Gradient-based Learning Updates Improve XCS Performance in Multistep Problems

Martin V. Butz, David E. Goldberg, and Pier Luca Lanzi

Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{butz,deg,lanzi}@illigal.ge.uiuc.edu

Abstract. This paper introduces a gradient-based reward prediction update mechanism to the XCS classifier system as applied in neural-network type learning and function approximation mechanisms. A strong relation of XCS to tabular reinforcement learning and more importantly to neural-based reinforcement learning techniques is drawn. The resulting gradient-based XCS system learns more stable and reliable in previously investigated hard multistep problems. While the investigations are limited to the binary XCS classifier system, the applied gradient-based update mechanism appears also suitable for the real-valued XCS and other learning classifier systems.

1 Introduction

Despite the recent encouraging applications of the accuracy-based XCS classifier system in data-mining problems [1–4], successful applications in multistep problems have been restricted to small problems [5, 6]. It was shown that without further additions, XCS is not able to solve environments robustly that allow only few generalizations or that require a larger number of steps until reinforcement is encountered [7, 8].

Although *learning classifier systems* (LCSs) were developed within the evolutionary computation community rather independently from reinforcement learning research, temporal difference learning methods in LCSs can be compared to *reinforcement learning* (RL). For example, Q-learning [9] is tightly linked to the RL mechanism in the ZCS system [10] and the XCS system [5, 11]. Besides the RL relation, LCSs may be viewed as evolutionary-based *function approximation* methods. The XCS system, for example, has been shown to be applicable as a pure function approximator [12].

Over the last years, it has become clear that tabular-based RL scales-up poorly. Thus, function approximation methods have been applied in the RL literature [13, 14]. These mostly neural-based function approximators can be highly unstable if direct gradient methods are used to implement Q-learning [14]. Accordingly, residual gradient methods have been developed to improve robustness [15].

The aim of this paper is to explore the possibility of applying gradient-based update methods in LCSs and in particular in the XCS system. We show how LCSs are related to neural function approximation methods and use similar gradient-based methods to improve performance. We show that XCS with gradient update methods reaches higher robustness and stability.

The paper is structured as follows. First, we provide the necessary background knowledge on RL including Q-learning and the gradient approaches. Next, we introduce XCS and reveal the differences in its RL approach. We consequently extend XCS with a gradient-based update mechanism. The subsequent performance analysis shows that XCS with gradient descent learns typically hard multistep problems much more reliable. In conclusion, we discuss the similarities of XCS, and LCSs in general, with neural-based RL methods and suggest further comparisons and enhancements.

2 Reinforcement Learning

Reinforcement learning problems are problems in which an agent interacts with an unknown environment. The environment provides state information and a numerical reward as feedback to the agent. The agent's task is to maximize the cumulative reward on the long run. LCSs essentially face a RL problem.

Most research in RL focuses on problems that can be modeled with a finite Markov decision process (MDP). An MDP is formally defined by a finite set S of states; a finite set A of actions; a transition function T ($T : S \times A \rightarrow \Pi(S)$) that assigns to each state-action pair a probability distribution ($\Pi(S)$) over states S, and a reward function R ($R : S \times A \rightarrow \mathbb{R}$).

At a certain time t, the agent senses its environment perceiving state s_t; based on this state information the agent selects an action a_t. After the execution of action a_t, the agent receives a scalar reward r_{t+1} and a new state s_{t+1}. The agent's goal is to *maximize* the amount of reward it receives from the environment *in the long run*. This is usually expressed as the *discounted expected payoff* which at time t is defined as follows:

$$E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}\right], \tag{1}$$

where γ is the *discount factor* ($0 \leq \gamma \leq 1$) that specifies the importance of future reward. The larger γ, the more important are more distant future rewards.

In RL, the agent learns how to maximize the incoming reward by developing an action-value function $Q(\cdot,\cdot)$ (or a state value function $V(\cdot)$) that maps state-action pairs (or states) into the corresponding expected payoff value (Equation 1).

