Standard and Averaging Reinforcement Learning in XCS

Pier Luca Lanzi
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano, I-20133, Milano, Italy
Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign,
Urbana, IL 61801, USA
lanzi@elet.polimi.it

Daniele Loiacono
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano, I-20133, Milano, Italy
loiacono@elet.polimi.it

ABSTRACT

This paper investigates reinforcement learning (RL) in XCS. First, it formally shows that XCS implements a method of generalized RL based on linear approximators, in which the usual input mapping function translates the state-action space into a niche relative fitness space. Then, it shows that, although XCS has always been related to standard RL, XCS is actually a method of averaging RL. More precisely, XCS with gradient descent can be actually derived from the typical update of averaging RL. It is noted that the use of averaging RL in XCS introduces an intrinsic preference toward classifiers with a smaller fitness in the niche. It is argued that, because of the accuracy pressure in XCS, this results in an additional preference toward specificity. A very simple experiment is presented to support this hypothesis. The same approach is applied to XCS with computed prediction (XCSF) and similar conclusions are drawn.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance.

Keywords

LCS, XCS, RL, Gradient Descent.

1. INTRODUCTION

Reinforcement learning (RL) deals with the problem of an agent that has to learn to perform a certain task by interacting with an unknown environment. The agent does not know anything about the structure of the environment, it only knows the current environmental state s. It can perform an action a selected from a set of available actions, and as a consequence of performing a in s it receives a reward r, an indication of how well the agent is behaving in terms of problem solution. In reinforcement learning, the agent learns by trying to maximize the amount of reward received. To accomplish this, the agent usually computes a value function $Q(s,a)$ which maps state-action pairs into an estimate of expected cumulative future reward. Reinforcement learning algorithms work on two main assumptions: that $Q(s,a)$ is represented as a lookup table and that each state-action pair is visited an infinite number of times. However, in large problems lookup tables become easily infeasible, both because of the memory requirements, and because it is often impossible to visit every state-action pair an infinite number of times. This introduces the problem of generalization: how to compute a compact approximation of $Q(s,a)$, while reusing previous experience in those areas of the problem space that are rarely visited. Generalization in reinforcement learning is usually solved by methods of function approximation and the value function $Q(s,a)$ is approximated by a parametrized function.

Learning classifier systems are a method of reinforcement learning which provides a different approach to generalization. In learning classifier systems the value function $Q(s,a)$ is represented by a set of possibly overlapping condition-action-prediction rules, called classifiers, which associate to the problem subspace (identified by the condition) and to the classifier action an expected payoff. Classifiers applying in the same state s, whose condition matches s, and advocating the same action a combine their predictions to provide an estimate of $Q(s,a)$. Effective generalization is obtained by evolving populations of maximally general classifiers that apply in as many situations as possible while providing a good approximation of $Q(s,a)$. Classifier prediction has been usually represented as a parameter associated to the classifier but recently Wilson [27] has introduced the concept of computed prediction. The typical prediction parameter is replaced by a prediction function $p(s,\mathbf{w})$ and a parameter vector $\mathbf{w}$. When a state s is encountered, the predictions of the classifier applying in s and having action a are first *computed* and then combined to provide an approximation of $Q(s,a)$. Computed prediction introduces a new dimension to the quest for effective generalization, while the genetic algorithm still searches for the best problem partitioning, computed prediction improves the approximation of $Q(s,a)$.

The study of the relations between learning classifier systems and reinforcement learning have been widely studied.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

In the recent years, these works have been mainly focused on Wilson's XCS [11, 5, 9, 20]. This most probably because XCS has a stronger relation to RL than other models, in that it uses a modification of Watkins's Q-learning [16] to update the classifier prediction parameter. In particular, it has been shown that XCS can be formally equated to a rule-based version of Q-learning [11] in which the genetic algorithm is used as the search engine for the best problem partitioning. Later, in [5], XCS was compared to (standard) generalized Q-learning and it was noted that the typical gradient term is missing for the classifier prediction update. The gradient was added to XCS and it was shown to improve XCS performance in multistep problems. The same approach was also applied to XCS with computed prediction (XCSF) in [12], but no relevant improvements were reported in this case. The analysis in [5] was extended by Wada et al. [20, 19] where it was noted that the derivation of gradient descent provided in [5] was not consistent with standard generalized Q-learning. Accordingly, another update was proposed [20] and tested on a strength based XCS [9]. However the results did not show improvements over the original XCS.

In this paper, we further investigate the relationships between XCS models and generalized reinforcement learning. First, we show that both XCS and XCS with computed prediction (XCSF) can be formally viewed as methods of generalized reinforcement learning based on linear approximation. We consider the two major approaches to generalized reinforcement learning, standard RL [16] and averaging RL [16, 8, 14], which was not considered in the comparisons performed in previous studies. Using the equivalence between XCS models and linear reinforcement learning, we derive the update formula for XCS and XCSF for standard and averaging RL. Standard RL in XCS and XCSF results in the XCS update introduced in [20] and in a similar update for XCSF. We show that XCS and XCSF with gradient descent [5, 12] actually implement averaging RL and not standard RL as previously argued in [5]. We also note that averaging RL in XCS and XCSF introduces, in the minimization of the overall prediction error, an additional gradient term which potentially favor classifiers with a small fitness niche-wise. We suggest that the interaction between this additional gradient term and the accuracy based fitness introduces an intrinsic preference toward specific classifiers. Experimental assessment is not a goal of this work. Accordingly, we present two illustrative experiments involving XCS and XCSF with standard and averaging RL appears to support the hypotheses we drawn from the theoretical derivations.

