Computed Prediction in Binary Multistep Problems

Daniele Loiacono and Pier Luca Lanzi

Abstract— **Computed prediction was originally devised to tackle problems defined over real-valued domains. Recent experiments on Boolean functions showed that the concept of computed prediction extends beyond real values and it can also be applied to solve more typical classifier system benchmarks such as Boolean multiplexer and parity functions. So far however, no result has been presented for other well known classifier system benchmarks, i.e., binary multistep problems such as the woods environments. In this paper, we apply XCS with computed prediction to woods environments and show that computed prediction can also tackle this class of problems. Our results demonstrate that (i) XCS with computed prediction converges to optimality faster than XCS, (ii) it solves problems that may be too difficult for XCS and (iii) it evolves solutions that are more compact than those evolved by XCS.**

I. Introduction

LEARNING Classifier Systems are a genetic based machine learning technique for solving problems through the interaction with an unknown environment. They maintain a population of condition-action-prediction rules called classifiers which represents the current solution to the target problem. Each classifier represents a small portion of the overall solution. The condition identifies a part of the problem domain. The action represents a decision on the subproblem identified by the condition. The prediction provides an estimate of the classifier value in terms of problem solution. In learning classifier systems, the genetic component works on classifier conditions searching for an adequate decomposition of the target problem into a set of subproblems.

The XCS classifier system [16] is probably the most successful learning classifier system to date. It couples effective temporal difference learning, implemented as a modification of the well-known Q-learning [14], to a niched genetic algorithm guided by an accuracy based fitness to evolve accurate maximally general solutions. Recently, Wilson [19] extended XCS with the idea of *computed prediction* to improve the estimation of the classifiers prediction. In XCS with computed prediction, XCSF in brief, the classifier prediction is not memorized into a parameter but computed as a linear combination of the current input and a weight vector associated to each classifier. In [19] XCSF was initially applied to solve function approximation problems. Later [11], [10] it was applied to solve some multistep problems well known in the reinforcement learning literature and to learn Boolean functions [9]. However, so far XCSF has never be applied to the typical multistep problems studied in the learning classifier systems literature: the Woods problems [16]. This is probably due to the few generalization possibilities offered by the Woods problems, making of them a not so interesting domain for the analysis of XCSF. Nevertheless, a recent work [8] suggested that the computed prediction does not only affect the generalization capabilities of XCSF but it may also affect the convergence speed and the optimality of the solution reached by XCSF in multistep problems. In addition, XCSF has been recently extended with recursive least squares for updating the classifier prediction. Recursive least squares has shown promising results [12] but so far they have been applied only to function approximation problems (single-step problems). Thus, it is not currently clear whether the same results might be extended to mutlistep problems.

In this paper, we apply XCSF to well known binary multistep problems, namely the woods environments, and compare its performance to that of XCS. The results show that XCSF converges faster than XCS to the optimal solution and it solves problems that may be too difficult for XCS. In addition, we show that XCSF can be applied with GA-subsumption to evolve solutions more compact than the one evolved by XCS and still solving optimally the problems. Then, we show how the recursive least squares approach can be extended to multistep problems and we test it on the woods problems. Our findings confirm the results of previous works on recursive least squares applied to function approximation problems: they let XCSF converge faster toward the optimal solution and evolve a more compact solution. Note that O'Hara and Bull proposed in [13] a similar analysis of neural prediction to the woods environments. However their results cannot be directly compared with the ones presented in this paper, because they focused on the N-XCS classifier system [1], which replaces the classifier condition-action structure with a neural network.

II. Description of XCSF

XCSF differs from XCS in three respects: (i) classifier conditions are extended for numerical inputs, as done in XCSI [18]; (ii) classifiers are extended with a vector of weights w, that are used to compute prediction; finally, (iii) the original update of classifier prediction must be modified so that the weights are updated instead of the classifier prediction. These three modifications result in a version of XCS, XCSF [19], [20], that maps numerical inputs into actions with an associated calculated prediction. In the original paper [19] classifiers have no action and it is assumed that XCSF outputs the estimated prediction, instead of the action itself. In this paper, we consider the

Daniele Loiacono (loiacono@elet.polimi.it) is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy.

Pier Luca Lanzi (lanzi@elet.polimi.it) is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy. Pier Luca Lanzi (lanzi@illigal.ge.uiuc.edu) is also member of the Illinois Genetic Algorithm Laboratory (IlliGAL), University of Illinois at Urbana Champaign, Urbana, IL 61801, USA.

978-1-4244-1823-7/08/$25.00©2008 IEEE

version of XCSF with actions and linear prediction (named XCS-LP [20]) in which more than one action is available. As said before, throughout the paper we do not keep the (rather historical) distinction between XCSF and XCS-LP since the two systems are basically identical except for the use of actions in the latter case.

