Three Architectures for Continuous Action

Stewart W. Wilson

Prediction Dynamics, Concord MA 01742 USA
Department of General Engineering
The University of Illinois at Urbana-Champaign IL 61801 USA
wilson@prediction-dynamics.com

Abstract. Three classifier system architectures are introduced that permit the systems to have continuous (non-discrete) actions. One is based on interpolation, the second on an actor-critic paradigm, and the third on treating the action as a continuous variable homogeneous with the input. While the last architecture appears most interesting and promising, all three offer potential directions toward continuous action, a goal that classifier systems have hardly addressed.

1 Introduction

Typically, classifiers in learning classifier systems have fixed, discrete actions, e.g., "turn left", "turn right", "yes", "no". Classifiers have not been introduced in which the action is *continuous* in the sense that it depends on and is a continuous function of the input. Continuous actions, however, are desirable in domains where fine reactions and control are important, such as robotics. This paper describes three distinct classifier system architectures that permit continuous actions.

The first architecture, termed "interpolating action learner" (IAL) [12], has one classifier system observing a second classifier system's optimal (discrete) actions and learning them as a smooth function of the input. IAL is perhaps the simplest approach to continuous action. The second approach, "continuous actor-critic" (CAC) is a continuous-action, classifier-system version of the well-known actor-critic architecture [9]. CAC is quite intricate but is a plausible approach to direct learning of optimal continuous actions. The third architecture, a "generalized classifier system" (GCS) [11], breaks new ground by aggregating a continuous action with the input, and learning payoff as a function of both together. In GCS, the optimal action is chosen as the action that maximizes this function. All three architectures make use of a recently introduced function-approximating classifier system called XCSF.

XCSF [14, 15] extends XCS [13, 3], a classifier system with fitness based on prediction accuracy, to real-valued inputs and continuous payoff landscapes. XCSF learns an approximation to $P(x, a_k)$, the environmental payoff function of input x and discrete actions a_k. Denoting the approximation $\hat{P}(x, a_k)$, and given a particular input x, XCSF maximizes payoff by picking the action a_k^* that maximizes $\hat{P}(x, a_k)$. However, XCSF may be used to approximate functions other than payoff, a flexibility that is exploited here.

The three architectures are illustrated using a simple one-dimensional but non-linear testbed problem introduced in [15]. In this so-called "frog" problem, a system (frog) senses an object (fly) via a signal that monotonically decreases with the distance between them. The frog has available a continuous range of jump lengths and is supposed to learn to jump exactly the distance to the fly. Following the jump, which may undershoot or overshoot the fly, payoff is given by the resulting sensory signal.

The paper begins with a more detailed description of the frog problem, providing a concrete context for the three architectures. Then XCSF is briefly reviewed. Further sections describe and test the architectures themselves, followed by conclusions as to their merits and suggestions for further work.

2 Frog and Fly: A 1-D Continuous-action Problem

As noted in Section 1, the frog should learn to pounce on the fly in one jump. Let d be the frog's distance from the fly, with $0.0 \leq d \leq 1.0$. For simplicity, we assume x, the frog's sensory input, falls linearly with distance: $x(d) = 1 - d$. The frog "sees the fly better" the closer it is.

The payoff should be any function of x and action a that is bigger the smaller the distance d' that remains after jumping. That is, $P(x, a)$ should monotonically increase with smaller d'. A natural choice is to let the payoff equal the sensory input *following* the jump, as though the frog is rewarding itself based on what it "sees". Then, with the sensory function above, $P = 1 - d'$.

To write the payoff in terms of x and a, we need to make one assumption. Suppose the frog's jump overshoots, i.e., the frog lands *beyond* the target fly. In this case we assume that d' equals the amount of the overshoot (taken as a positive number). Thus $d' = d - a$ for $a \leq d$ and $d' = a - d$ for $a \geq d$. Substituting for d' in $P = 1 - d'$, then using $d = 1 - x$ and rearranging, we get

$$P(x, a) = \begin{cases} x + a & : \quad x + a \leq 1 \\ 2 - (x + a) & : \quad x + a \geq 1 \end{cases} \tag{1}$$

Figure 1 illustrates $P(x, a)$. For a given sensory input x, the payoff in general first rises, reaches a peak, then falls as a increases. The "ridge" of the function's tent shape in fact corresponds to the maximum payoff and thus optimal action over x's range. Note that the payoff function is highly non-linear—albeit composed of two linear planes.

To solve the frog problem, a system must learn to choose, given x, the value of a corresponding to maximum payoff. In the context of classifier systems (and reinforcement learning (RL) [9] in general) this is a non-trivial problem. The reason is that the only feedback allowed the system is in terms of payoff; the system gets no "error signal" with respect to its action choice. From the RL point of view, this restriction models the actual learning situation of many natural and artificial systems. Classifier systems (as well as other RL systems) deal with the restriction by, in effect, observing and estimating the payoffs associated with different action choices, then, given an input, choosing the action that appears

to pay the best. This process is quite well understood within the framework of discrete actions. We build on this understanding in the three architectures for continuous action.