2.1 Q-Learning

The Q-learning algorithm [9] iteratively approximates the optimal action-value function Q^*, which maps all state-action pairs to the associated expected payoff. Usually, Q^* is approximated by a tabular state-action representation often referred to as *tabular Q-learning*. At time step t, when the agent senses the environment to be in state s_t, and receives reward r_t for performing a_{t-1} in state s_{t-1}, the entry $Q(s_{t-1}, a_{t-1})$, is updated according to the formula:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \beta(r_t + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1}))$$

where β is the *learning rate* ($0 \leq \beta \leq 1$). Given a RL problem modeled as an MDP, under adequate hypotheses, Q-learning converges with probability one to the optimal action-value function Q^*.

2.2 Q-learning with Gradient Descent

Tabular Q-learning is simple and easy to implement but it is infeasible for larger problems because the size of the Q-table (which is $|S| \times |A|$) grows exponentially in the problem dimensions. To cope with this complexity, approximation methods need to be used. Particularly, a generalized representation of the optimal action-value function Q^* needs to be learned from a limited number of experiences. In RL, generalization is usually implemented by function approximation techniques. Often, gradient descent techniques are used to build a good approximation of function Q^* from online experience.

When applying gradient descent to approximate Q^* online, we are actually trying to minimize the error between the desired payoff value associated with the current state-action pair (estimated by $r + \gamma \max_{a \in A} Q(s_t, a)$) and the current corresponding payoff estimate $Q(s_{t-1}, a_{t-1})$. If the function approximator is parameterized by a weight matrix W, the change Δw for each weight w is

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w},$$

where β is the learning rate and γ is the discount factor [14, 15, 13]. It can be seen, that the weight update depends both (i) on the difference between the desired value and the current value associated with the current state-action pair and (ii) on the gradient component represented by the partial derivate of the current payoff value with respect to the weight. The gradient component essentially adjusts the weight update with respect to its relative contribution to the Q-value estimate. Function approximation techniques that update their weights according to the equation above are called *direct algorithms*.

While tabular RL methods can be guaranteed to converge, function approximation methods based on direct algorithms have been shown to be fast but often unstable [14, 15]. To improve the convergence of function approximation techniques in RL applications, another class of techniques, namely *residual gradient algorithms*, have been developed [15]. Residual gradient algorithms are slower but more stable than direct algorithms and, most importantly, they can be guaranteed to converge under adequate assumptions. Residual algorithms extend direct gradient descent approaches by adjusting the gradient of the current state with an estimate of the effect of the weight change on the successor state.

The weight update Δw for Q-learning implemented with a *residual* approach becomes the following:

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1}))$$
$$\left[\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w} - \phi\gamma \frac{\partial}{\partial w}\left(\max_{a \in A} Q(s_t, a)\right)\right], \tag{2}$$

where the partial derivative $\frac{\partial}{\partial w}(\max_{a \in A} Q(s_t, a))$ estimates the effect that the current modifications of the weight have on the value of the next state. Note that since this adjustment involves the next state, the discount factor γ must also be taken into account. Parameter *phi* weighs the degree of influence of the next state on the update in the current state.

With this RL knowledge in mind, we now turn to the XCS classifier system investigating how XCS approximates the optimal Q-function Q^* and how (residual) gradient-based updates can be incorporated into XCS and into LCSs, in general.

3 XCS in Brief

XCS [5] is essentially a RL method in which generalization is obtained through the evolution of a *population* of condition-action-prediction rules (*classifiers*). With respect to the usual RL settings in XCS, (i) the population of classifiers approximates the optimal action-value function Q^* and (ii) the classifiers and their parameters roughly correspond to the weight matrix W that is used in function approximation approaches to generalize over the space of possible solutions. This section gives a short introduction to XCS. A detailed algorithmic description can be found in [16].

In XCS, classifiers consist of a condition, an action, and four main parameters: (i) the prediction p estimates the average payoff that the system expects when the classifier is used; (ii) the prediction error ε estimates the average absolute error of the prediction p; (iii) the fitness F estimates the average relative accuracy of the payoff prediction given by p; and (iv) the numerosity num indicates how many copies of classifiers with the same condition and the same action are present in the population.