2. REINFORCEMENT LEARNING

In reinforcement learning an *agent* learns to perform a task through *trial and error interactions* with an unknown *environment* which provides feedback in terms of numerical *reward* [16]. At time t the agent senses the environment to be in state s_t; based on its current sensory input s_t the agent selects an action a_t which it then performs in the environment. Depending on the state s_t, on the action a_t performed, and on the effect of a_t 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 amount of reward received or *expected payoff* [16]. 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 ($\gamma \in [0, 1)$).

2.1 Generalized Reinforcement Learning

Reinforcement learning assumes that the action-value function is represented as a *look-up table* with one entry for each state-action pair. However, look-up tables are infeasible in problems involving many states since they require more memory and, most important, more on-line experience to converge. To tackle large problems an agent must be able to *generalize* over its experiences, i.e., to produce accurate approximations of the optimal action-value function from a limited number of experiences, possibly using a small amount of storage. In reinforcement learning generalization is typically implemented by methods of function approximation: the action-value function $Q(s, a)$ is not represented as a table but approximated by a function f parametrized with a vector $\boldsymbol{\theta}$. These methods are also characterized by an input mapping $\boldsymbol{\phi}(\cdot)$ that translates the state-action space to a feature space. For instance, in tile coding [16] $\boldsymbol{\phi}(\cdot)$ maps a continuous state into a binary vector representing the state membership to a set of overlapping tiles. Thus, the action-value function $Q(s, a)$ is approximated by a function f of $\boldsymbol{\phi}(\cdot)$ and $\boldsymbol{\theta}$:[1]

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

The problem of learning $Q(s, a)$ thus translates into the problem of estimating the parameter vector $\boldsymbol{\theta}$. This is usually implemented by gradient descent: at time t, $\boldsymbol{\theta}$ is modified following, with step β_t, the *direction* that reduces the error E_t in f, i.e.:

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

The typical update of the action-value function $Q(s, a)$ is thus replaced by the update of $\boldsymbol{\theta}$.

2.2 Linear Methods

Linear methods are probably the most important class of approximators used in reinforcement learning [3, 15, 16]. 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 as,

$$f(\boldsymbol{\phi}(s, a), \boldsymbol{\theta}) = \phi(s, a)\boldsymbol{\theta}. \quad (4)$$

Most of the linear methods in reinforcement learning define, at time t, the error function E_t as,

$$E_t = \frac{1}{2}\left(\hat{Q}(s_t, a_t) - f(\boldsymbol{\phi}(s_t, a_t), \boldsymbol{\theta})\right)^2 \quad (5)$$

where $\hat{Q}(s_t, a_t)$ is the new estimate of the expected payoff for performing action a_t in s_t (Equation 1). Given the above error function, gradient descent updates each parameter θ_i as,

$$\theta_i \leftarrow \theta_i + \beta(\hat{Q}(s, a) - \boldsymbol{\phi}(s, a)\boldsymbol{\theta})\phi_i(s, a), \quad (6)$$

[1]Note that, based on the problem, either $\boldsymbol{\theta}$ or $\boldsymbol{\phi}(\cdot)$ may also be function of the action a. However, without lacking in generality, the action may be omitted from the notation [14].

where $\phi_i(s,a)$ is the i-th element of the feature vector $\boldsymbol{\phi}(s,a)$, i.e., the gradient of $f(\boldsymbol{\phi}(s_t,a_t),\boldsymbol{\theta})$ with respect to θ_i. Equation 6, due to Widrow-Hoff [22] and known as Delta Rule or Least Mean Square rule, can be shown to find, under adequate assumptions, a local minima of

$$\sum_i \frac{1}{2}\left(\hat{Q}(s_i,a_i) - f(\boldsymbol{\phi}(s_i,a_i),\boldsymbol{\theta})\right)^2, \tag{7}$$

that is the sum, over all the steps i, of the error function in Equation 5.

This *standard* updating scheme [24] (or LMS updating [14]) has been rather successful in practice [15, 23] and it has been proved to converge for several reinforcement learning scheme, e.g., TD(λ) [16, 17]. But it can lead to the divergence of the range of f when applied to other learning scheme such as value-iteration and Q-learning (our focus here) [1, 18]. And even TD(0) can be shown to diverge under adequate assumptions [18]. There is actually a general problem affecting reinforcement learning with function approximation: the training examples fed into the approximator are not independent from the approximator output. When $\boldsymbol{\theta}$ is adjusted to minimize the error in a state s it is possible that the same adjustment increases the error in other areas of the state space, usually related to s in typical multistep problems; so that this error can be fed back into subsequent updates and it can cause the range of f to diverge to infinity. To deal with this issue a number of methods have been introduced (see [14] for a review). Among the others *linear averagers* [8] exploit a different error function and therefore a different updating scheme which is always guaranteed not to diverge [8], although it may converge to a non optimal action-value function [24].