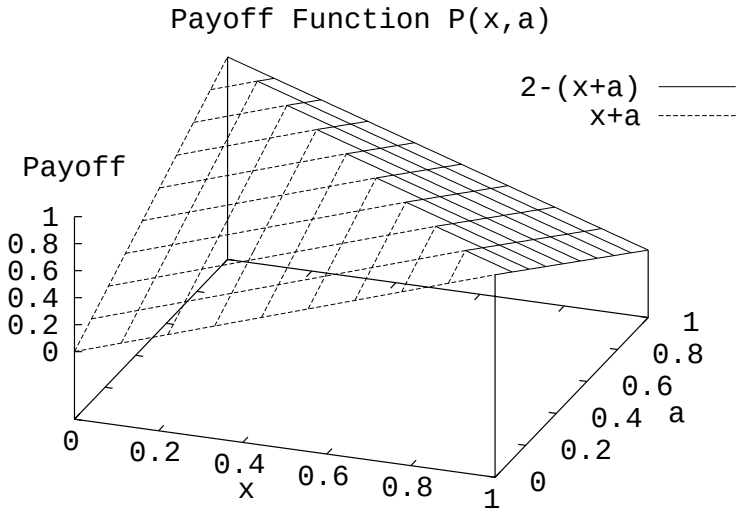

Fig. 1. Frog problem payoff function $P(x, a)$.

3 Brief Review of XCSF

XCSF is similar to XCS. This section assumes familiarity with XCS and discusses only differences between XCSF and the earlier system.

The essential innovation in XCSF with respect to XCS is that classifier predictions in XCSF are *calculated* instead of being fixed scalars. Each classifier has a *weight vector* w which is combined with the input vector x to form a prediction that is a linear function of the input. As a result, continuous, real-valued payoff landscapes can be approximated more efficiently [15] than would be the case with fixed scalar predictions, which allow only piecewise-constant approximations. Polynomials of order higher than linear may also be used in XCSF [7].

The classifier condition in XCSF is a truth function, denoted $t(x)$, which defines a subspace of the real-valued (or integer-valued) input space such that a classifier matches if an input is in that subspace. In the simplest case, $t(x)$ is a

concatenation of "interval predicates", $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are reals. A classifier matches an input x with components x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i. However, $t(x)$ can be any evolvable function that defines a subspace of the input space.

With these definitions, a classifier in XCSF can be conveniently notated

$$t(x) : w : a_k \Rightarrow p(x, a_k), \tag{2}$$

where a_k is the kth discrete action. The expression $p(x, a_k)$ is the classifier's prediction given x and a_k.

In operation, XCSF differs significantly from XCS only with respect to prediction calculation and updating. To calculate its prediction, a classifier with action a_k forms $p(x, a_k) = w \cdot x'$, where x' is x augmented by a constant x_0, i.e., $x' = (x_0, x_1, ..., x_n)$. Just as in XCS, the prediction is only produced when the classifier matches the input. As a result, $p(x, a_k)$ in effect computes a *hyperplane approximation* to the payoff function $P(x, a_k)$ over the subspace defined by $t(x)$. The value of x_0 is chosen to be of the same order as the component values of x.

As in XCS, XCSF forms a *system prediction* $\hat{P}(x, a_k)$ for each a_k as a fitness-weighted average of the predictions $p(x, a_k)$ of the classifiers that match the input. Then, if XCSF is in *explore* mode, one of the a_k is chosen at random and sent to the environment. If in *exploit* mode, the action with the highest system prediction, a_k^* is chosen. In explore mode, the payoff actually received, $P(x, a_k)$, is used to update the predictions of the classifiers that advocated the chosen action (i.e., the *action set* classifiers).

The predictions are updated using a *modified delta rule*

$$\Delta w_i = (\eta / |x'|^2)(t - o)x_i \tag{3}$$

in which t ("target") is the current payoff $P(x, a_k)$ and o ("output") is the current calculated value of $p(x, a_k)$. The factor η is a rate constant; the normalization $|x'|^2$ makes the change in output strictly proportional to $(t - o)$ and controllable by η. More powerful techniques of updating have been investigated [6].

Other differences from XCS adjust the genetic algorithm (GA) and the covering of unmatched inputs to accord with the structure of $t(x)$. Further details on this and the above material may be found in [15]. To be noted at this point is the fact that XCSF can be employed either as a discrete-action classifier system or, by reducing the a_k to a single, dummy, action, as a function approximator. In the latter case, the prediction becomes just $p(x)$, with $P(x)$ representing a function to be learned by approximation. XCSF is used both ways in the three architectures.