At each iteration, XCS builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory inputs; if [M] contains less than θ_{nma} actions, *covering* takes place and creates a new classifier that matches the current inputs and has an unrepresented action. For each possible action a_i in [M], XCS computes the *system prediction* $P(a_i)$ which estimates the payoff that XCS expects if action a_i is performed. The *system prediction* is computed as the fitness weighted average of the predictions of classifiers in [M], $cl \in$[M], which advocate action a_i (i.e., $cl.a{=}a_i$):

$$P(a_i) = \frac{\sum_{cl_k \in [M]|_{a_i}} p_k \times F_k}{\sum_{cl_k \in [M]|_{a_i}} F_k}, \tag{3}$$

where $[M]|_{a_i}$ represents the subset of classifiers in $[M]$ with action a_i, p_k refers to the prediction of classifier cl_k, and F_k refers to the fitness of classifier cl_k. Next, XCS selects an action using, e.g., an epsilon-greedy action selection mechanism [13] based on the system prediction values $P(a_i)$. The classifiers in [M] which advocate the selected action form the current *action set* [A]. The selected action is performed in the environment, and a scalar reward r is returned to XCS together with a new input configuration.

After the reward r is received and the next match set $[M]$ is formed with respect to the resulting sensory input, the prediction value P is computed as follows.

$$P = r + \gamma \max_{a \in A} P(a). \tag{4}$$

Next, the parameters of the classifiers in [A] are updated in the following order [16]: prediction, prediction error, and finally fitness. Prediction p and prediction error ε are updated with learning rate β $(0 \le \beta \le 1)$,

$$p \leftarrow p + \beta(P - p). \tag{5}$$

$$\varepsilon \leftarrow \varepsilon + \beta(|P - p| - \varepsilon) \tag{6}$$

Based on the current prediction error, fitness is updated. The update consists of three steps successively determining *raw accuracy* κ, *relative accuracy* κ', and new fitness F:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon \le \varepsilon_0 \\ \alpha(\varepsilon/\varepsilon_0)^{-\nu} & \text{otherwise.} \end{cases} \tag{7}$$

$$\kappa' = \frac{(\kappa \times num)}{\sum_{cl \in [A]}(cl.\kappa \times cl.num)}, \tag{8}$$

$$F \leftarrow F + \beta(\kappa' - F) \tag{9}$$

Parameter ε_0 $(\varepsilon_0 > 0)$ specifies the threshold that determines to what extent prediction errors are accepted; α $(0 < \alpha < 1)$ causes a strong distinction between accurate and inaccurate classifiers; ν $(\nu > 0)$ and ε_0 determine the steepness of the slope used to calculate classifier accuracy; $cl.\kappa$ is the raw accuracy of classifier cl; $cl.num$ is the numerosity of classifier cl.

On a regular basis depending on the parameter θ_{GA}, a genetic algorithm (GA) is applied to classifiers in [A]. The GA selects two classifiers with probability *proportional to their fitness*, copies them, and performs crossover with probability χ; each allele is mutated with probability μ . The resulting offspring are inserted into the population and two classifiers are deleted from the whole population to keep the population size constant.

4 XCS with Gradient Descent

In general, XCS uses Q-learning techniques but can also be compared to a function approximation mechanism. In this section, we analyze the similarities between, tabular Q-learning, Q-learning with gradient descent, and XCS. We show

how to fuse the two capabilities adding gradient descent to XCS's parameter estimation mechanism.

4.1 Q-Value Estimations and Update Mechanisms

In Section 2.1 we saw that tabular Q-learning iteratively approximates the Q-table entries using the difference between estimated and experienced reward signal to adjust the estimate:

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})). \quad (10)$$

As seen in Section 2.2, in a function approximation approach the Q-table is approximated by a weight matrix. Using the direct (*gradient descent*) approach, each weight w in the matrix W is modified by the quantity Δw:

$$\Delta w = \beta(r + \gamma \max_{a \in A} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w}, \quad (11)$$

where the gradient component $\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w}$ is used to guide the weight update.

