforcement learning [4, 21]. They assume that the feature vector $\boldsymbol{\phi}(s, a)$ and the parameter vector $\boldsymbol{\theta}$ are of the same size so that the approximated value of $Q(s, a)$ is simply computed by the inner product, $f(\boldsymbol{\phi}(s, a), \boldsymbol{\theta}) = \phi(s, a)'\boldsymbol{\theta}$. *Linear averagers* [14] are a type of linear approximators that minimize the error E_t defined as,

$$E_t = \frac{1}{2} \sum_i \left(\hat{Q}(s_t, a_t) - \theta_i\right)^2 \phi_i(s_t, a_t) \quad (4)$$

where θ_i and $\phi_i(s_t, a_t)$ are the i-th component of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}(s_t, a_t)$; this leads to the update:

$$\theta_i \leftarrow \theta_i + \beta_i(\hat{Q}(s_t, a_t) - \theta_i))\phi_i(s_t, a_t), \quad (5)$$

which minimizes the difference between the new estimate $\hat{Q}(s_t, a_t)$ and the parameter θ_i.

3. THE XCS CLASSIFIER SYSTEM

XCS is a reinforcement learning algorithm that works on a rule based representation [16]. It maintains a population of rules (the classifiers), which represents the current solution to a reinforcement learning problem. Classifiers consist of a condition, an action, and four main parameters [24, 10]: (i) the prediction p, which estimates the relative payoff that the system expects when the classifier is used; (ii) the prediction error ε, which estimates the error of the prediction p; (iii) the fitness F, which estimates the accuracy of the payoff prediction given by p; and (iv) the numerosity *num*, which indicates how many copies of classifiers with the same condition and the same action are present in the population.

At time t, XCS builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input s_t; if [M] contains less than θ_{mna} actions, *covering* takes place and creates a new classifier that matches s_t and has a random action. For each possible action a in [M], XCS computes the *system prediction* $P(s_t, a)$, which estimates the payoff that the XCS expects if action a is performed in s_t. The system prediction $P(s_t, a)$ is computed as the fitness weighted average of the predictions of classifiers in [M] that advocate action a:

$$P(s_t, a) = \sum_{cl_k \in [M](a)} p_k \times \frac{F_k}{\sum_{cl_i \in [M](a)} F_i} \quad (6)$$

where, [M](a) represents the subset of classifiers of [M] with action a, p_k identifies the prediction of classifier cl_k, and F_k identifies the fitness of classifier cl_k. Next, XCS selects an action to perform; the classifiers in [M] that advocate the selected action form the current *action set* [A]. The selected action a_t is performed, and a scalar reward r_{t+1} is returned to XCS together with a new input s_{t+1}. When the reward r_{t+1} is received, the estimated payoff $P(t)$ is computed as follows:

$$P(t) = r_{t+1} + \gamma \max_{a \in [M]} P(s_{t+1}, a) \quad (7)$$

Next, the parameters of the classifiers in [A] are updated in the following order [10]: prediction, prediction error, and finally fitness. Prediction p is updated with learning rate β ($0 \leq \beta \leq 1$):

$$p_k \leftarrow p_k + \beta(P(t) - p_k) \quad (8)$$

Then, the prediction error ϵ and classifier fitness are updated as usual [24, 10]. On a regular basis (dependent on parameter θ_{ga}), the genetic algorithm is applied to classifiers in [A]. It selects two classifiers, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. The resulting offspring classifiers are inserted into the population and two classifiers are deleted to keep the population size constant.

4. GRADIENT DESCENT IN XCS

The relation between reinforcement learning and XCS has been widely investigated in the literature [16, 6, 23, 22, 12, 13]. An initial analysis in [16] showed that (i) the system prediction $P(s_t, a)$ in Equation 6 actually represents the value of $Q(s_t, a)$ using the classifiers in [A]; while (ii) the prediction value of $P(t)$ in Equation 7 actually represents the new estimate $\hat{Q}(s, a)$ in Equation 1. The comparison

has been extended by Butz el al. [6] to the case of reinforcement learning with value function approximation. In [6], it is noted that (i) the classifier predictions in XCS corresponds to the parameter in $\boldsymbol{\theta}$ in Equation 2; (ii) although XCS implements generalized Q-learning, its prediction update (Equation 8) does not include the gradient term; (iii) the gradient term for a classifier cl_k in the action set [A] is computed as,

$$\frac{\partial P(s_t, a_t)}{\partial cl_k} = \frac{F_k}{\sum_{[A]} F_i}, \quad (9)$$

so that the prediction update for XCS with gradient descent for classifier cl_kbecomes,