Classifiers. In XCSF, classifiers consist of a condition, an action, and four main parameters. The condition specifies which input states the classifier matches; as in XCSI [18], it is represented by a concatenation of interval predicates, $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers, though they might be also real. The action specifies the action for which the payoff is predicted. The four parameters are: the weight vector w, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; the numerosity *num*, a counter used to represent different copies of the same classifier. Note that the size of the weight vector w depends on the type of approximation. In the case of piecewise-linear approximation, considered in this paper, the weight vector w has one weight w_i for each possible input, and an additional weight w_0 corresponding to a constant input x_0, that is set as a parameter of XCSF.

Performance Component. XCSF works as XCS. At each time step t, XCSF 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 the current inputs and has a random action. Each interval predicate $int_i = (l_i, u_i)$ in the condition of a covering classifier is generated as $l_i = s_t(i) - \text{rand}(r_0)$, and $u_i = s_t(i) + \text{rand}(r_0)$, where $s_t(i)$ is the input value of state s_t matched by the interval predicated int_i, and the function rand(r_0) generates a random integer in the interval $[0, r_0]$ with r_0 fixed integer. The weight vector w of covering classifiers is randomly initialized with values from [-1,1]; all the other parameters are initialized as in XCS (see [4]).

For each action a_i in [M], XCSF computes the *system prediction* which estimates the payoff that XCSF expects when action a_i is performed. As in XCS, in XCSF the *system prediction* of action a is computed by the fitness-weighted average of all matching classifiers that specify action a. However, in contrast with XCS, in XCSF classifier prediction is computed as a function of the current state s_t and the classifier vector weight w. Accordingly, in XCSF system prediction is a function of both the current state s and the action a. Following a notation similar to [3], the system prediction for action a in state s_t, $P(s_t, a)$, is defined as:

$$P(s_t, a) = \frac{\sum_{cl \in [M]|_a} cl.p(s_t) \times cl.F}{\sum_{cl \in [M]|_a} cl.F} \quad (1)$$

where *cl* is a classifier, $[M]|_a$ represents the subset of classifiers in [M] with action a, *cl.F* is the fitness of *cl*; $cl.p(s_t)$ is the prediction of *cl* computed in the state s_t. In particular, when piecewise-linear approximation is considered, $cl.p(s_t)$ is computed as:

$$cl.p(s_t) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times s_t(i)$$

where $cl.w_i$ is the weight w_i of *cl* and x_0 is a constant input. The values of $P(s_t, a)$ form the *prediction array*. Next, XCSF selects an action to perform. The classifiers in [M] that advocate the selected action are put in the current *action set* [A]; the selected action is sent to the environment and a reward P is returned to the system.

Reinforcement Component. XCSF uses the incoming reward P to update the parameters of classifiers in action set [A]. The weight vector w of the classifiers in [A] is updated using a *modified delta rule* [15]. For each classifier $cl \in [A]$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as:

$$\Delta w_i = \frac{\eta}{|s_t(i)|^2}(P - cl.p(s_t))s_t(i) \quad (2)$$

where η is the correction rate and $|s_t|^2$ is the norm the input vector s_t, (see [19] for details). Equation 2 is usually referred to as the "*normalized*" Widrow-Hoff update or "*modified* delta rule", because of the presence of the term $|\mathbf{s}_t(i)|^2$ [6]. The values Δw_i are used to update the weights of classifier *cl* as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (3)$$

Then the prediction error ε is updated as:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p(s_t)| - cl.\varepsilon)$$

Finally, classifier fitness is updated as in XCS.

Discovery Component. The genetic algorithm and subsumption deletion in XCSF work as in XCSI [18]. On a regular basis depending on the parameter θ_{ga}, the genetic algorithm is applied to classifiers in [A]. It selects two classifiers with probability *proportional to their fitness*, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. Crossover and mutation work as in XCSI [18], [19]. The resulting offspring are inserted into the population and two classifiers are deleted to keep the population size constant.

Subsumption Deletion. Wilson [17] introduced two *subsumption deletion* procedures to broaden the generalization capability of XCS. The same procedures can be also applied to XCSF for the same reasons. The first procedure, *GA subsumption*, checks offspring classifiers to see whether their conditions are logically subsumed by the condition of an accurate and sufficiently experienced parent. If an offspring is "GA subsumed", it is not inserted in the population but the parent's numerosity is increased. The second procedure, *action set subsumption*, searches in the current action set for the most general classifier that is both *accurate* and *sufficiently experienced*. Then all the classifiers in the action set are tested against the general one to see whether it subsumes them and the subsumed classifiers are eliminated from the population.

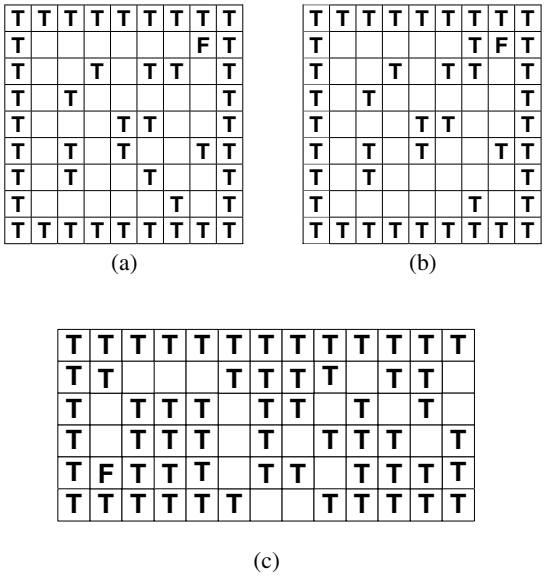


Fig. 1. The Woods environments: (a) `Maze5`, (b) `Maze6`, and (c) `Woods14`. The "`T`" symbols identify the cell with obstacles, while "`F`" symbols identify the goal positions.