As seen above, XCS exploits a modification of Q-learning, updating at each time step t, given current input s_t and current reward r, the prediction estimates of each classifier in the action set $[A]_{t-1}$ of the previous time step by

$$p \leftarrow p + \beta(r + \gamma \max_{a \in [A]} P(a) - p). \quad (12)$$

We can note that while tabular Q-learning only updates one value at each learning iteration (updating $Q(s_{t-1}, a_{t-1})$), XCS updates all classifiers in the action set $[A]_{t-1}$. In fact, each position in the Q-table is represented by the corresponding prediction array value (Equation 3). Comparing the weight update for gradient descent (Equation 11) and the update for classifier predictions (Equation 12) we note that in the latter no term plays the role of the gradient. Classifier prediction update for XCS was directly inspired by *tabular* Q-learning [5]. Until now, gradient approaches have not been considered in XCS.

4.2 Adding Gradient Descent to XCS

To improve the learning capabilities of XCS we add gradient descent to the equation for the classifier prediction update in XCS. As noted above, the value of a specific state-action pair is represented by the system prediction $P(\cdot)$, which is computed as a fitness weighted average of classifier predictions (Equation 3). In general, learning classifier systems consider the rules that are active and combine their predictions (their *strength*) to obtain an overall estimate of the reward that should be expected. In this perspective, the classifier predictions play the role of the weights in function approximation approaches. The gradient component for a particular classifier cl_k in the to-be-updated action set $[A]_{t-1}$ can be estimated by computing the partial derivate of $Q(s_{t-1}, a_{t-1})$ with respect to the prediction p_k of classifier cl_k:

$$\frac{\partial Q(s_{t-1}, a_{t-1})}{\partial w} = \frac{\partial}{\partial p_k}\left[\frac{\sum_{cl_j \in [A]_{t-1}} p_j F_j}{\sum_{cl_j \in [A]_{t-1}} F_j}\right] =$$

$$= \frac{1}{\sum_{cl_j \in [A]_{t-1}} F_j}\frac{\partial}{\partial p_k}\left[\sum_{cl_j \in [A]_{t-1}} p_j F_j\right] = \frac{F_k}{\sum_{cl_j \in [A]_{t-1}} F_j}. \tag{13}$$

Thus, for each classifier the gradient descent component corresponds to its relative contribution (measured by its current relative fitness) to the overall prediction estimate.

To include the gradient component in XCS's classifier prediction update mechanism, prediction p_k of each classifier $cl_k \in [A]_{t-1}$ is now updated using

$$p_k \leftarrow p_k + \beta(r + \gamma \max_{a \in A} P(a) - p_k)\frac{F_k}{\sum_{cl_j \in [A]_{t-1}} F_j}. \tag{14}$$

The other parameters are updated as usual (see Section 3). In the remainder of the paper we refer to the version of XCS with the prediction updated based on gradient descent as XCSG.

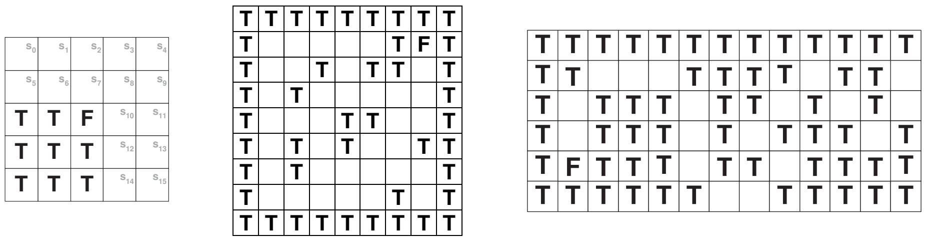
Due to the contribution-weighted gradient-based update, the estimate of the payoff surface, that is, the approximation of the optimal action-value function Q^*, becomes more reliable. As a side effect, the evolutionary component of XCS can work more effectively since the classifier parameter estimates are more accurate. The next section validates this supposition.

5 Experimental Validation

To evaluate XCSG, we compare its performance to XCS in several typically used maze environments. The experimental setup distinguishes between learning problems and test problems as has been usually done in the literature [5]. In learning problems, the system selects actions randomly from those represented in the match set and applies all learning mechanisms. In test problems, the system always selects the action with highest prediction and applies parameter updates and covering only. Learning problems and test problems alternate. In the investigated multistep problems, performance is computed as the average number of steps needed to reach the goal position averaged over the last 50 test problems. If the goal is still not reached after the execution of 1500 steps in one problem, the next problem begins. All results reported in this paper are averaged over 20 experiments.