$$p_k \leftarrow p_k + \beta(P(t) - p_k)\frac{F_k}{\sum_{[A]} F_i}. \quad (10)$$

The results reported in [6, 8] show that the additional gradient term improves XCS performance over a set of multi-step problems. Such an improvement is explained by noting that in XCS the gradient acts as an adaptive learning rate that slows down the update of classifier prediction in low fitness classifiers [6, 8]. Butz et al. [8] argue that the gradient improves the estimation of the classifier prediction in inaccurate classifiers. Thus, the prediction and the prediction error of inaccurate classifiers can be estimated more accurately, increasing their chance of being deleted and decreasing the problem of misleading reward propagation. Later, Wada et al. [23] showed that the derivation in [6] was not correct if "standard" reinforcement learning was considered [20]. Nevertheless, Lanzi and Loiacono [17] also showed that the derivation in [8] is consistent to "averaging" reinforcement learning and that the gradient descent update in Equation 10 is coherent with the update in Equation 5.

Recently, Drugowitsch and Barry [12, 13] developed a theoretical framework to study the properties of accuracy-based learning classifier systems. The framework models the main non-evolutionary components of a typical learning classifier system, that is, function approximation, reinforcement learning, and classifier replacement. When discussing gradient descent in XCS [13, pag. 20], they argue that the improvements that gradient descent produces in XCS may be mainly due to an additional support to overspecific classifiers, which will eventually result in very little generalization. However, no theoretical nor empirical validation was included in [13] to support this claim. Therefore the purpose of this paper is to investigate the effect of gradient descent in XCS on the generalization and learning capabilities of XCS in a principled manner.

5. THE EFFECT OF GRADIENT DESCENT

The problems considered in the previous study [8] are challenging for XCS in that they require relatively long action sequences but allow few generalizations [15, 1]. To analyze the effect of gradient descent on the generalization and learning capabilities of XCS, we now consider the slightly simpler Boolean multiplexer and `Woods2` environments, which allow many generalizations.

5.1 Boolean Multiplexer

In the first set of experiments, we apply XCS with gradient (XCSG) and XCS without gradient to the Boolean multiplexer. These functions are defined for strings of l bits where $l = k + 2^k$. The first k bits, $x_0, \ldots, x_{k-1}$, represent an address, which indexes the remaining 2^k bits, $y_0, \ldots, y_{2^k-1}$. The function returns the value of the indexed bit. For instance, in the 6-multiplexer, mp_6, we have that $mp_6(100010) = 1$ while $mp_6(000111) = 0$. More formally, the 6-multiplexer can be represented by the following disjunctive normal form:

$$mp_6(x_0, x_1, y_0, \ldots, y_3) = \overline{x_0}\,\overline{x_1}y_0 + \overline{x_0}\,x_1y_1 + x_0\,\overline{x_1}y_2 + x_0x_1y_3$$

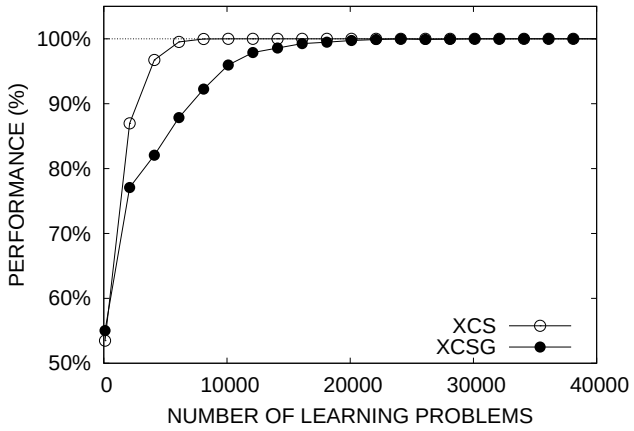
We initially applied XCS and XCSG to the 11-multiplexer with the following parameter setting: $N = 1000$; $P_\# = 0.3$; $\beta = 0.2$; $\alpha = 0.1$; $\epsilon_0 = 10$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$; $\theta_{GA} = 0$; $\delta = 0.1$; GA-subsumption is on with $\theta_{GAsub} = 20$; while action-set subsumption is off; when the correct action is performed, the system receives a 1000 reward, 0 otherwise. The experiment has been designed according to the standard approach used in previous works [24].