Linear averagers [8] minimize the error

$$E_t = \frac{1}{2}\sum_i \left(\hat{Q}(s_t,a_t) - \theta_i\right)^2 \phi_i(s_t,a_t) \tag{8}$$

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,

$$\theta_i \leftarrow \theta_i + \beta_i(\hat{Q}(s_t,a_t) - \theta_i))\phi_i(s_t,a_t), \tag{9}$$

which minimizes the difference between the state value and the *actual parameter*, instead of the approximated value $f(\boldsymbol{\phi}(s_t,a_t),\boldsymbol{\theta})$ in Equation 6, and overall minimizes,

$$\frac{1}{2}\sum_t\sum_i \left(\hat{Q}(s_t,a_t) - \theta_i\right)^2 \phi_i(s_t,a_t), \tag{10}$$

Linear averagers belong to a broader class of reinforcement learning algorithms, called averagers, which includes nearest neighbor methods, kernels, and local weighted regression, but not popular methods such as tile-coding and backpropagation [16, 8, 13]. The area of reinforcement learning devoted to the study of averagers is usually referred to as *averaging reinforcement learning*.

3. THE XCS CLASSIFIER SYSTEM

XCS works as a typical reinforcement learning algorithm though it is based on a rule based representation [11]. In XCS, classifiers consist of a condition, an action, and four main parameters [25, 7]: (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_{nma} 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.[2] The system prediction $P(s_t,a)$ is computed as the fitness weighted average of the predictions of classifiers in [M] which advocate action a:

$$P(s_t,a) = \sum_{cl_k\in[M](a)} p_k \times \frac{F_k}{\sum_{cl_i\in[M](a)} F_i} \tag{11}$$

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. Then XCS selects an action to perform; the classifiers in [M] which 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:

$$P(t) = r_{t+1} + \gamma \max_{a\in[M]} P(s_{t+1},a) \tag{12}$$

Next, the parameters of the classifiers in [A] are updated in the following order [7]: prediction, prediction error, and finally fitness. Prediction p is updated with learning rate β ($0 \le \beta \le 1$):

$$p \leftarrow p + \beta(P(t) - p) \tag{13}$$

Then, the prediction error ϵ and classifier fitness are updated as usual [25, 7]. On regular basis (dependent on parameter θ_{ga}), the genetic algorithm is applied to classifiers in [A]. It selects two classifiers with probability *proportional to their fitnesses*, 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 studied in the literature [11, 5, 20, 19]. The first analysis in [11] shows that (i) the system prediction $P(s,a)$ in Equation 11 actually represents the value of $Q(s,a)$ using the classifiers in [A]; while (ii) the prediction value of $P(t)$ in Equation 12 actually represents the new estimate $\hat{Q}(s,a)$. The comparison was extended in [5] to the case of generalized reinforcement learning. In [5], it is noted that (i) the classifier prediction in XCS corresponds to the parameters θ_i in Equation 2; (ii) although XCS implements generalized

[2]System prediction is usually denoted as $P(a)$ in XCS and as $P(s_t,a)$ in XCSF; here we always use the latter to emphasize that system prediction depends on the current input s_t since the match set [M] is actually a function of s_t.

Q-learning, its prediction update (Equation 13) does not include the gradient term; (iii) the gradient term for classifier cl_k is computed as,

$$\frac{\partial P(s_t, a_t)}{\partial cl_k} = \frac{F_k}{\sum_{[A]} F_i}. \quad (14)$$

Finally, in [5] it is also shown that the addition of the gradient component improves XCS performance over a set multistep problems; although more advanced gradient descent techniques (such as residual gradient) seem not to cause a similar improvement.

Later, [20] compared ZCS and XCS to Q-learning with function approximation. In [20] it is argued that the first comparison presented in [5] is not completely correct. In fact, [20] notes that the Q-learning update adjusts $Q(s,a)$ using the difference between the new estimate $\hat{Q}(s,a)$ and the current estimate $Q(s,a)$. But, XCS adjusts each classifier prediction (which contributes to $Q(s,a)$, but does not represent it) according to the difference between the new estimate of $\hat{Q}(s,a)$ and the current classifier prediction. To provide a fair comparison between Q-learning and XCS, [20] proposed a different classifier prediction update,

$$p \leftarrow p + \alpha(P(t) - P(s_t, a_t))\frac{\partial P(s_t, a_t)}{\partial cl_k} \quad (15)$$

which is more similar to the Q-learning update. However, this type of update did not produce improvements as those reported for the gradient in [5]. It is essential to note that neither [20] nor [5] distinguish between standard and averaging RL. So that while the two approaches might be regarded as radically different, as we show in this paper, they just refer to two different approaches to generalized reinforcement learning.