4 Interpolating Action Learner (IAL)

4.1 IAL Concept

In IAL, a second classifier system learns the action choices of a first classifier system. The overall system, shown schematically in Fig. 2, consists of two classifier systems S1 and S2 working in tandem.

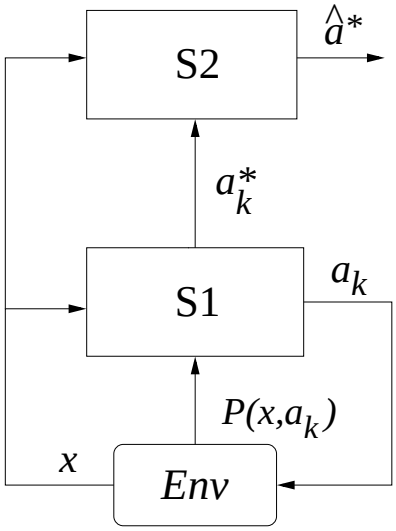

Fig. 2. Interpolating Action Learner. Both S1 and S2 are XCSF systems, but S2 observes S1's optimal actions and learns a continuous approximation to $a^*(x)$, the optimal policy for environment *Env*.

The lower system S1 is in contact with an environment *Env* that provides real state vectors x to S1, receives discrete actions a_k from S1, and provides real payoffs $P(x, a_k)$ to S1 in response. The system S1 is based on XCSF. For each possible x, S1 learns to approximate the payoff $P(x, a_k)$ to be expected for each a_k it might take.

While S1 is learning, the system S2 *observes* S1's *prediction array* (the array containing S1's system predictions). Thus, given x, S2 can see what payoff S1 expects for each a_k. In particular, S2 notes which a_k has the *maximum* expected payoff, and uses this action value, a_k^*, as an input. The other input to S2 is x. S2's objective is to learn to predict, given x, the correct value of a_k^*.

The combined system S1-S2 has two innovations. First, once S2 has learned to predict correctly, it represents a *direct* mapping from x to the optimal action, thus embodying the optimal policy $a^*(x)$. Ordinarily, RL systems do not directly map from x to $a^*(x)$. Instead, as with S1, they learn expected payoffs for each a_k, then, given an x, choose the a_k with the maximum expectation and send it to the environment. In contrast, S2 sees x and predicts $a^*(x)$ directly.

Second, the architecture permits the output of S2 to be *continuous-valued*, despite the fact that S2 is only capable of discrete actions. Like S1, S2 is a classifier system based on XCSF, but it is configured for just one, dummy, action. In this configuration a_k^* is treated formally like a *payoff to be learned*, with $\hat{a}^*$ the predicted value. Thus S2 in effect acts as a function approximator approxi-

mating $a^*(x)$. If S2's error threshold ϵ_0 is small (strict) then S2's approximation $\hat{a}^*$ should be similar to $a_k^*(x)$ which—for the frog problem employed here—is a staircase-like function of x. However, if ϵ_0 is not small, S2's approximation should be less discontinuous, becoming smoother with larger ϵ_0. Consequently, as x changes continuously, $\hat{a}^*$ should also change continuously so that S2 will approximate $a^*(x)$, the optimal *continuous* policy for *Env*.

4.2 IAL Experiments

The IAL system was tested on a version of the frog problem in which the frog had five discrete jump lengths: 0.0, 0.25, 0.50, 0.75, and 1.0. This is the same problem learned by XCSF in [15].

Parameter settings for S1 were the same as in [15]: population size $N = 500$, learning rate $\beta = 0.2$, error threshold $\epsilon_0 = 0.01$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 48$, crossover probability $\chi = 0.8$, mutation probability $\mu = 0.04$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$. Also, mutation increment $m_0 = 0.1$, covering interval $r_0 = 0.1$, $\eta = 0.2$ and $x_0 = 1.0$. GA subsumption was enabled, with $\theta_{GAsub} = 100$. The same parameter set was used in S2, except that the error threshold ϵ_0 was different in each of three experiments: 0.01, 0.05, and 0.10 [1]

In an experiment, each problem consisted of placing the fly at a random distance ($0.0 \leq d \leq 1.0$) from the frog and having S1 go through a standard explore cycle in which it matched the input, chose an action at random, got payoff according to the resulting distance from the fly, updated the action set classifiers, and possibly performed the GA. At the same time, S2 watched S1's prediction array, noted the optimal action (the one with the largest predicted payoff), and learned that action, a_k^*, by approximating it via the XCSF mechanism, as a function of x. That is, given x, S2's classifiers predicted S1's action (instead of a payoff) and were updated according to the actual value of a_k^*. (Those classifiers thus had the format, $t(x) : w \Rightarrow a(x)$, where $a(x)$ took the place of $p(x, a)$ and the classifiers themselves had no action variable per se.)