Figure 1a compares the predictive accuracy of XCS with gradient (XCSG), solid dots, and without gradient, empty dots, in the 11-multiplexer. XCS learns faster than XCSG (Figure 1a). This result is not surprising: the gradient term $F_k/\Sigma_{[A]}F_i$ in Equation 9 ranges between 0 and 1. Thus, from a mathematical viewpoint, it actually reduces the effect of the learning rate β in the prediction update (Equation 10). Accordingly, it should be expected that with gradient the learning may be slower, as it is reported in Figure 1a. XCS also generalizes faster (Figure 1b). However at the end XCS and XCSG converge to solutions that contain the same number of accurate maximally general classifiers. In fact they reach approximately the same population size. It is interesting to analyze how the specificity of classifiers in the action sets varies through the evolution.[1] Figure 2 compares the average specificity of classifiers in XCS and XCSG. Both systems start from the same average specificity, determined by the value $P_\#$. Then, the average specificity decreases faster in XCSG than in XCS, indicating that the classifiers evolved by XCSG are initially more general than those evolved by XCS. Later, the average specificity becomes similar (the two plots cross) and at the end XCS and XCSG reach the same, optimal, average specificity (i.e., approximately 36%).

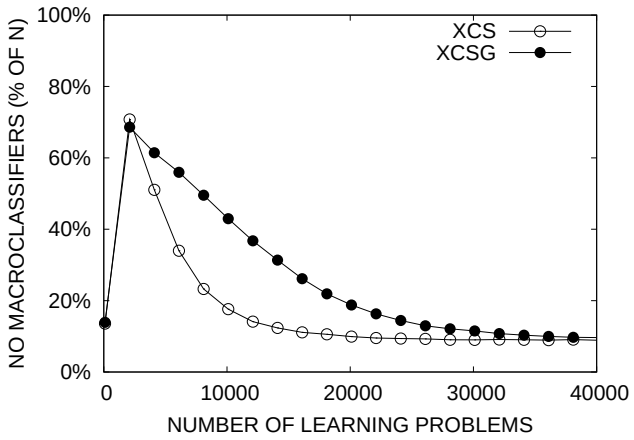
Butz et al. [9] showed that a small learning rate decreases the learning performance of XCS using roulette wheel selection. For instance, in the 20-multiplexer, when β is 0.2 XCS reliably reaches 100% performance by 100,000 learning steps. However, when β is 0.01, after 200,000 learning steps XCS performance is still around 60% [9, pag. 59]. Butz et al. [9] explain this by noting that the majority of classifiers in the initial population is over-general. Better offspring often loose against over-general parents since the fitness of the offspring classifiers increases slowly due to the small learning rate. In addition, the initially small differences between the fitness of accurate and overgeneral classifiers have small effects when using proportionate selection. Accordingly, Butz et al. [9] replaced the roulette wheel selection with tournament selection, showing that the latter was able to eliminate this sensitivity to the value of the learning rate.

As noted earlier, the gradient term $F_k/\Sigma_{[A]}F_i$ in the prediction update (Equation 8) can be viewed as an adaptive

[1]We remind the reader that classifier specificity is measured as the percentage of completely specified inputs in the condition.



(a)



(b)

Figure 1: XCS and XCSG with applied to 11-multiplexer: (a) predictive accuracy; (b) number of classifiers in the population.

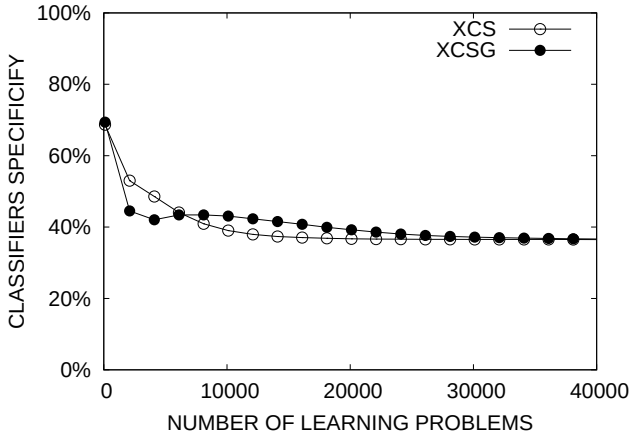


Figure 2: XCS and XCSG: average specificity of the classifiers in [A].

learning rate, which slows down the update of classifier prediction in low fitness classifiers [6, 8]. It is important to note that, although $F_k/\Sigma_{[A]}F_i$ ranges between 0 and 1, it can rarely be one. In fact, it is only one when there is only *one* accurate classifier in the action set [A] and either (i) there are no other classifiers in [A] or (ii) all the others classifiers in [A] are inaccurate. But this actually happens only when XCS has reached an optimal solution and it has also identified the best classifier for each problem subspace (i.e., for each action set). More often, action sets contain many different classifiers and the gradient is consequently much smaller than 1, drastically reducing the effect of learning rate β in the prediction update (Equation 10). For instance, if an action set contains all equally accurate or equally inaccurate classifiers, the gradient term for each classifier is around $1/|[A]|$. If the action set contains 32 equally accurate classifiers, the gradient is around 0.03, i.e., the gradient term reduces the effect of the learning by the 97% for all the classifiers in [A]. Accordingly, we can expect that in larger problems, in which the niches initially contain many classifiers, the gradient will consistently slow down the learning process. On the other hand, if such an effect is mainly due to the sensitivity of roulette wheel selection on the learning rate, tournament selection should solve the problem. Given XCS learns the accurate, maximally general classifiers to a problem, though, no particular effect on the generalization capabilities of XCS can be expected.