5. XCS AS LINEAR RL

We now show that XCS actually is a method of generalized reinforcement learning in which the input mapping $\boldsymbol{\phi}(\cdot)$ translates the state-action space into a relative fitness landscape. First, we define two functions,

$$C_i(s_t) = \begin{cases} 1 & \text{if } cl_i.C \text{ matches } s_t \\ 0 & \text{otherwise} \end{cases}$$

$$A_i(a_t) = \begin{cases} 1 & \text{if } cl_i.a = a_t \\ 0 & \text{otherwise} \end{cases}$$

so that system prediction $P(s_t, a_t)$ in Equation 11 can be rewritten as,

$$P(s_t, a_t) = \sum_{[P]} p_k \times \frac{C_k(s_t)A_k(a_t)F_k}{\sum_{[P]} C_j(s_t)A_j(a_t)F_j}. \quad (16)$$

Given the correspondence (i) between classifier predictions and the parameters θ_i [5, 20] and (ii) between system prediction $P(s_t, a_t)$ and the action value function $Q(s,a)$ [11, 20], then by defining,

$$\phi_k(s_t, a_t) = \frac{C_k(s_t)A_k(a_t)F_k}{\sum_{[P]} C_j(s_t)A_j(a_t)F_j}, \quad (17)$$

and defining $\boldsymbol{\theta}$ as the vector containing the predictions of all the N classifiers in the population,

$$\boldsymbol{\theta} = \langle p_0, p_1, \ldots, p_N \rangle, \quad (18)$$

we have that the system prediction $P(s_t, a_t)$ is computed as,

$$P(s_t, a_t) = \boldsymbol{\phi}(s_t, a_t)\boldsymbol{\theta}.$$

Thus, the system prediction in XCS (Equation 16) implements the typical linear reinforcement learning approach (Equation 4). In this case, the input mapping function $\boldsymbol{\phi}(s_t, a_t)$ maps the state action pair $\langle s_t, a_t \rangle$ into the gradient components of the classifiers $cl_k \in [A]$; these gradient components are a measure of the fitness contribution of cl_k to [A] (Equation 14). Therefore, XCS can be viewed as a method of generalized reinforcement learning based on linear approximators. In this perspective, XCS is similar to tile coding [15]. In tile coding, the function $\boldsymbol{\phi}(s_t, a_t)$ maps the state-action space into a set of t overlapping *tilings*. In XCS, the function $\boldsymbol{\phi}(s_t, a_t)$ maps the state-action space into a gradient landscape. Note that, a similar discussion can be also found in [2, Section 2] where equations 11 and 16 are related to softstate aggregation [16].

6. REINFORCEMENT LEARNING IN XCS

In this section, we consider the two classes of approaches to generalized reinforcement learning, (the standard one in Equation 7 and the averaging one in Equation 10) and we derive the corresponding classifier prediction update. We show that while standard reinforcement learning maps into the most recent gradient descent approach discussed in [20], averaging reinforcement learning maps into the first gradient descent approach introduced in [5]. Most important, we show that the latter approach, based on averaging reinforcement learning, actually causes an intrinsic preference toward specificity.

6.1 Standard RL in XCS

To derive the classifier prediction update for standard reinforcement learning, we rewrite the error function in Equation 7 using XCS notation as,

$$\frac{1}{2}\sum_t (P(t) - P(s_t, a_t))^2 \quad (19)$$

which corresponds to the following error function E_t to be minimized at time t,

$$E_t = \frac{1}{2}(P(t) - P(s_t, a_t))^2. \quad (20)$$

So the gradient term for the update of $\boldsymbol{\theta}$ in Equation 3 (i.e., classifier prediction in XCS [5, 20]) can be rewritten as,

$$\begin{aligned} \frac{\partial E_t}{\partial \theta_i} &= -(P(t) - P(s_t, a_t))\frac{\partial P(s_t, a_t)}{\partial \theta_i} \\ &= -(P(t) - P(s_t, a_t))\frac{\partial(\sum_k \theta_k \phi_k(s_t, a_t))}{\partial \theta_i} \\ &= -(P(t) - P(s_t, a_t))\phi_i(s_t, a_t) \end{aligned}$$

while the parameter update is,

$$\theta_i \leftarrow \theta_i + \beta(P(t) - P(s_t, a_t))\phi_i(s_t, a_t) \quad (21)$$

which in XCS corresponds, at time t, to the following classifier prediction update for classifiers $cl_i \in$[A],

$$p_i \leftarrow p_i + \beta(P(t) - P(s_t, a_t))\frac{F_i}{\sum_{[A]} F_j}, \quad (22)$$

firstly introduced in [20].

6.2 Averaging RL in XCS

We now apply the same approach to the case of averaging reinforcement learning. We begin by rewriting Equation 10 in XCS notation,

$$\frac{1}{2}\sum_t\sum_j \phi_j(s_t,a_t)(P(t)-\theta_j)^2 \quad (23)$$

which at time t leads us to the minimization of the following error,

$$E_t = \frac{1}{2}\sum_j \phi_j(s_t,a_t)(P(t)-\theta_j)^2 \quad (24)$$

so that the gradient term for updating the parameters $\boldsymbol{\theta}$ (Equation 3) is computed as,

$$\frac{\partial E_t}{\partial \theta_i} = -(P(t)-\theta_i)\phi_i(s_t,a_t), \quad (25)$$

the corresponding parameter update is,

$$\theta_i \leftarrow \theta_i + (P(t)-\theta_i)\phi_i(s_t,a_t), \quad (26)$$

so that the update of classifier prediction in XCS is performed, at time t, on all the classifiers in [A], as follows,

$$p_i \leftarrow p_i + \beta(P(t)-p_i)\frac{F_i}{\sum_{[A]}F_j}. \quad (27)$$

This equation corresponds to the formulation of gradient descent for XCS introduced in [5, 6] which actually implements averaging reinforcement learning in XCS.