Each experiment consisted of 400,000 problems to be sure the system had stabilized. At the end of each experiment, the input range was *scanned* with a resolution of 0.01 and S2's predicted action $\hat{a}^*$ was calculated for each point and plotted. Figure 3 shows the results for each value of ϵ_0.

For the smallest ϵ_0, the plot of $\hat{a}^*$ is an almost perfect staircase corresponding to the best discrete a_k vs. x. With the middle value of ϵ_0, the plot shows a tendency to flatten, and with the largest value, 0.10, it is a straight line corresponding to the optimum *continuous* action (except that the line is displaced slightly upward!).

Examination of the final populations for the three cases showed 14 classifiers for $\epsilon_0 = 0.01$, 39 for 0.05, and exactly one classifier for $\epsilon_0 = 0.10$. That classifier's weight vector is directly indicated by the slope and intercept of the straight-line

[1] The values given are the actual error threshold divided by the maximum possible payoff, in this case 1000.

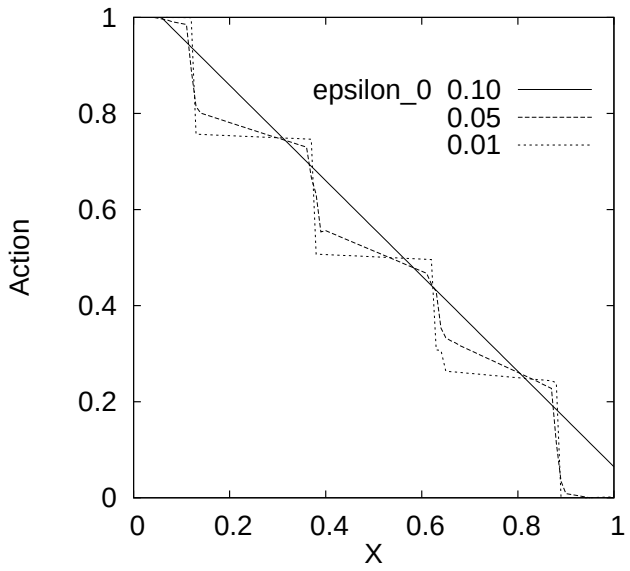

Fig. 3. Frog problem results for IAL. Predicted optimal action $\hat{a}^*$ (jump length) vs. input x (sensory input) as learned by S2 for three values of error threshold ϵ_0. Larger values permit closer approximation of optimal continuous action.

plot in Figure 3. The population for $\epsilon_0 = 0.10$ reached less than a dozen classifiers by 50,000 problems and fell to one classifier around 200,000 problems. Maximum accuracy in approximating $a^*(x)$ was reached much quicker, in a few thousand problems.

4.3 IAL Discussion

The experiments suggest that the IAL system can do what was intended, i.e., approximate $a^*(x)$ from the payoffs to a set of discrete actions. However, the function to be approximated—a straight line over a 1-D input domain—is certainly about the simplest imaginable. Because the IAL method achieves smoothness of approximation through a large error threshold, approximation accuracy in functions with higher bandwidth may be limited. A second drawback is that while the optimal continuous action is learned, nothing is learned about the value of non-optimal actions. That is, the rest of the continuous payoff landscape remains unknown to the system. Depending on the goals for the system, this may not matter, however.

The fall of the population size to a single classifier in the straight-line case (and to small numbers in the others) is encouraging because it confirms the ability of XCSF to evolve classifiers that approximate functions with high efficiency. This property of XCS-like systems has been investigated theoretically [1]. On the other hand, the time—tens of thousands and more of problem instances—to

reach this efficiency seems very long. That may be a consequence of working in the real domain—instead of the binary, as with XCS—and calls for further investigation.

Overall, IAL seems interesting but not outstanding as an architecture for achieving continuous action. Better would seem an architecture that more directly developed an approximation to $a^*(x)$, instead of indirectly as a secondary process. A step in this direction is taken in the next section.

5 Continuous Actor-Critic (CAC)

5.1 CAC Concept

In the actor-critic approach to reinforcement learning problems, one system, termed "actor", produces actions in response to an environmental input, x, and a second system, "critic", adjusts the actor with the intention of improving the actor's response. On what basis can the critic do this? The general scheme is shown in Figure 4, where S3 is the actor and S4 the critic.

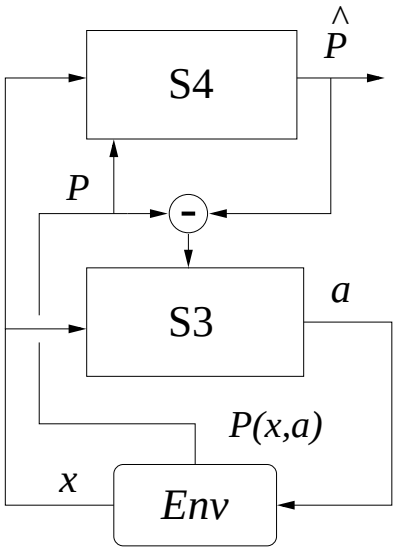

Fig. 4. Continuous Actor-Critic. S4 ("critic") is an XCSF system, learning $P(x)$, but S3 ("actor") is a special system with classifiers of the form $t(x) : w \Rightarrow a(x)$. S4's prediction error modifies probabilities of activation of S3's classifiers.