5.1 The `Woods1` Environment

First we apply XCSG to `Woods1` (shown in Figure 1a) and compare XCSG's performance with that of XCS. Since `Woods1` is very simple we do not expect



(a) (b) (c)

Fig. 1. Environments (a) `Woods1`; (b) `Maze6`; (c) `Woods14`. `Woods1` is a toroid. Obstacles (trees) are denoted by T. The goal (food) is denoted by F. Perceived are the eight neighboring positions starting north and coding clockwise. Actions are possible to the eight neighboring positions. An action that leads to a position with an obstacle has no effect.

much difference in the performance of the two systems[1]. Figure 2 reports the performance of XCS (dashed line) and XCSG (solid line); as expected there is almost no difference between the two versions, the problem is very simple, and both algorithms reach optimality. A closer look at the very beginning of the experiments shows that XCS learns actually somewhat faster than XCSG. Since the gradient approach effectively decreases the overall update rate of the reward prediction, learning is slightly delayed.

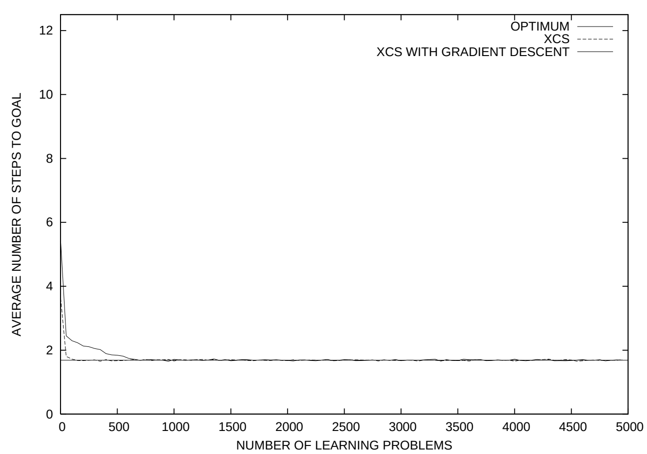


Fig. 2. The performance of XCS (dashed line) and XCSG (solid line) in `Woods1`.

[1] In `Woods1`, parameters were set as follows: $N = 1600$, $P_{\#} = 0.6$, β=0.2, γ=0.7, χ=0.8, μ=0.04, θ_{nma}=8, $p_{\text{explr}} = 1.0$, θ_{GA}=25, ε_0=10, θ_{del}=20. Subsumption is not applied.

5.2 The `Maze6` Environment

The `Maze6` environment (shown in Figure 1b) was shown to be a hard problem for XCS [7]. Figure 3 reports the performance of XCS (dashed line) and XCSG (dotted line) in `Maze6` [2]; note that XCS performance is quite far from the optimum while XCSG reaches the optimum rapidly and stably. The analysis of single runs shows that XCS cannot reach optimal performance in many runs. In contrast, XCSG always reaches full optimality. This suggests that the improvement in XCS performance provided by gradient descent becomes more and more relevant as the problem complexity increases.

To assure that the gradient approach is effective not only due to the reduced update rate of the reward prediction estimate, we also compare the performance of XCSG with that of a modified version of XCS in which the update of classifier prediction is based on a learning rate β_P smaller than the actual learning rate β used for the update of the classifier prediction error and classifier fitness. Figure 3 reports the performance of XCS with $\beta_P = 0.01$ in `Maze6` (solid line). Albeit a smaller learning rate β_P does improve XCS performance, the overall improvement is far from that of XCSG. Due to the distinct update of the reward prediction measure in the gradient approach, the reward prediction of overgeneral (and thus low-fitness) classifiers fluctuates less preventing prediction overestimations and error underestimations.