The second set of experiments, performed on the 20-multiplexer, confirms our hypotheses. Figure 3 compares the performance of XCS and XCSG on the 20-multiplexer with the standard parameter setting [25]: $N = 2000$, $\beta = 0.2$; $\alpha = 0.1$; $\epsilon_0 = 1$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$; $\theta_{GA} = 0$; $\delta = 0.1$; GA-subsumption is on with $\theta_{GAsub} = 20$; while action-set subsumption is off. As we expected, XCS rapidly reaches 100% performance while after 200,000 learning steps the performance of XCSG is still around 60%. This result is coherent with the findings in [9]: to solve the 20-multiplexer XCS has to keep 64 separate niches [8]; with 2000 classifiers, an action set will contain an average of 31.25 classifiers. Given that all classifiers in the action set are distinct, the gradient component for each classifier can be as low as 0.03. Combined with a learning rate β of 0.2, the actual learning rate is approximately 0.01, which corresponds to the rate used in [9]).

We repeated the same experiment using tournament selection instead of roulette wheel selection. The results in Figure 4a confirm what we expected: tournament selection is not sensitive to the scaling effect on the learning rate. Accordingly, both XCS and XCSG reach optimal performance fast. Note, however, that the small learning rate still affects the parameter update so that XCS converges faster than XCSG. However, the slower learning does not hinder the generalization capabilities of XCSG so that XCS and XCSG reach the same generalization level (Figure 4b).

5.2 Woods2

To evaluate the generalization capabilities also in multi-step problems, we applied XCS with and without gradient descent to the `Woods2` environment, depicted in Figure 5. This problem allows many generalizations [25]. `Woods2` is a grid world with two types of obstacles (represented by "`O`" and "`Q`" symbols), goal positions (represented by "`F`" and "`G`" symbols), and empty positions (represented by "`.`"

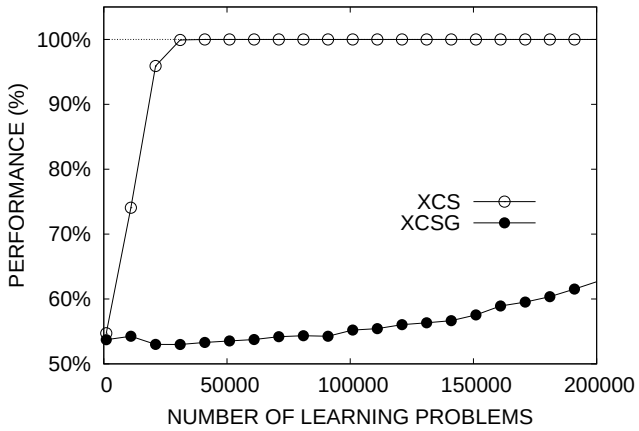


Figure 3: Performance of XCS and XCS with gradient (XCSG) in the 20-multiplexer when roulette wheel selection is used.

symbols). `Woods2` is a torus: its left and right edges are connected as well as its top and bottom edges. The agent can stay in any of the empty positions and it is able to move to any adjacent position that is empty. The agent has eight sensors, one for each adjacent position. Sensors are encoded by three bits coding features of the object. Thus, the agent's sensory input is a string with 24 bits (3 bits $\times$ 8 positions). There are eight possible actions, one for each possible adjacent position. The agent has to learn how to reach a goal position from any empty position. When the agent reaches a goal position (`F` or `G`) the problem ends and the agent receives a constant reward equal to 1000; otherwise it receives zero.

Initially, we applied XCS and XCSG to `Woods2` with the following parameter settings: $N = 1600$, $\beta = 0.2$; $\gamma = 0.71$; $\alpha = 0.1$; $\epsilon_0 = 10$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$; $\theta_{GA} = 25$; $\delta = 0.1$; GA-subsumption is on with $\theta_{GAsub} = 20$; while action-set subsumption is off. Figure 6a compares the performance of XCS and XCSG with roulette wheel selection. In this case, the performance is computed as the average number of steps to reach a goal over the last 100 test problems [25]. The results in Figure 6a are consistent with the previous findings. The introduction of the gradient in XCS slows down the learning process. Nonetheless, XCS and XCSG have the same generalization capabilities: at the end, both models evolve solutions of the same size (Figure 6b). This is confirmed by the analysis of the classifier specificity (Figure 7). Also in this case, the behavior is similar to what we observed in the Boolean multiplexer. At the very beginning the classifiers evolved by XCS with gradient descent are slightly more general, in fact their specificity is lower (Figure 7). Almost immediately XCS reaches the same generalization as XCSG (the two lines cross) and from then on, XCS works on more general classifiers than XCSG. At the end, both systems reach the same degree of generalization.