III. The Woods Problems

In this set of experiments we compared XCSF and XCS on the *Woods* series of environments. These are grid environments in which each cell can contain an obstacle (identified by a "`T`" symbol), a goal (an "`F`" symbol), otherwise it can be empty. An agent placed in the environment must learn to reach goal positions. The agent perceives the environment by eight sensors, one for each adjacent cell, and can move in any of the adjacent cells. If the destination cell contains an obstacle the move does not take place; if the destination cell is blank then the move takes place; finally, if the cell contains a goal the agent moves receiving a constant reward, and the problem ends. Each sensor is represented by two bits depending on whether the environment has one or two types of obstacles and goals: `10` indicates the presence of an obstacle `T`; `11` indicates a goal `F`; `00` indicates an empty cell. Classifiers conditions are 16 bits long (2 bits $\times$ 8 cells). In particular we focused on the Woods environments reported in Figure 1. These three environments, called respectively `Maze5` (Figure 1a), `Maze6` (Figure 1b), and `Woods14` (Figure 1c), have been widely studied in the LCS literature [2].

For each problem the agent starts in a random blank cell of the environment; then it moves under the control of the system until it reaches a goal position receiving a constant reward; then the problem ends.

IV. Experimental Results

We compared XCS to XCSF on the Woods problems, a set of well known problems in the learning classifier systems literature. In this work, we followed the standard experimental design used in the literature [16]. Each experiment consists of a number of problems that the system must solve. Each problem is either a *learning* problem or a *test* problem. In *learning* problems, the system selects actions randomly from those represented in the match set. In *test* problems, the system always selects the action with highest prediction. The genetic algorithm is enabled only during *learning* problems, and it is turned off during *test* problems. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate. In the following experiments, the performance is computed as the average number of steps to reach the goal position in the last 50 test problems. In addition, as a measure of the generalization capabilities of the systems compared, is reported the number of macroclassifiers evolved during the learning process.

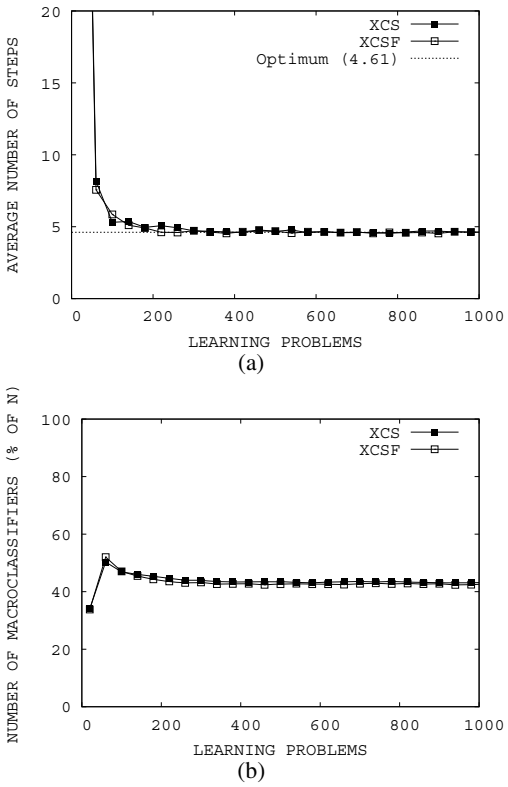


Fig. 2. XCS and XCSF applied to `Maze5` problem: (a) performance and (b) population size. Curves are average over 50 experiments.

Maze5. In the first experiment we applied XCS and XCSF to the `Maze5` problem. The parameters are set, according to [2], as follows: N = 3000, ϵ_0=10 $P_\# = 0.3$, β=0.2, $\alpha = 0.1$, $\nu = 5$, γ=0.7, χ=0.8, μ=0.01, $p_{\text{explr}} = 1.0$, θ_{del}=20 θ_{GA}=25, and $\delta = 0.1$; both GA-subsumption and action-set subsumption are off; the parameter x_0 for XCSF is 5. Figure 2 compares XCS and XCSF on the `Maze5` problem. It can be noticed that XCSF reaches an optimal performance almost as fast as XCS (Figure 2a) and both systems evolve almost the same number of macroclassifiers (Figure 2b). This is not surprising