While S3 interacts with *Env*, S4, an XCSF system, learns to predict *Env*'s payoffs. Specifically, S4 learns an approximation to $P(x)$. On every learning step, there will in general be a difference between S4's prediction and the actual payoff

received. If, for instance, the actual payoff is less than the predicted one, it is likely that S3 could have chosen a better action than it did. This information, though possibly erroneous, is used to *reduce the probability* of that action being chosen in the future. Conversely, if the payoff is larger than predicted, the probability of the current action is increased.

If S3 were simply choosing actions randomly, these corrective signals from S4 would be meaningless. However, if the probability of S3's actions is affected as above by S4's prediction errors, there should be a convergence in which S4's predictions become more and more accurate and S3's actions produce payoffs that are increasingly likely to be maximal. This is the essence of the actor-critic idea [9]. As a method in RL, it has been somewhat displaced by the introduction of Q-learning [10]. However, actor-critic retains computational advantages where the action is computed as a continuous function of the input, as it is here (in the language of RL, "the policy is explicitly stored" [9]).

CAC is an actor-critic architecture implemented with classifier systems. As noted, S4 is XCSF used as a function approximator to predict $P(x)$. S3, however, is not an XCSF system. It contains classifiers with the format $t(x) : w \Rightarrow a(x)$ each of which has an associated probability of activation π_i (i indexes the classifier). Given x, classifiers form a match set as usual, but then one of the matching classifiers is chosen based on their relative probabilities (the π_i are first normalized). This classifier calculates an action using x and its weight vector w, then the action is sent to the environment. The environment returns payoff, but this is unusable directly by S3 since S3 emitted an action, not a payoff prediction. Instead, the payoff is sent to S4, where it is used to adjust the payoff prediction of S4. Finally, the difference between S4's prediction and the actual payoff is used to adjust the probability π of the activated classifier.

A straightforward method would adjust the probability as follows:

$$\pi \leftarrow \begin{cases} \pi + g(\Delta P/P)\pi & : \quad \Delta P \leq 0 \\ \pi + g(\Delta P/P)(1.0 - \pi) & : \quad \Delta P > 0 \end{cases} \tag{4}$$

Here g ($0 < g \leq 1$) is a gain factor and ΔP is the error in S3's prediction, i.e., $(\hat{P} - P)$. The adjustment is intended to have the effect that the best matching classifier's probability tends toward 1.0 and the probabilities of other classifiers tend toward 0.0.

S3's classifiers are generated by a genetic algorithm acting on the match sets, using the probabilities π_i as fitnesses. Thus classifiers whose actions result, on the average, in above-average payoffs should tend to be selected and reproduced. Furthermore, there should be a pressure to preferentially reproduce higher-probability, more-*general* (having large $t(x)$ domains) classifiers because such classifiers will occur in more match sets than more-specific classifiers [1]. As a result, S3's population should tend toward efficient coverage of the input space.

An important difference between S3 and XCSF-like systems is that the weight vector w must be *evolved* along with $t(x)$; it cannot be adjusted based on environmental feedback since the classifiers compute actions, and the feedback is in terms of payoff.

5.2 CAC Discussion

Unfortunately, significant experiments on CAC have not been done. As a preliminary test, an XCSF system was investigated in which the weight vector was evolved instead of adjusted. The adaptation was slow and irregular, suggesting that adjustment might be superior to evolution in this case. However, recent work by Hamzeh and Rahmani [4] evolved weight vectors successfully. Also, evolutionary techniques other than the GA may prove better for this kind of smooth maximization problem.

Assuming the CAC concept can work, it is an advance over the first technique, IAL, because it should result in classifiers in S3 that closely approximate $a^*(x)$, instead of achieving continous action through large approximation error. However, like IAL, CAC still fails to learn anything about the payoff landscape associated with non-optimal actions. The third architecture, presented in the next section, remedies this and though it too has problems, it may turn out to be the best-founded direction for continuous action.