As we reported in the experiments with the Boolean multiplexer, also in this case tournament selection improves XCSG (Figure 8). However, in this case the improvement in learning speed is rather small compared to that reported for the Boolean multiplexer. This result is consistent to the findings in [6, 8]: in sequential decision making problems

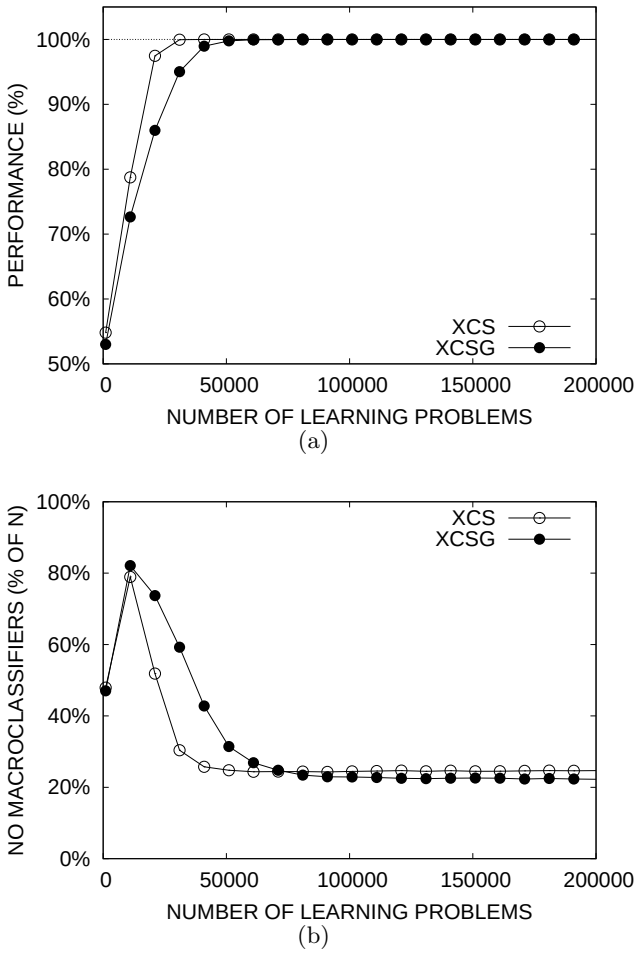


Figure 4: XCS and XCSG with tournament selection in the 20-multiplexer: (a) Performance; (b) Number of classifiers in the population.

the difference in learning speed between roulette wheel and tournament selection is limited. On the other hand, the two systems perform similar in terms of generalization as the curves for population size (Figure 8b) and those for average specificity show (Figure 9).

6. DISCUSSION

The results show that gradient descent slows down learning in XCS, but it has no effect on the generalization achieved. The reported slowdown is due to the addition of the gradient term $F_k/\Sigma_{[A]}F_i$ to the classifier prediction update equation, which decreases the effect of the prediction learning rate in Equation 10. To quantify the effect of gradient, the expected value of $F_k/\Sigma_{[A]}F_i$ should be computed. But this would require knowledge about the distribution of classifier fitness in the action set, which is unavailable in practice. We can trace the evolution of the average gradient of the classifiers in the action set to get a rough quantification of the effect that gradient has on the learning rate. Figure 10 reports the average gradient of the classifiers in the action sets for the experiment with XCSG using tournament selection on the 20-multiplexer, discussed in Sec-

```
.............................
.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.
.............................
```

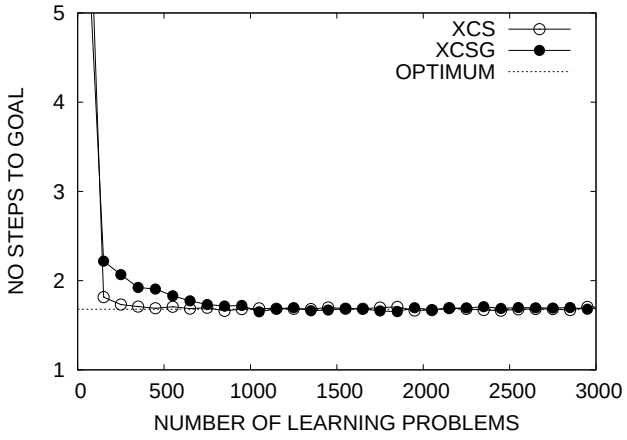
Figure 5: The `Woods2` Environment.

tion 5.1 (Figure 4). At the very beginning, the gradient term is large since the action sets contain few classifiers. Then, as more classifiers appear in the action sets, the gradient decreases: the average gradient is around 0.03, when 4000 learning steps were performed. Later, as accurate maximally general classifiers emerge in the population and take control in the action sets, the gradient increases until it stabilizes when the maximally accurate, maximally general classifiers dominate the action sets.