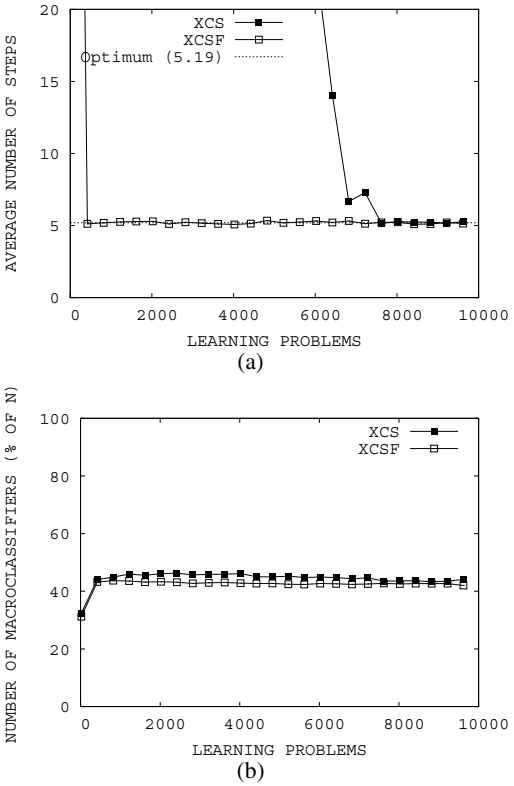


Fig. 3. XCS and XCSF applied to `Maze6` problem: (a) performance and (b) population size. Curves are average over 50 experiments.

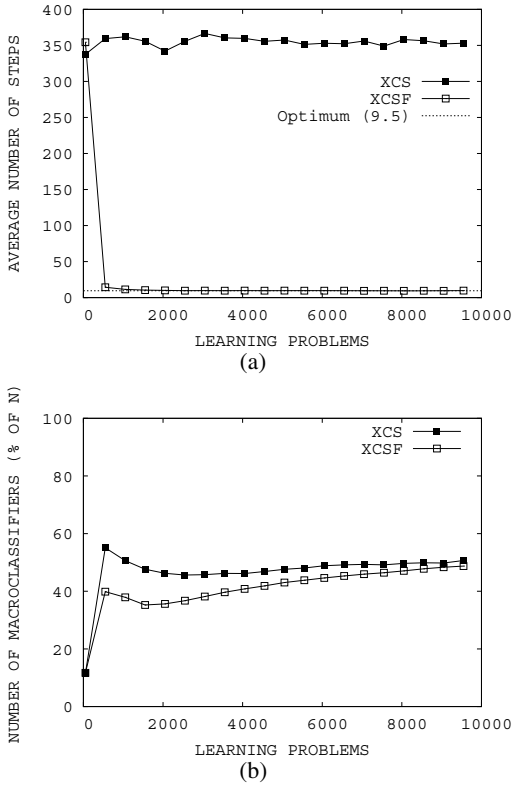
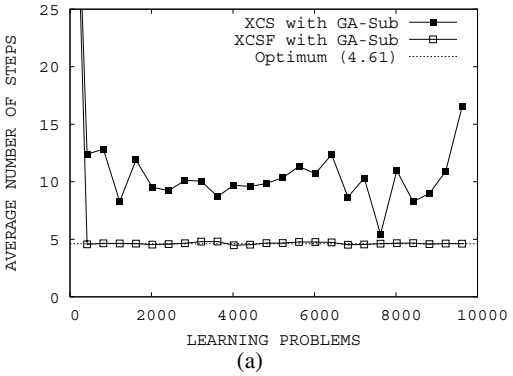


Fig. 4. XCS and XCSF applied to `Woods14` problem: (a) performance and (b) population size. Curves are average over 50 experiments.

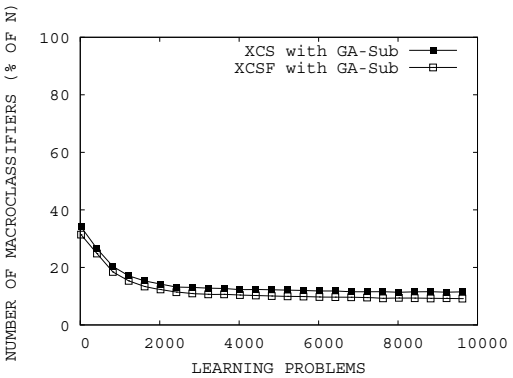
because `Maze5` is rather a simple problem and the XCSF generalization capabilities do not seem to give any substantial advantage.

Maze6. In the second experiment we moved to the `Maze6` problem. Although it differs from `Maze5` only in two positions, it is a much more difficult problem since the goal position can be reached only from the bottom, making the exploration of the environment a much longer random walk and resulting in more delayed reward distribution. The experiments has been performed with the same parameter settings of the previous experiment except for $\theta_{GA} = 100$ according to the settings suggested in [2] Figure 3 compares XCS and XCSF on the `Maze6` problem. Despite both systems reach an optimal performance in the end, Figure 3a shows that XCSF learns much faster than XCS. The analysis of single runs reveals that XCS actually learns almost as fast as XCSF in 14 out of 20 runs. Thus, when the genetic search is able to find quickly a suitable problem decomposition, XCS has a performance comparable to XCSF. Instead, in 6 out of 20 runs, XCS has a lot of problems to find a suitable problem decomposition. On the other hand, the same problem does not affect the performance of XCSF, because the generalization capabilities of the computed prediction, offer a better guidance to the discovery component of the system to search a good problem decomposition. Concerning the size of the final population, Figure 3b shows that XCS and XCSF evolve almost the same number of macroclassifiers.