6 Generalized Classifier System (GCS)

6.1 GCS Concept

As a discrete-action classifier system, XCSF learns $P(x, a_k)$. It uses classifiers of the form, $t(x) : w : a_k \Rightarrow p(x, a_k)$, in which the variable a_k is not only discrete, but treated separately from the variable x. However, the underlying payoff landscape is simply $P(x, a)$; not only is a continuous but it is functionally homogeneous with x. Why not learn $P(x, a)$ directly? Why not have classifiers of the form $t(x, a) : w \Rightarrow p(x, a)$, in which the condition depends on, and the weight vector refers to, both x and a? We term this sort of classifier system "generalized" in recognition of the homogeneous treatment of input and action, with both permitted to be real variables [2]

Learning, or exploration, in GCS is straightforward and similar to learning in XCSF. Given an input x, the system picks an action a according to its current exploration regime, e.g., it might try actions randomly. Then a match set [M] is formed of classifiers satisfying $t(x, a)$. The predictions $p(x, a)$ of each member of [M] are calculated, using the member's w, just as in XCSF. The action is performed in the environment and payoff P received. P is then used to adjust the error and fitness of each classifier in [M], and to update the w's of [M], again just as in XCSF. New classifiers are formed by the GA, and classifiers created by covering if needed, just as done in XCSF, except of course that now the condition and weight vector include a as well as x.

It is in the "exploit" side of GCS that substantially new steps are required. The object is to determine, given an input x, what action a will produce the

[2] As defined here, GCS is not, of course, fully general since operation in multi-step tasks is not discussed, and the system has no provision for internal state variables, as in [8]. Neither extension appears problematic.

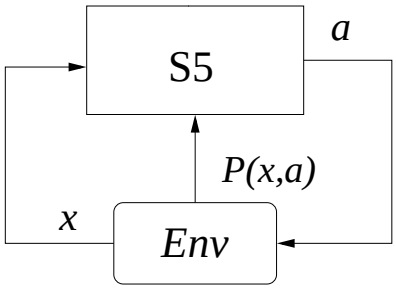

Fig. 5. Generalized Classifier System. S5 is a new system partly based on XCSF with classifiers of the form $t(x,a) : w \Rightarrow P(x,a)$.

greatest payoff. The simplest way conceptually would appear to be to scan all possible actions a, form an [M] for each, and choose a^* as the one for which the system prediction $\hat{P}(x,a)$ was highest. But there are difficulties. In the first place, scanning the range of a inevitably means discretizing the range, which in turn implies that the system's action is not truly continuous. Secondly, the higher the scan's resolution the greater the number of match sets [M] that must be formed, so that higher approximate continuity in a trades off against efficiency. For GCS, a new approach to picking a^* seems desirable.

Consider the exploit situation. An input x comes in. The action a has not been determined, so how does the system determine which classifiers belong in [M]? The answer is: a classifier belongs in [M] if there exists an a such that that classifier's $t(x,a)$ is satisfied. For example, if $t(x,a)$ employs interval predicates, x need only satisfy the predicates devoted to x's components (a can be any value satisfying its predicate). For other forms of $t(x,a)$ the answer to the question and the computational expense of answering it depend on the form. An important form of $t(x,a)$ for which matching is quick will be mentioned shortly.

Once the match set [M] is built, the system moves on to determine the highest-paying action a^*. Consider first a classifier in [M] and suppose for concreteness that interval predicates are employed in $t(x,a)$. Let (a_l, a_u) be the action predicate. Since, $p(x,a)$ is computed *linearly*, i.e., using a weight vector w, it is clear that *either* a_l *or* a_u will yield a higher prediction than any other value of a that satisfies the interval. Thus, no scanning is necessary to deter-

mine this classifier's best action, a_{best}; it is only necessary to compare $p(x, a_l)$ with $p(x, a_u)$. Finally, for a^*, the system picks the highest-paying a_{best} of the individual classifiers of [M].

At this point it is necessary to note a disabling drawback of basing $t(x, a)$ on interval predicates. Consider a classifier in [M]. For this classifier, a_{best} is a constant; it equals either a_l or a_u, independent of x. Similarly, the values of a_{best} for the other members of [M] are also constant. The implication is that if x changes slightly, a^* will either not change, or it will change abruptly as a different a_{best}, from another classifier in [M], becomes a^*. Thus the use of interval predicates has the consequence that a^* is discontinuous with respect to x.

This effect will be especially evident if the payoff landscape, $P(x, a)$ is *oblique*, as in Figure 1. The "ridge" corresponding to the maximum of the function is not parallel with either axis. Consequently, approximating the function with conditions based on (axis-parallel) interval predicates would require that classifiers overlap in staircase fashion at the ridge, so that a^* could approximate continuity only by employing large numbers of such classifiers. On the other hand, if the classifier conditions could have non-axis-parallel contours then much or all of the staircasing discontinuity could be avoided.

Consider, as a sort of thought-experiment, the possibility of evolving conditions with arbitrary contours. Then imagine a classifier whose $t(x, a)$ exactly covered the portion of the x-a plane lying below the line $x + a = 1$. This classifier would match any x and, significantly, its $a_{best} = 1 - x$, which is both perfectly continuous and exactly the correct solution to the frog problem. To represent the complete landscape, just one more classifier would be needed; its condition would cover the portion of the plane lying above the $x + a = 1$ line. Thus it appears that GCS can achieve both continuous action and exact solutions—as well as economy of representation—if classifier conditions can be evolved that have arbitrary contours. While this thought-experiment is illuminating, evolution of arbitrary $t(x, a)$ is difficult (see [5] for an initial attempt). The essential requirement, however, is for conditions that can evolve so that their contours align with oblique features of the payoff landscape. The rest of this section outlines an example approach using orientable elliptical conditions that appears to offer considerable continuity, precision, and economy.