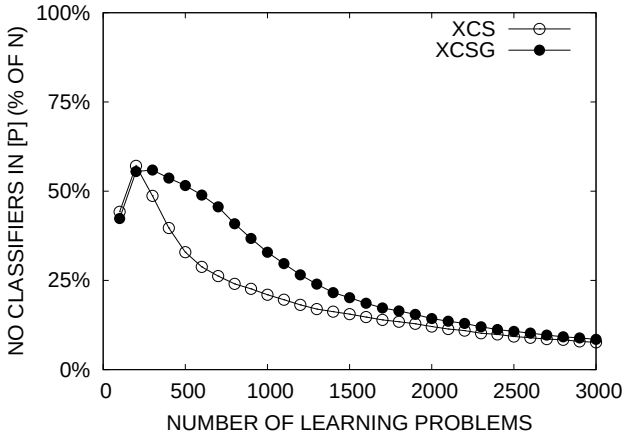
Since the gradient term in XCS decreases the effect of the learning rate in the prediction update (Equation 10), a higher learning rate for classifier prediction in Equation 10 should decrease the gap between the learning speed of XCS with and without gradient. Figure 11 compares the performance of XCS and XCSG using different learning rates for classifier prediction in Equation 10: 0.2, 0.6, and 1.0. Note that we did not increase the overall learning rate β but only the learning rate for the prediction update in Equation 10, named β_P from now on. XCS reaches the optimum faster than all the other versions of XCSG; when the prediction learning rate β_P is increased from 0.2 to 0.6 the learning speed of XCSG increases and XCSG reaches optimality almost at the same time as XCS, although the learning of XCSG is initially slower. When the prediction learning rate is increased further up to 1.0, the learning speed of XCSG shows a slight, though not statistically significant, improvement (analysis not reported). These results can be easily explained. At the beginning, when the accurate classifiers in the niches are specific, that is, before accurate maximally general classifiers appear, the gradient term is very small (around 0.03 after 4000 learning steps). Thus, even when the prediction learning rate β_P is set to 1.0, the combination of the gradient and the prediction learning rate $\beta+P$ is still smaller than the learning rate used with XCS in Equation 8 (that is, β=0.2).

7. CONCLUSIONS

We have analyzed generalization and learning in XCS with gradient descent. Our results show that the gradient slows down learning but has no effect on the generalization achieved. The experiments on `Woods2` show that this slowdown is rather limited: a result coherent with the previous results reported in [6, 8]. However, our experiments on simple functions show that the difference in learning speed can be dramatic when gradient descent with roulette wheel selection is used. We argued that this is due to a combination of two effects: the reduction of the learning rate that the gradient introduces and the sensitivity that roulette wheel has with respect to small values of the learning rate [9]. In fact, when tournament selection replaces roulette wheel selection, the slowdown caused by the introduction of gradient descent is limited. In contrast to what was suggested elsewhere [13], in none of the experiments performed the use of gradient descent had an effect on the generalization achieved: the final solutions evolved by XCS with gradient descent are as maximally general as those evolved by XCS both in terms of compactness and in terms of classifier generality. Overall, our analysis suggests that the combination of gradient descent and tournament selection can provide a good trade-off between robustness and learning speed.



(a)



(b)

Figure 6: XCS and XCSG with roulette wheel selection in `Woods2`: (a) average number of steps to reach a goal; (b) number of classifiers in the population.

Acknowledgments

This work was supported by the European commission contract no. FP6-511931. This work was also sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, the National Science Foundation under ITR grant DMR-03-25939

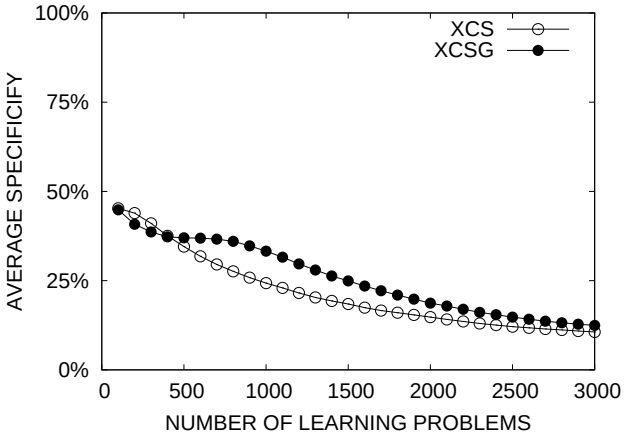


Figure 7: Average classifier specificity for XCS and XCSG in `Woods2` with roulette wheel selection.

at Materials Computation Center, UIUC. The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

8. REFERENCES

[1] Alwyn M. Barry. Limits in long path learning with XCS. In Springer-Verlag, editor, *Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1832–1843, Chicago, IL, 2003.