Woods14. Then, we applied XCS and XCSF to a still more difficult problem, the `Woods14` problem. This environment is a kind of corridor where reaching the goal may take up to 18 actions from the farthest position. The experiment has been performed with the following parameter settings: N = 4000, ϵ_0=10 $P_\# = 0.3$, β=0.2, $\alpha = 0.1$, $\nu = 5$, γ=0.95, χ=0.8, μ=0.01, $p_{\text{explr}} = 0.3$, θ_{del}=20 θ_{GA}=400, and $\delta = 0.1$; both GA-subsumption and action-set subsumption are off; the parameter x_0 for XCSF is 5. In particular, we set $p_{\text{explr}} = 0.3$ in order to keep limited the number of steps during the exploration episodes; in addition, we set γ=0.95 to make it easier for XCS and XCSF to distinguish between the different payoff values in the farthest position from the goal. Figure 4 compares XCS and XCSF on the `Woods14` problem. It can be noticed from Figure 4b that only XCSF is able to reach an optimal performance, while XCS is very far from being optimal. In this experiment the analysis of the population size evolved by XCS does not make sense, since XCS is unable to solve the problem. An analysis of the single runs shows that XCS is able to learn, reaching an optimal performance, only in 1 out of 20 runs. On the other hand, XCSF is able to

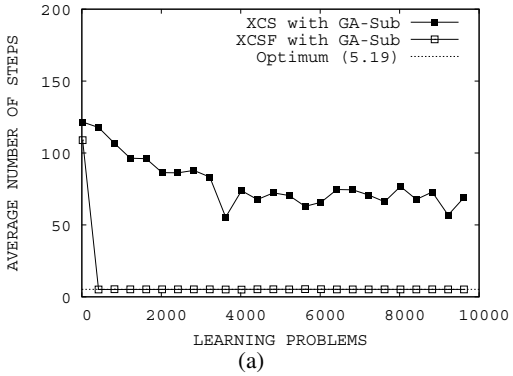


(a)

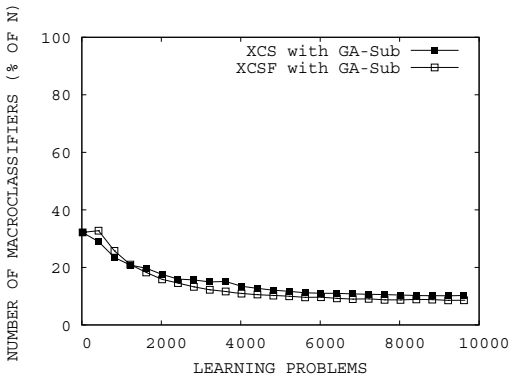


(b)

Fig. 5. XCS and XCSF applied to `Maze5` problem: (a) performance and (b) population size. Curves are average over 50 experiments.



(a)



(b)

Fig. 6. XCS and XCSF applied to `Maze6` problem with GA-subsumption: (a) performance and (b) population size. Curves are average over 50 experiments.

effectively reach the optimal performance in all the 20 runs.

Experiments with Subsumption. Finally, in the last set of experiments we investigated how deletion subsumption affects the performance of the XCS and XCSF classifier systems. The deletion subsumption was introduced to increase the pressure toward maximally general and accurate solutions [16], [17]. It works well in single step problems, like the Boolean functions, but it is seldom used in multistep environments, such as woods environments. In fact, the additional generalization pressure may be detrimental in problems allowing few generalizations [7], [2]. Therefore, we tested whether the more powerful generalization capabilities due to computed prediction can be combined with the GA-subsumption to let XCSF evolve solutions that are more compact than those evolved by XCS while preserving the optimality. To this purpose, we repeated the first two experiments reported in this section, on the `Maze5` and on the `Maze6`, with the same parameter settings except for the GA-subsumption that is on with the θ_{sub} set to the same value of θ_{GA}. Figure 5a and Figure 6a compare the performance of XCS and XCSF respectively on the `Maze5` and on the `Maze6` problems. The results show that, while the GA-subsumption does not affect the final performance of XCSF, XCS is not able to solve this problem optimally any longer. In particular, with the GA-subsumption on, XCS learns to solve the `Maze5`, although it learn a final performance far from optimal, but it is not able to learn a reasonable policy to solve `Maze6`. On the other hand, Figure 5b and Figure 6b show that with GA-subsumption both XCS and XCSF evolve a very compact population, that is about four times smaller than the one evolved without GA-subsumption.