Using XCSF, Butz [2] investigated classifiers having general hyperellipsoidal conditions and showed they could approximate highly oblique functions more efficiently (smaller populations, lower system error) than classifiers having hyperrectangular interval conditions. The general hyperellipsoids had the property that they could rotate with respect to the coordinate axes, permitting them to orient to the contours of the function [3]. Butz's work applied to function approximation, but the idea appears promising for continuous action as well. The following describes GCS with orientable elliptical conditions and its application to the frog problem.

[3] For hyperellipsoids, rotation (which is not possible to visualize) is considered more generally as the introduction of cross-product terms, e.g., $x_4 x_7$, in the hyperellipsoid definition, in analogy to the definition of a rotated ellipse.

In the frog problem, $P(x, a)$ is 2-dimensional so that conditions can be represented with ordinary ellipses. We define such a condition using the parameters $< m_x, m_a, l_x, l_a, \theta >$. In general, the ellipse is not centered at the origin, and it is rotated with respect to the x-axis. To obtain the actual ellipse using the parameters, we imagine it "starts" at the origin and that its semi-axis in the x direction is aligned with that axis and has length l_x; similarly, the a semi-axis has length l_a. Now, rotate the ellipse (counterclockwise) with respect to the x-axis through angle θ and displace its center to the point (m_x, m_a) to obtain the actual ellipse used in the condition. The equation for that ellipse is

$$\left(\frac{(x - m_x)\cos\theta + (a - m_a)\sin\theta}{l_x}\right)^2 + \left(\frac{-(x - m_x)\sin\theta + (a - m_a)\cos\theta}{l_a}\right)^2 = 1 \tag{5}$$

This may be expressed compactly as

$$(\mathbf{EV})^2 = 1, \tag{6}$$

where $\mathbf{E}$ equals the ellipse matrix

$$\begin{bmatrix} \cos\theta/l_x & \sin\theta/l_x \\ -\sin\theta/l_a & \cos\theta/l_a \end{bmatrix}$$

and $\mathbf{V}$ is the column vector

$$\begin{bmatrix} (x - m_x) \\ (a - m_a) \end{bmatrix}$$

The condition $t(x, a)$ is represented by the ellipse of Eqn. 6; it is satisfied if

$$(\mathbf{EV})^2 \leq 1, \tag{7}$$

i.e., if the point (x, a) is *inside* the ellipse.

In explore mode, as noted earlier, GCS combines the input x with a randomly chosen a and forms the match set [M] with classifiers whose conditions $t(x, a)$ satisfy (7). In exploit mode, [M] consists of classifiers for which, given x, there exists an a such that (7) holds. This is accomplished by setting $\mathbf{V} = \begin{bmatrix} (x - m_x) \\ 0 \end{bmatrix}$, i.e., by choosing a equal to the center a-value of the condition (if the condition does not hold for this value, it will certainly not hold for any other value).

The next step, in exploit, is to determine the a_{best} of each classifier in [M]. Consider the condition of a matching classifier. The input x defines a straight line crossing the ellipse parallel to the a axis. The line intersects the ellipse twice, at two values of a. By the reasoning used earlier in this section in connection with interval predicates, one of these values must be a_{best}. The two values are found by solving Eqn. 5, a quadratic equation with two roots, for a. The root with the larger predicted payoff, as calculated using w, is a_{best}. Finally, the largest a_{best} of the classifiers in [M] becomes the system's chosen action, a^*.

It is important to observe that, for a given matching classifier the value of a_{best} will in general change—continuously—as x changes. The reason is that

since $t(x, a)$ is elliptical, its contour varies continuously with x. Furthermore, since the ellipses can rotate and elongate, they may evolve to align with and along a considerable length of the oblique ridge in $P(x, a)$. Thus the transition between different a^*s as x changes may be substantially continuous and not staircased. Moreover, the ability of the conditions to rotate holds out the possibility, in the frog problem, of evolving very small populations, since in principle just two extremely elongated ridge-parallel classifiers could cover the whole payoff landscape.

This completes the description of the GCS concept. In summary, the landscape $P(x, a)$ is approximated using an XCSF-like method in which a is an "input" along with x. Optimal actions are chosen as the best of the best of recommendations of individual classifiers. The actions are largely continuous with respect to x because the condition contours are continuous with respect to x and the conditions themselves can orient to align with landscape features. In the next section we examine the results of an experiment with GCS.