[2] Alwyn M. Barry, John H. Holmes, and Xavier Llorà. Data mining using learning classifier systems. In Bull [5], pages 15–67.

[3] Ester Bernadó, Xavier Llorà, and Josep M. Garrell. XCS and GALE: A comparative study of two learning classifier systems on data mining. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*, pages 115–132. Springer-Verlag, Berlin, 2002.

[4] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro et al., editors, *Advances in Neural Information Processing Systems 7*, pages 369–376, 1995. The MIT Press.

[5] Larry Bull, editor. *Applications of Learning Classifier Systems*. Studies in Fuzziness & Soft Computing. Springer-Verlag, 2004.

[6] Martin Butz, David G. Goldberg, and Pier Luca Lanzi. Gradient descent methods in learning classifier systems. Technical Report 2003028, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, January 2003.

[7] Martin V. Butz. *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*. Springer-Verlag, Berlin, 2006.

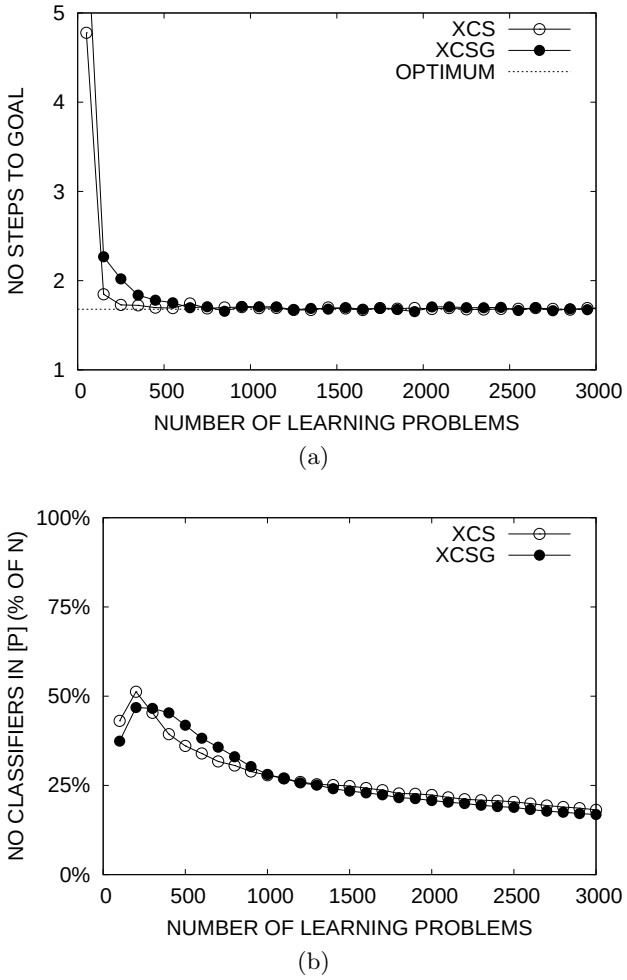


Figure 8: XCS and XCSG with tournament selection applied to `Woods2`: (a) average number of steps to reach a goal; (b) number of classifiers in the population.

[8] Martin V. Butz, David E. Goldberg, and Pier Luca Lanzi. Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. *IEEE Transaction on Evolutionary Computation*, 9(5):452–473, October 2005.

[9] Martin V. Butz, Kumara Sastry, and David E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6:53–77, 2005.

[10] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. *Journal of Soft Computing*, 6(3–4):144–153, 2002.

[11] Marco Dorigo. Genetic and non-genetic operators in alecsys. *Evolutionary Computation*, 1(2):151–164, 1993.

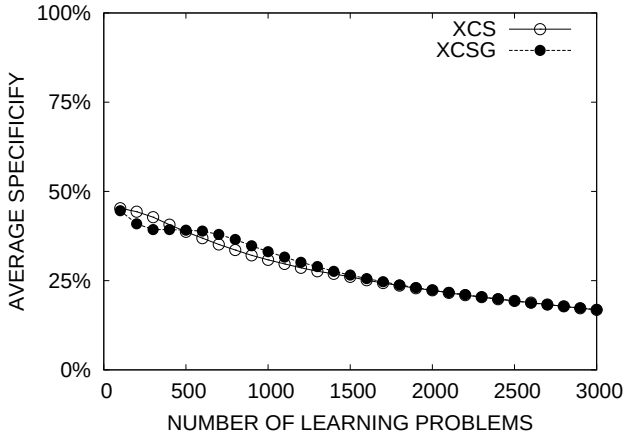


Figure 9: Average classifier specificity for XCS and XCSG in `Woods2` with tournament selection.