V. XCSF WITH RECURSIVE LEAST SQUARES

Recently [12], XCSF has been extended with the recursive least squares for updating the classifier weights. At time step t, given the current input $\mathbf{x}_t = [x_0 \mathbf{s}_t]^T$ and the target payoff P, recursive least squares updates the weight vector $\mathbf{w}$ as

$$\mathbf{w}_t \quad = \quad \mathbf{w}_{t-1} + \mathbf{k}_t[P - \mathbf{x}_t \mathbf{w}_{t-1}],$$

where, $\mathbf{k}_t$ is called *gain vector* and is computed as

$$\mathbf{k}_t \quad = \quad \frac{V_{t-1}\mathbf{x}_t}{1 + \mathbf{x}_t^T V_{t-1} \mathbf{x}_t}, \tag{4}$$

while matrix $\mathbf{V}_t$ is computed recursively by,

$$\mathbf{V}_t \quad = \quad \left[I - \mathbf{k}_t \mathbf{x}_t^T\right] V_{t-1}. \tag{5}$$

Algorithm 1 Update classifier cl with RLS algorithm

procedure UPDATE_PREDICTION(cl, s, P)
  error $\leftarrow P - cl.p(s)$; ▷ Compute the current error
  $x(0) \leftarrow x_0$; ▷ Build $\mathbf{x}$ by adding x_0 to s
  for $i \in \{1, \ldots, |s|\}$ **do**
    $x(i) \leftarrow s(i)$;
  end for
  if # of updates from last reset $> \tau_{rls}$ **then**
    $cl.V \leftarrow \delta_{rls}\mathbf{I}$ ▷ Reset cl.V
  end if
  $\eta_{rls} \leftarrow (1 + \mathbf{x} \cdot cl.V \cdot \mathbf{x}^T)^{-1}$;
  $cl.V \leftarrow cl.V - \eta_{rls}^{-1} cl.V \cdot \mathbf{x}^T \mathbf{x} \cdot cl.V$; ▷ Update $cl.V$
  $\mathbf{k}^T \leftarrow cl.V \cdot \mathbf{x}^T$;
  for $i \in \{0, \ldots, |s|\}$ **do** ▷ Update classifier's weights
    $cl.w_i \leftarrow cl.w_i + \mathbf{k}(i) \cdot$ error;
  end for
end procedure

. The matrix $\mathbf{V}(t)$ is usually initialized as $\mathbf{V}(0) = \delta_{rls}\mathbf{I}$, where δ_{rls} is a positive constant and $\mathbf{I}$ is the $n \times n$ identity matrix. An higher δ_{rls}, denotes that initial parametrization is uncertain, accordingly, initially the algorithm will use a higher, thus faster, update rate ($\mathbf{k}_t$). A lower δ_{rls}, denotes that initial parametrization is rather certain, accordingly the algorithm will use a slower update. It is worthwhile to say that the recursive least squares approach presented above involves two basic underlying assumptions [6], [5]: (i) the noise on the target payoff P used for updating the classifier weights can be modeled as a unitary variance white noise and (ii) the optimal classifier weights vector does not change during the learning process, i.e., the problem is stationary. While the first assumption is often reasonable and has usually a small impact on the final outcome, the second assumption is not justified in many problems and may have a big impact on the performance. In the literature [6], [5] many approaches have been introduced for relaxing this assumption. In particular, a straightforward approach is the *resetting* of the matrix $\mathbf{V}$: every τ_{rls} updates, the matrix $\mathbf{V}$ is reset to its initial value $\delta_{rls}\mathbf{I}$. Intuitively, this prevent RLS to converge toward a fixed parameter estimate by continually restarting the learning process. We refer the interested reader to [6], [5] for a more detailed analysis of recursive least squares and other related approaches, like the well known Kalman filter.

The extension of XCSF with recursive least squares is straightforward: we added to each classifier the matrix V as an additional parameter and we replaced the usual update of classifier weights with the recursive least squares update described above and reported as Algorithm 1.

Experimental Results. Now we compare the two XCSF versions on a set of multistep problems: the Woods problems. In the first two experiments we compared the RLS and the Widrow-Hoff update rules on `Maze5` and on `Maze6`. The experiments have been performed with the following parameter settings: N = 3000, ϵ_0=10 $P_\# = 0.3$, β=0.2, $\alpha = 0.1$, $\nu = 5$, γ=0.7, χ=0.8, μ=0.01, $p_{\text{explr}} = 1.0$, θ_{del}=20,

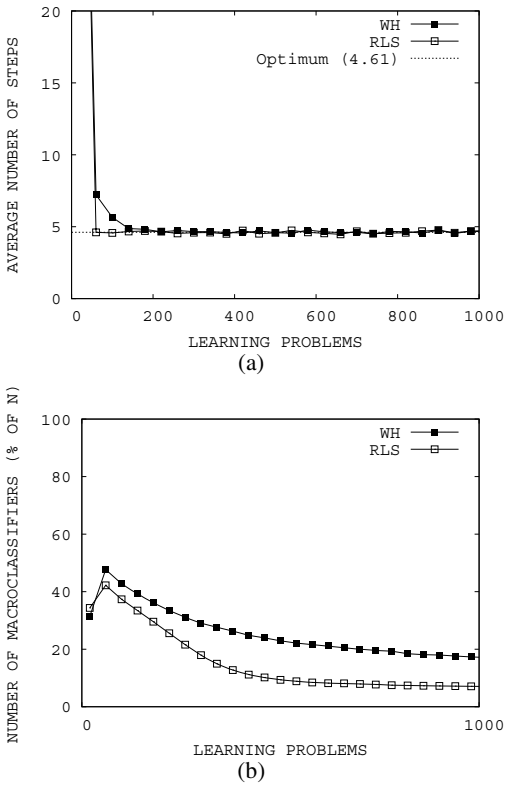


Fig. 7. XCSF with Widrow-Hoff and with RLS update applied to `Maze5` problem: (a) performance and (b) population size. Curves are average over 50 experiments.