Most important, it is interesting note that Equation 23 can be actually rewritten as,

$$\frac{1}{2}\sum_t\sum_{[A]_t}\frac{F_i}{\sum_{[A]_t}F_j}(P(t)-\theta_j)^2 \quad (28)$$

Thus, averaging reinforcement learning applied to XCS minimizes, over all the encountered action sets $[A]_t$, the *fitness weighted* difference between the target prediction value $P(t)$ and the predictions of the classifiers in $[A]_t$. Accordingly, since the update based on averaging reinforcement learning minimizes the term in Equation 28, the corresponding prediction update (Equation 27) has an intrinsic preference toward classifiers with a smaller gradient component, i.e., $F_i/\sum_{[A]_t}F_j$. In XCS such classifiers are either overgeneral or overspecific, therefore the gradient component for averaging RL in XCS actually introduces an intrinsic preference toward overgeneral or overspecific classifiers. On the other hand, the accuracy based genetic pressure in XCS provides a pressure toward accurate classifiers so that the interactions between the averaging update and the genetic algorithm should result in an intrinsic pressure toward specificity. This would explain under a different perspective the improvements reported in [5, 6] for the XCS with averaging RL. In fact, such improvements were also reported when the pressure toward specificity was enhanced using additional heuristics [10].

6.3 XCS as RL

We now analyze what type of reinforcement learning does XCS implement. In generalized reinforcement learning, the parameter update minimizes a target error function. Thus, the question is what type of error function the update of classifier prediction in XCS minimizes. For this purpose, we consider the prediction update procedure in XCS, which at time t updates the classifiers cl_i in [A] as,

$$p_i \leftarrow p_i + \beta(P(t)-p_i),$$

and we rewrite it as an update over all the classifiers in [P],

$$p_i \leftarrow p_i + \alpha(P(t)-p_i)C_i(s_t)A_i(a_t)$$

which corresponds to the following derivative term,

$$\frac{\partial E_t}{\partial p_i} = -(P(t)-p_i)C_i(t)A_i(t).$$

and thus to the following error,

$$E_t = \frac{1}{2}\sum_j (P(t)-p_j)^2 C_j(t)A_j(t),$$

so that overall we have that classifier prediction update in XCS minimizes,

$$\frac{1}{2}\sum_t\sum_j (P(t)-p_j)^2 C_j(t)A_j(t)$$

that is,

$$\frac{1}{2}\sum_t\sum_{[A]_t}(P(t)-p_j)^2. \quad (29)$$

Thus, the classifier update in XCS minimizes, over all the encountered action sets $[A]_t$, the difference between the target prediction value $P(t)$ and the prediction of classifiers in the action set $[A]_t$. Thus in XCS the prediction update minimizes only the prediction error of the classifiers in each niche and but it does not take into account classifier fitness, introduced when averaging RL is considered.

6.4 An Illustrative Example

As an illustrative example, we apply the three versions of XCS discussed in this section, XCS [25], XCS with prediction update based on standard RL (6.1), and XCS with prediction update based on averaging RL (6.2), to the 20-multiplexer using the standard setting in [26]. All the three models are identical except for the prediction update. Figure 1 compares (a) the performance and (b) the number of macroclassifiers. XCS and XCS with averaging RL reach optimal performance; in contrast XCS with standard RL performs poorly. These results are coherent with those reported in the literature [5, 19]: XCS with averaging RL (also called gradient descent in [5]) performs well; XCS with standard RL performs poorly similarly to the results discussed in [19]. XCS with averaging RL converges slower than XCS and in the beginning it evolves larger populations, containing classifiers that on the average are more specific than those evolved by XCS. This is consistent with the discussion in Section 6.2 where we noted that the prediction update for averaging RL in XCS potentially introduces an intrinsic preference toward more specific classifiers.

7. THE XCSF CLASSIFIER SYSTEM

The introduction of computed prediction requires three simple modifications to XCS. First, classifier prediction parameter p is replaced with a parameter vector $\mathbf{w}$, that is used to compute classifier prediction. Second, a prediction function $p(s_t,\mathbf{w})$ is introduced which defines how classifier prediction is computed based on the current input s_t and