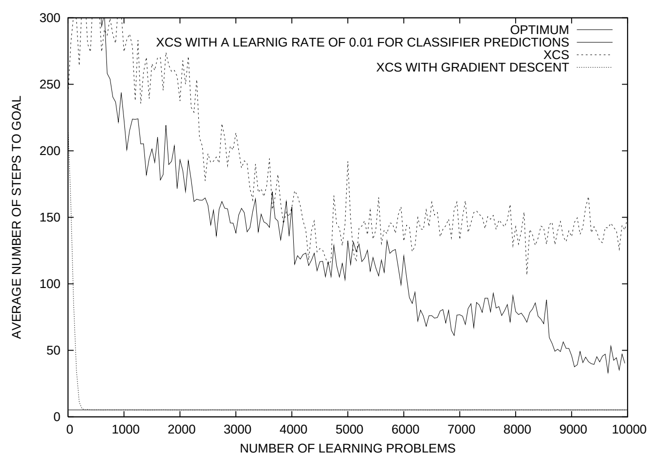


Fig. 3. The performance of XCS with $\beta_P = 0.01$ (solid line), XCS (upper dashed line), and XCSG (lower dashed line) in `Maze6`.

---

[2] Parameter setting in the `Maze6` environment: $N = 3000$, $P_{\#} = 0.3$, β=0.2, γ=0.7, χ=0.8, μ=0.01, θ_{nma}=8, $p_{\mathrm{explr}} = 1.0$, θ_{GA}=100, ε_0=1, θ_{del}=20. Subsumption is not applied.

5.3 The Woods14 Environment

Woods14 is a particular hard problem for XCS because a long chain of classifiers needs to be evolved and maintained to reach the goal optimally [8]. Due to the large number of steps to the food, the prediction error ε_0 must be set small enough to allow XCS to distinguish among small payoff values.[3] To speed up the experiments we reduced the maximum number of steps per problem to 500 steps.

Figure 4 reports the performances of XCS (dashed line) and XCSG (solid line) in Woods14 [4]. XCS does not reach even near optimal performance in any of the 20 runs. In contrast, XCSG reaches the optimum rapidly and stably. The analysis of single runs shows that in 18 of the 20 runs XCSG was able to reach optimal performance; in the remaining two runs, XCSG was able to learn the optimal policy for Woods14 for all the positions except the last one at the end of the corridor, that is, the position farthest from the goal.

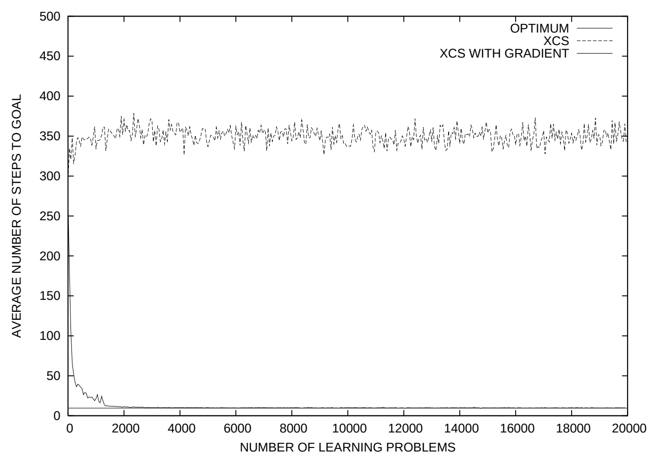


Fig. 4. The performance of XCSG (solid line) and XCS (dashed line) in Woods14 when $N = 4000$.

6 Summary and Conclusions

This paper showed that the addition of a gradient-based update mechanism in XCS results in more stable and reliable learning. Hard multistep problems, such as Maze6, which allows only few generalizations, and Woods14, which requires a

[3] To be able to reach optimal performance XCS must be able to distinguish between two payoff values: $1000\gamma^{18}$ and $1000\gamma^{19}$; accordingly, ε_0 must be smaller than $1000(1-\gamma)\gamma^{18}/2 = 0.245$.

[4] Parameters were set as follows in Woods14: $N = 4000$, $P_{\#} = 0.3$, β=0.2, γ=0.7, χ=0.8, μ=0.01, θ_{nma}=8, $p_{\text{explr}} = 0.3$, θ_{GA}=400, ε_0=0.05, θ_{del}=20. Subsumption was not applied.

long classifier chain, were both solved reliably. Further investigations with various population sizes N and other GA thresholds θ_{GA} confirmed the robustness of the mechanism [17].

Interestingly, in the investigated problems, stability was reached without any residual update addition as it is necessary in neural-network type Q-value function approximators (see Section 2.2). Residual methods were added in [17] but did not improve XCS's performance in the investigated problems.