and $\delta = 0.1$; GA-subsumption is on with $\theta_{sub} = 50$ while action-set subsumption is off. We set θ_{GA}=25 in `Maze5` and θ_{GA}=100 in `Maze6`. The parameter x_0 for XCSF is 5; with the RLS update we used $\delta_{rls} = 100$ and $\tau_{rls} = 50$. With respect to the experiments presented in the previous chapter, we have used the GA-subsumption for a better comparison of the generalization capabilities of the two versions of XCSF. Figure 7 compares the two versions of XCSF on the `Maze5` problem. Both systems reach an optimal performance (see Figure 7a), but XCSF with RLS learns slightly faster. In addition, it can be noticed (Figure 7b) that when the RLS update is used, XCSF is able to evolve a more compact solution. When we move on the `Maze6` problem, we got almost the same results. Again, both versions of XCSF converge toward to the optimal solution, but as showed in Figure 8a, XCSF with RLS update converges much faster. Also the population evolved by XCSF with RLS is, on the average, much more compact than the one evolved by XCSF with the Widrow-Hoff update rule (Figure 8b).

In the last experiment, we compared the two prediction updates also on the `Woods14`, with the following parameter settings: N = 4000, ϵ_0=10 $P_\# = 0.3$, β=0.2, $\alpha = 0.1$, $\nu = 5$, γ=0.95, χ=0.8, μ=0.01, $p_{\text{explr}} = 0.3$, θ_{del}=20 θ_{GA}=400, and

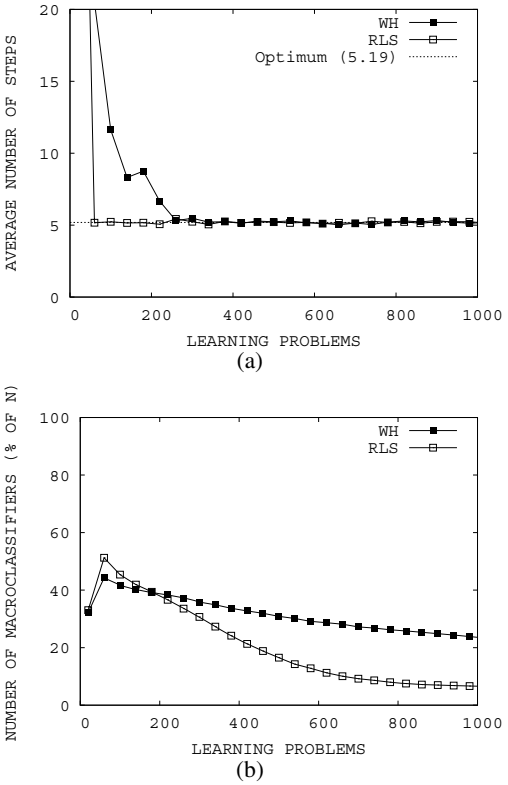


Fig. 8. XCSF with Widrow-Hoff and with RLS update applied to Maze6 problem: (a) performance and (b) population size. Curves are average over 50 experiments.

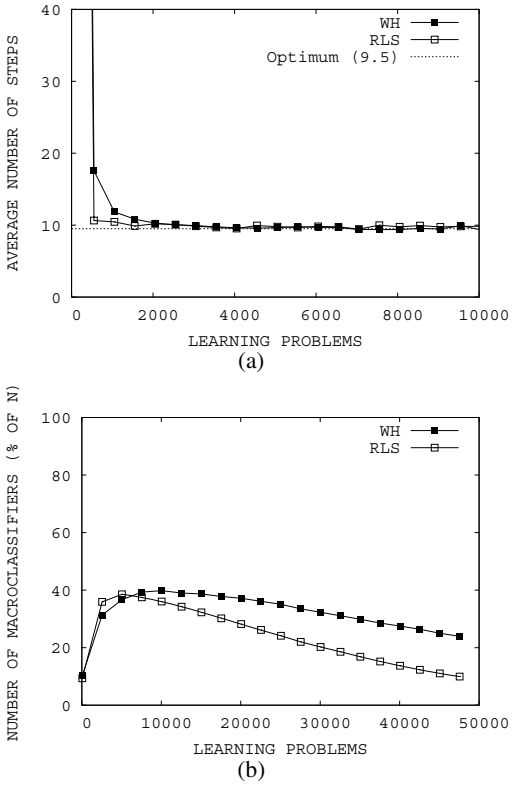


Fig. 9. XCSF with Widrow-Hoff and with RLS update applied to Woods14 problem: (a) performance and (b) population size. Curves are average over 50 experiments.