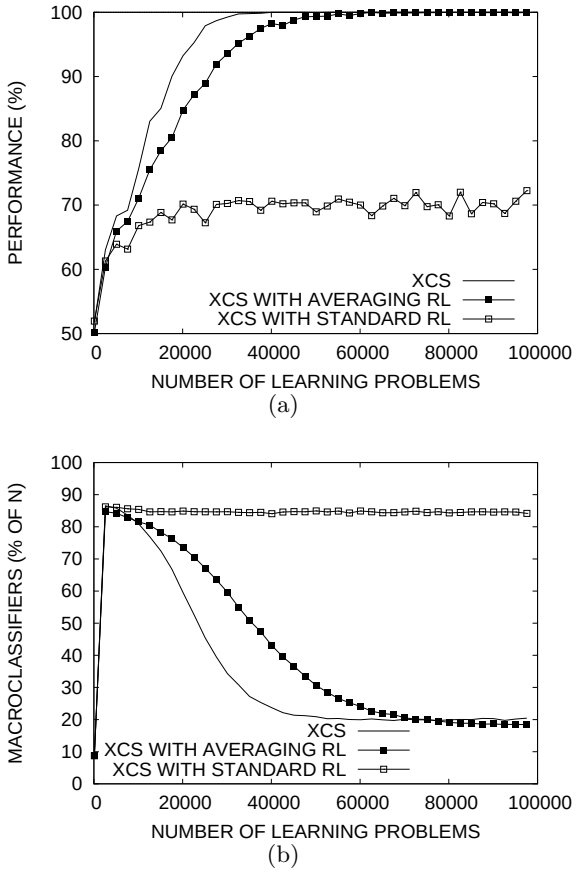

Figure 1: The three versions of XCS applied to the 20-multiplexer: (a) performance; (b) % of macroclassifiers. Curves are averages over 20 runs.

parameter vector $\mathbf{w}$. Finally, the update of classifier prediction (Equation 13) is replaced by the update of $\mathbf{w}$ based on an estimated payoff value $P(t)$ and the current classifier prediction value $p(s_t, \mathbf{w})$. The system prediction for action a is now defined at time t as,

$$P(s_t, a) = \sum_{cl_k \in [M](a)} p(s_t, \mathbf{w}_k) \times \frac{F_k}{\sum_{cl_i \in [M](a)} F_i} \quad (30)$$

where $\mathbf{w}_k$ is the parameter vector associated to classifier cl_k, F_k is the fitness of cl_k, and [M](a) is the subset of classifiers in [M] with action a. Classifier prediction $p(s_t, \mathbf{w}_k)$ is usually computed as the linear combination of the input vector s_t and the parameter vector $\mathbf{w}_k$, i.e., $s_t \mathbf{w}_k$. The expected payoff $P(t)$ is computed as:

$$P(t) = r_{t+1} + \gamma \max_{a \in A} P(s_{t+1}, a) \quad (31)$$

The expected payoff $P(t)$ is used to update the weight vector $\mathbf{w}$ of the classifier in [A] using a *modified delta rule* as,

$$\mathbf{w}_k(i) \leftarrow \mathbf{w}_k(i) + \frac{\eta}{|s_t|^2}(P - p(\mathbf{s}_t, \mathbf{w}_k))s_t(i) \quad (32)$$

where $\beta(t)$ is computed as $\eta/|s_t|^2$, η is the correction rate, and $|\mathbf{s}_t|^2$ is the norm of the input vector $\mathbf{s}_t$ [27].

8. REINFORCEMENT LEARNING IN XCSF

We now apply the same approach used for XCS to XCSF. First, we show that also XCSF can be viewed as a method of generalized reinforcement learning based on linear approximation. Then, we derive the update for standard and averaging reinforcement learning in XCSF. Finally, we derive the error function that the prediction update in XCSF minimizes.

8.1 XCSF as Linear RL

The system prediction for XCSF (Equation 30) can be rewritten as,

$$P(s_t, a_t) = \sum_{[P]} p(s_t, \mathbf{w}_k) \frac{C_k(s_t) A_k(a_t) F_k}{\sum_{[P]} C_j(s_t) A_j(a_t) F_j},$$

which is an example of reinforcement learning based on averagers: the system prediction is computed as the fitness weighted average of a set of local linear approximators, i.e., the functions $p(s_t, \mathbf{w}_k)$. By defining, $\pi_k(t) = p(s_t, \mathbf{w}_k)$ and,

$$\boldsymbol{\theta}_t = \langle \dots, \pi_k(t), \dots \rangle \quad (33)$$

$$\boldsymbol{\phi}(s_t, a_t) = \langle \dots, \frac{F_k C_k(s_t) A_k(a_t)}{\sum_{[P]} C_j(s_t) A_j(a_t) F_j}, \dots \rangle \quad (34)$$

where $k \in \{1, \dots N\}$, the system prediction for XCSF can be rewritten as,

$$P(s_t, a_t) = \boldsymbol{\theta} \boldsymbol{\phi}(s_t, a_t),$$

therefore, also XCSF can be viewed as a method of reinforcement learning based on linear approximators. In this case, the input mapping function $\boldsymbol{\phi}(s_t, a_t)$ translates the state-action space into the same fitness based landscape.