XCS, and learning classifier systems in general, appear to be somewhat in between tabular RL mechanisms and neural-based RL mechanisms. Previous comparison to neural-network learning mechanisms have emphasized the co-adaptivity in LCSs and the genetic learning component [18]. Our investigations show that even more importantly the representation significantly differs: XCS (as our exemplar system) estimates reward not only by one rule, as in a tabular learning approach, and also not by all rules, as in a neural-network approach, but by a subset of matching rules. Thus, residual methods may not be necessary in LCSs because only the matching subset of classifiers is used for the estimation of a Q-value. The distinction of successive Q-values is realized by classifier conditions instead of classifier weights. Since the evolutionary component evolves the conditions, the distinction of successive Q-values is realized by the evolutionary component and not by the reinforcement component in LCSs.

The successful application of neural-network type mechanisms suggests further comparisons between XCS, LCSs, and neural-network learning and function approximation mechanisms. Classifier conditions, evolved by an evolutionary mechanism, effectively identify independent problem subspaces. This property appears to be highly similar to the development of kernel methods in neural-networks, in which different kernel types result in different spatial separations. The extension of LCSs to kernel-based condition parts appears to be an interesting future research direction. The real-value extension of XCS may be seen as a first step in this direction [1, 12]. Future research will show in which problems the more explicit, rule-based spatial problem separation in conjunction with an evolutionary learning component in LCSs may be advantageous in comparison with neural-network type learning mechanisms.

Acknowledgment

We are grateful to Xavier Llorà, Kei Onishi, Kumara Sastry, and the whole IlliGAL lab for their help and the useful discussions. The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Additional funding from the German research foundation (DFG) under grant DFG HO1301/4-3 is acknowledged. Additional support from the Computational Science and Engineering graduate option program (CSE) at the University of Illinois at Urbana-Champaign is acknowledged. 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 or the U.S. Government.

References

1. Wilson, S.W.: Get real! XCS with continuous-valued inputs. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Learning classifier systems: From foundations to applications. Springer-Verlag, Berlin Heidelberg (2000) 209–219
2. Bernadó, E., Llorà, X., Garrell, J.M.: XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001. Springer-Verlag, Berlin Heidelberg (2002) 115–132
3. Lanzi, P.L.: Mining interesting knowledge from data with the XCS classifier system. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 958–965
4. Dixon, P.W., Corne, D.W., Oates, M.J.: A preliminary investigation of modified XCS as a generic data mining tool. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001. Springer-Verlag, Berlin Heidelberg (2002) 133–150
5. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175
6. Wilson, S.W.: Generalization in the XCS classifier system. Genetic Programming 1998: Proceedings of the Third Annual Conference (1998) 665–674
7. Lanzi, P.L.: An analysis of generalization in the XCS classifier system. Evolutionary Computation **7** (1999) 125–149
8. Barry, A.M.: The stability of long action chains in XCS. Journal of Soft Computing **6** (2002) 183–199
9. Watkins, C.J.C.H.: Learning from Delayed Rewards. PhD thesis, King's College, Cambridge, UK (1989)
10. Wilson, S.W.: ZCS: A zeroth level classifier system. Evolutionary Computation **2** (1994) 1–18
11. Lanzi, P.L.: Learning classifier systems from a reinforcement learning perspective. Soft Computing: A Fusion of Foundations, Methodologies and Applications **6** (2002)
12. Wilson, S.W.: Function approximation with a classifier system. Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 974–981
13. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge, MA (1998)
14. Baird, L.C.: Residual algorithms: Reinforcement learning with function approximation. Machine Learning: Proceedings of the Twelfth International Conference (1995)
15. Baird, L.C.: Reinforcement Learning Through Gradient Descent. PhD thesis, School of Computer Science. Carnegie Mellon University, Pittsburgh, PA 15213 (1999)
16. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. Soft Computing **6** (2002) 144–153
17. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Gradient Descent Methods in Learning Classifier Systems. IlliGAL report 2003028, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2003)
18. Smith, R.E., Cribbs, H.B.: Is a learning classifier system a type of neural network? Evolutionary Computation **2** (1994) 19–36