$\delta = 0.1$; both GA-subsumption and action-set subsumption are off; $p_{\text{explr}} = 0.3$ in order to keep limited the number of steps during the exploration episode; with the RLS update we used $\delta_{rls} = 100$ and $\tau_{rls} = 50$ and we set $x_0 = 5$. In this experiment we did not use the GA-subsumption because we observed that it may have disruptive effects on the final performance (results are not reported here). Figure 9 compares the performance of XCSF with the two prediction updates studied. In particular, Figure 9a shows that XCSF with RLS update is able to learn faster than XCSF with Widrow-Hoff rule, confirming the previous experimental results. On the other hand, Figure 9b shows that, in this problem, XCSF with the RLS update evolves a bigger population. However, the analysis of the populations evolved by XCSF with both the update algorithms, reveals that XCSF with the RLS update evolves, on the average, classifier more general than the one evolved by XCSF with the Widrow-Hoff update. Therefore, when GA-subsumption is used only the more general classifier survive and XCSF with RLS update is able to evolve a very compact solution, as in the Maze5 and Maze6 problems. Instead when the GA-subsumption is not used, it may require a lot of problems for the more general classifiers to take over the populations, as in the Woods14 problem.

VI. Conclusion

So far, XCSF has been mainly applied to problems defined over a real-valued domain. We compared XCS to XCSF on well known binary multistep problems, the woods environments. Although in these problems the higher generalization capabilities of XCSF cannot be exploited to evolve more compact populations, our results show that XCSF outperforms XCS both in terms of convergence speed and in terms of final performance on difficult problems. In fact the higher approximation capabilities of XCSF play an important role in the early stage of the learning process to prevent premature overgeneralizations. This issue is still more evident when the GA-subsumption is applied: our results suggest that while the GA-subsumption may have a disruptive effect on the final performance of XCS, it does not affect negatively the performance of XCSF. In addition, we investigated the application of the recursive least squares update, a very effective algorithm to update the classifier prediction, to multistep problems. The recursive least squares update was originally devised for single-step problems and it was never applied before to multistep problems. In this work we extended it with covariance matrix resetting, a well known approach to

deal with the problem of learning a non-stationary target as it happens in multistep problems during learning. The experimental results show that XCSF with recursive least squares converges faster toward the optimal solution and evolves more compact solutions than XCSF with the usual classifier prediction update.

REFERENCES

[1] Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis andR. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegenerand L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E.Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[2] 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.

[3] Martin V. Butz and Martin Pelikan. Analyzing the evolutionary pressures in xcs. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-MichaelVoigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshkand Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 935–942, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

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

[5] Graham C. Goodwin and Kwai Sang Sin. *Adaptive Filtering: Prediction and Control*. Prentice-Hall information and system sciences series, March 1984.

[6] Simon Haykin. *Adaptive Filter Theory*. Prentice-Hall, 2001. 4th Edition.

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

[8] 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.

[9] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction for the learning of boolean functions. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 588–595, Edinburgh, UK, September 2005. IEEE.

[10] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with Computed Prediction in Continuous Multistep Environments. In *Proceedings of the IEEE Congress on Evolutionary Computation – CEC-2005*, pages 2032–2039, Edinburgh, UK, September 2005. IEEE.

[11] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction in multistep environments. In Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantu-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llora, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors, *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1859–1866, Washington DC, USA, 25-29 June 2005. ACM Press.

[12] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the XCSF classifier system: Analysis, improvement, and extension. *Evolutionary Computation*, 15(2):133–168, 2007.

[13] T. Ohara and Larry Bull. Prediction calculation in accuracy-based neural learning classifier systems. Technical Report UWELCSG04-004, Learning Classifier System Group. University of West England. Bristol, U.K., 2004.

[14] C.J.C.H. Watkins and P. Dayan. Technical note: *Q*-Learning. *Machine Learning*, 8:279–292, 1992.

[15] B. Widrow and M. E. Hoff. *Adaptive Switching Circuits*, chapter Neurocomputing: Foundation of Research, pages 126–134. The MIT Press, Cambridge, 1988.

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

[17] Stewart W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998. http://prediction-dynamics.com/.

[18] Stewart W. Wilson. Mining Oblique Data with XCS. volume 1996 of *Lecture notes in Computer Science*, pages 158–174. Springer-Verlag, April 2001.

[19] Stewart W. Wilson. Classifiers that approximate functions. *Journal of Natural Computing*, 1(2-3):211–234, 2002.

[20] Stewart W. Wilson. Classifier systems for continuous payoff environments. In Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund Burke, Paul Darwen, Dipankar Dasgupta, Dario Floreano, James Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andy Tyrrell, editors, *Genetic and Evolutionary Computation – GECCO-2004, Part II*, volume 3103 of *Lecture Notes in Computer Science*, pages 824–835, Seattle, WA, USA, 26-30 June 2004. Springer-Verlag.