8.2 Standard RL in XCSF

To derive the parameter update for standard RL in XCSF we start from the error E_t,

$$E_t = \frac{1}{2}(P(t) - P(s_t, a_t))^2$$

and derive the term for each parameter $\mathbf{w}_k(i)$ of XCSF as,

$$\frac{\partial E_t}{\partial \mathbf{w}_k(i)} = -(P(t) - P(s_t, a_t)) \frac{\partial P(s_t, a_t)}{\partial \mathbf{w}_k(i)}$$

$$= -(P(t) - P(s_t, a_t)) \frac{s_t(i) F_k C_k(s_t) A_k(a_t)}{\sum_{[P]} C_j(s_t) A_j(a_t) F_j}$$

so that standard reinforcement learning update in XCSF corresponds to the following update of the parameters $\mathbf{w}_k(i)$ of the classifiers in [A],

$$\mathbf{w}_k(i) = \mathbf{w}_k(i) + \beta(P(t) - P(s_t, a_t)) s_t(i) \frac{F_k}{\sum_{[A]} F_j}, \quad (35)$$

which is the version for XCSF of the classifier update proposed in [20].

8.3 Averaging RL in XCSF

Given the previous definitions, averaging reinforcement learning in XCSF minimizes,

$$\frac{1}{2} \sum_t \sum_i (P(t) - \pi_i(t))^2 \phi_i(s_t, a_t) \quad (36)$$

that corresponds at time t to the error E_t defined as,

$$E_t = \frac{1}{2}\sum_j (P(t) - \pi_i(t))^2 \phi_i(s_t, a_t)$$

so that,

$$\frac{\partial E_t}{\partial \mathbf{w}_k(i)} = (P(t) - s_t \mathbf{w}_k) s_t(i) \frac{F_k}{\sum_{[A]} F_j}$$

and the update of classifier parameters for averaging reinforcement learning in XCSF is,

$$\mathbf{w}_k(i) = \mathbf{w}_k(i) + \beta(P(t) - s_t \mathbf{w}_k) s_t(i) \frac{F_k}{\sum_{[A]} F_j}. \quad (37)$$

Similarly to the case of XCS, also in XCSF the update for averaging reinforcement learning results in an additional gradient descent term (Equation 14) to the original updated (Equation 32) introduced in [27]. Since averaging reinforcement learning in XCSF minimizes Equation 36, as in XCS, the parameter update introduces an intrinsic preference toward classifiers with a smaller gradient component.

8.4 XCSF as RL

We now start from the update of classifier parameters in XCSF and derive the error that the update minimizes. First, we rewrite Equation 32 as,

$$\mathbf{w}_k(i) \leftarrow \mathbf{w}_k(i) + \beta(P - p(\mathbf{s}_t, \mathbf{w}_k)) s_t(i) C_k(s_t) A_k(a_t). \quad (38)$$

Following the same approach we used for XCS, we derive the following formulation of the overall error in Equation 10 for XCSF,

$$\frac{1}{2}\sum_t \sum_{cl_k \in [A]_t} (P(t) - s_t \mathbf{w}_k)^2. \quad (39)$$

As XCS, XCSF minimizes the difference over all the encountered action sets between the expected payoff $P(t)$ and the classifier prediction, which in this case is computed. The original XCSF update (Equation 32) minimizes Equation 39 which misses the additional gradient term that appears in Equation 37. Thus, the parameter update of XCSF does not introduce the same intrinsic preference toward classifiers with a smaller gradient component that is present in the update for averaging reinforcement learning.

8.5 An Illustrative Example

We now present an illustrative example, as we did for XCS. We apply the three versions of XCSF we discussed in this section, XCSF [27], XCSF with standard RL (8.2), and XCSF with averaging RL (8.3), to a simple problem taken from the XCSF literature, the approximation of the sine function [27]. All the three versions of XCSF are identical to [27] except for the update of classifier parameters which is replaced by the standard (Equation 35) and the averaging update (Equation 37). The parameters are the same for the three systems and they are set as in [27].

Figure 2 compares the prediction error and the average classifier generality for the three models. The generality of a classifier is computed as the fraction of the domain that the condition matches [27]. All the three versions of XCSF evolve accurate solutions: in fact they all reach a prediction error below the error threshold $\epsilon 0$. The prediction error of XCSF with averaging RL converges slower than XCSF;

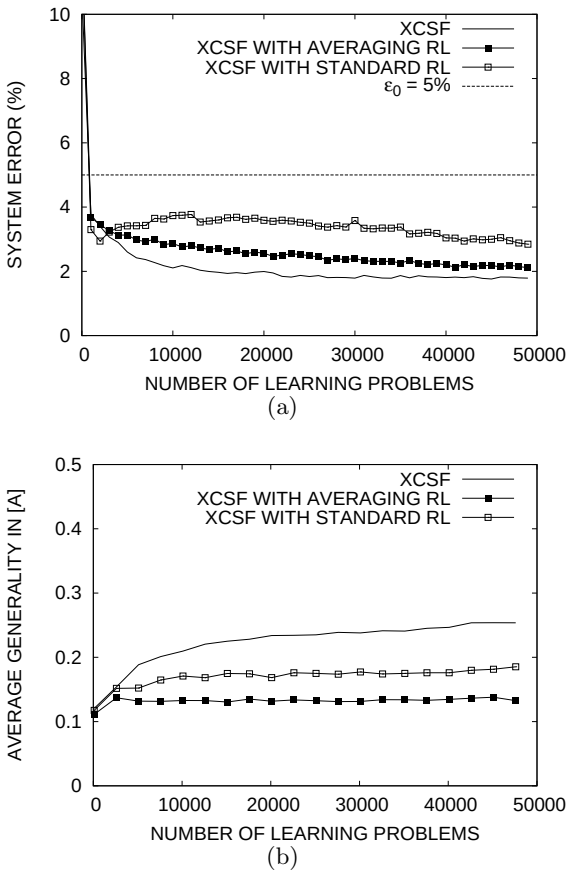

Figure 2: The three versions of XCSF applied to the sine problem: (a) prediction error; (b) average classifier generality. Curves are averages over 10 runs