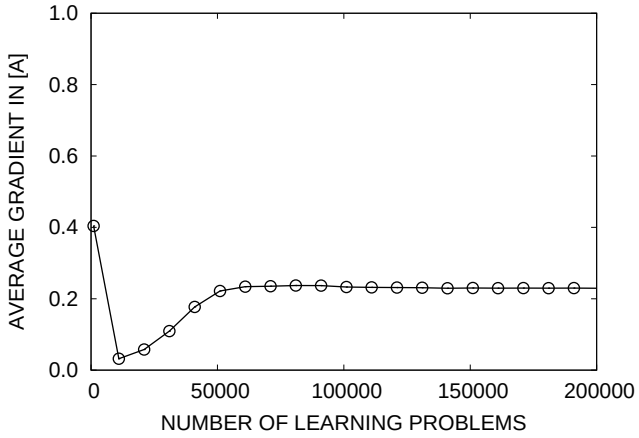


Figure 10: XCSG in the 20-multiplexer with tournament selection: average gradient for the classifiers in the action set.

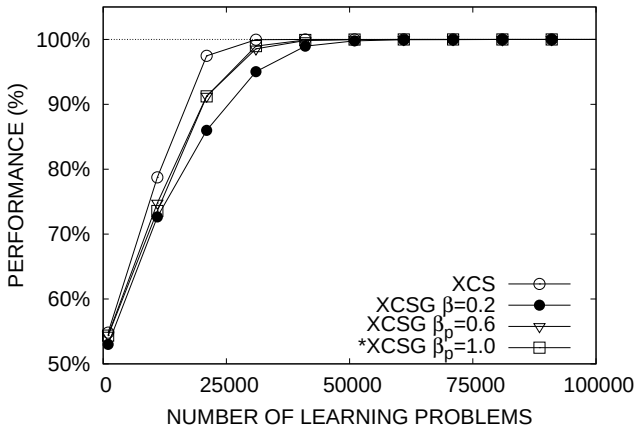


Figure 11: Average classifier specificity for XCS and XCSG in the 20-multiplexer with tournament selection and different values of learning rate.

[12] Jan Drugowitsch and Alwyn M. Barry. A formal framework and extensions for function approximation in learning classifier systems. Technical Report CSBU-2006-02, Department of Computer Science, University of Bath, January 2006.

[13] Jan Drugowitsch and Alwyn M. Barry. A formal framework for reinforcement learning with function approximation in learning classifier systems. Technical Report CSBU-2006-02, Department of Computer Science, University of Bath, January 2006.

[14] Geoffrey J. Gordon. Online fitted reinforcement learning from the value function approximation. Workshop on Value Function Approximation held during the 12th International Conference on Machine Learning, 1995.

[15] Pier Luca Lanzi. An Analysis of Generalization in the XCS Classifier System. *Evolutionary Computation Journal*, 7(2):125–149, 1999.

[16] Pier Luca Lanzi. Learning classifier systems from a reinforcement learning perspective. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 6(3):162–170, 2002.

[17] Pier Luca Lanzi and Daniele Loiacono. Standard and averaging reinforcement learning in XCS. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1489–1496, New York, NY, USA, 2006. ACM Press.

[18] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Classifier prediction based on tile coding. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1497–1504, New York, NY, USA, 2006. ACM Press.

[19] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *Lecture Notes in Computer Science*. Springer-Verlag, April 2000.

[20] Stuart Ian Reynolds. *Reinforcement Learning with Exploration*. PhD thesis, School of Computer Science. The University of Birmingham, Birmingham, B15 2TT, December 2002.

[21] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 1998.

[22] Atsushi Wada, Keiki Takadama, and Katsumori Shimohara. Learning classifier system equivalent with reinforcement learning with function approximation, 2005. Eighth International Workshop on Learning Classifier Systems (IWLCS-2005).

[23] Atsushi Wada, Keiki Takadama, Katsumori Shimohara, and Osamu Katai. Learning classifier systems with convergence and generalization. In Larry Bull and Tim Kovacs, editors, *Foundations of Learning Classifier Systems*, volume 183 of *Studies in Fuzziness and Soft Computing*, pages 285–304. Springer, 2005.

[24] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[25] Stewart W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998.