also the average classifier generality for XCSF with averaging RL is generally smaller than that of XCSF. Again, this result supports the discussion in Section 8.3 where we noted that, as in XCS, averaging RL in XCSF introduces an intrinsic preference toward more specific classifiers. Interestingly, XCSF with standard RL can evolve accurate approximations, whereas the results reported in [19] show that the same type of update (even when the genetic algorithm is not active) can result in poor performance.

9. CONCLUSIONS

In this paper, we have tried to contribute a step further in the understanding of the relations between XCS models and generalized reinforcement learning. We based our work on the previous studies [11, 5, 20, 9] which relate XCS either to tabular reinforcement learning [11] or to standard generalized reinforcement learning [5, 20, 9]. We have shown that both XCS and XCSF can be formally viewed as methods of linear reinforcement learning. We considered the two major approaches to generalized reinforcement learning, the standard one [16] and the averaging one [8], which was not considered in the previous studies. Using the equivalence between XCS models and linear reinforcement learning, we have derived the update formula for XCS models with standard and averaging approaches. We have shown that clas-

sifier update in XCS models [25, 27] actually implements methods of averaging reinforcement learning, not standard one as previously considered. More precisely, averaging RL in XCS models is equivalent to XCS models with gradient descent as introduced in [5, 12]. Using the same equivalence, we noted that the introduction of gradient descent in XCS models results in an intrinsic preference toward classifiers with a small fitness with respect to classifier niche. We suggest that the interaction between this intrinsic preference and the usual accuracy based genetic algorithm results in a preference toward more specific classifiers. Although experimental assessment is not the goal of this paper, we have presented two simple experiments involving XCS and XCSF which support the hypotheses we drawn from the theoretical derivations. But more experiments must be performed as future research direction to validate the contents of this paper further.

10. REFERENCES

[1] L. C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–77. Morgan Kaufman Publishers, San Francisco, CA., July 1995.

[2] L. Booker. Approximating value functions in classifier systems. In Bull and Kovacs [4], pages 45–61.

[3] J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 369–376, Cambridge, MA, 1995. The MIT Press.

[4] L. Bull and T. Kovacs, editors. *Foundations of Learning Classifier Systems*, volume 183 of *Studies in Fuzziness and Soft Computing*. Springer, 2005.

[5] M. Butz, D. G. Goldberg, and P. L. 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, Jan. 2003.

[6] M. V. Butz, D. E. Goldberg, and P. L. Lanzi. Gradient descent methods in learning classifier systems: Improving xcs performance in multistep problems. *IEEE Transaction on Evolutionary Computation*, 9(5):452–473, Oct. 2005.

[7] M. V. Butz and S. W. Wilson. An algorithmic description of xcs. *Journal of Soft Computing*, 6(3–4):144–153, 2002.

[8] G. 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.

[9] T. Kovacs. *Strength or Accuracy: Credit Assignment in Learning Classifier Systems*. Distinguished Dissertations. Springer, 2004.

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

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

[12] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Generalization in the xcsf classifier system: Analysis, improvement, and extension. Technical Report 2005012, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 2005.

[13] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, 2002.

[14] S. I. Reynolds. *Reinforcement Learning with Exploration*. PhD thesis, School of Computer Science. The University of Birmingham, Birmingham, B15 2TT, Dec. 2002.

[15] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1044. The MIT Press, Cambridge, MA., 1996.

[16] R. S. Sutton and A. G. Barto. *Reinforcement Learning – An Introduction*. MIT Press, 1998.

[17] R. S. Sutton, D. Precup, and S. Dasgupta. Off-policy temporal-difference learning with function approximation. In *Proceedings of the 18th International Conference on Machine Learning.*, pages 417–424. Morgan Kaufmann, San Francisco, CA, 2001.

[18] J. N. Tsitsiklis and B. V. Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, May 1997.

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

[20] A. Wada, K. Takadama, K. Shimohara, and O. Katai. Learning classifier systems with convergence and generalization. In Bull and Kovacs [4], pages 285–304.

[21] C. Watkins. *Learning from delayed reward*. PhD thesis, 1989.

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

[23] M. Wiering. *Explorations in Efficient Reinforcement Learning*. PhD thesis, Universiteit van Amsterdam, The Netherlands, Feb. 1999.

[24] M. Wiering. Convergence and divergence in standard and averaging reinforcement learning. In *Proceedings of the 15th European Conference on Machine Learning*, volume 3201, pages 477–488. Springer-Verlag, Berlin, Heidelberg, 2004.

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

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

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