6.2 GCS Experiments

Experimental investigations of GCS are work in progress. Reported here is one of the first experiments that showed some aspects of what the concept suggested should happen. In fact, it took a number of experiments to arrive at an implementation of the concept that seemed correct. This was due largely to the unaccustomed use of the action variable as a kind of input.

Parameters for the experiment were as follows: population size $N = 2000$, learning rate $\beta = 0.5$, error threshold $\epsilon_0 = 0.01$, fitness power $\nu = 5$, GA threshold $\theta_{GA} = 48$, crossover probability $\chi = 0.8$, mutation probability $\mu = 0.04$, deletion threshold $\theta_{del} = 50$, fitness fraction for accelerated deletion $\delta = 0.1$, $\eta = 0.2$ and $x_0 = 1.0$. Neither GA nor action-set subsumption was enabled.

Mutation was handled differently according to the ellipse parameter being mutated. For ellipse center coordinates m_x and m_a, the value was changed by a random quantity from [-0.1,0.1]. For ellipse axis lengths l_x and l_a, the value v was changed (following [2]) by a random quantity from $[-0.5v, 0.5v]$. For ellipse angle θ, the value was changed by a random quantity from [-0.5,0.5] (radians). In covering: m_x was given by the x input; m_a was the selected action if it was an explore problem, otherwise a random value from [0.0,1.0]; the axis lengths were random from [0.0,0.1]; and θ was random from [0.0,1.0].

In each problem of an experiment, the fly was placed at a random distance ($0.0 \leq d \leq 1.0$) from the frog. A conventional design was used in which with probability 0.5 the problem was an explore problem, otherwise an exploit problem. Figure 6 shows the results of one run that ended at 100,000 explore problems. Each curve plots a moving average over the past 50 exploit problems.

The Payoff curve shows that by a few thousand problems, the system was regularly receiving payoff greater than 0.95. Thus, since $P = 1 - d'$ in this experiment, the frog was regularly jumping to a position less an 0.05 from the fly. The System Error curve approximately complements the Payoff curve, though not precisely since system error is the difference between actual and predicted

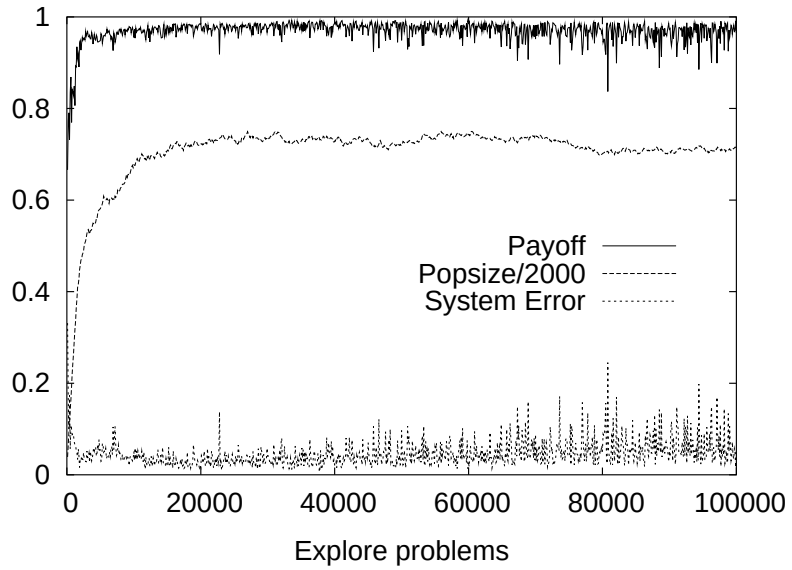

Fig. 6. Results for GCS in a frog problem experiment (curve order same as in legend).

payoffs. In both curves there appears to be a slow degradation toward greater volatility. The population size curve rises to about 70% of N and then declines very gradually.

Inspection of the population at 100,000 problems (as well as much earlier) showed that the w_1 and w_2 components of the weight vectors of most classifiers had converged either to (1.0,1.0) or to (-1.0,-1.0), depending on whether the classifier's condition applied below the $a = 1 - x$ diagonal or above it. Thus most classifiers were indeed approximating the payoff function (Eqn. 1). In fact, most classifiers had an error less than .0005, even though the system error was much higher.

The conditions of the population's ten highest-numerosity classifiers were plotted to get an idea of their size and orientation (Fig. 7). The figure indicates that the conditions have evolved so that their major axes are approximately aligned with the $a = 1 - x$ landscape ridge. The ellipse sizes appear reasonable for collectively approximating the regions above and below the ridge.

As a direct indication of the system's ability to choose the best action a^* and to gauge a^*'s continuity with respect to x, the input was scanned with increment 0.001 and the resulting a^* plotted (Fig. 8). The plot lies close to the diagonal and shows intervals of continuous change broken by abrupt small discontinuities suggesting the system is switching from one highest-predicting classifier to another.

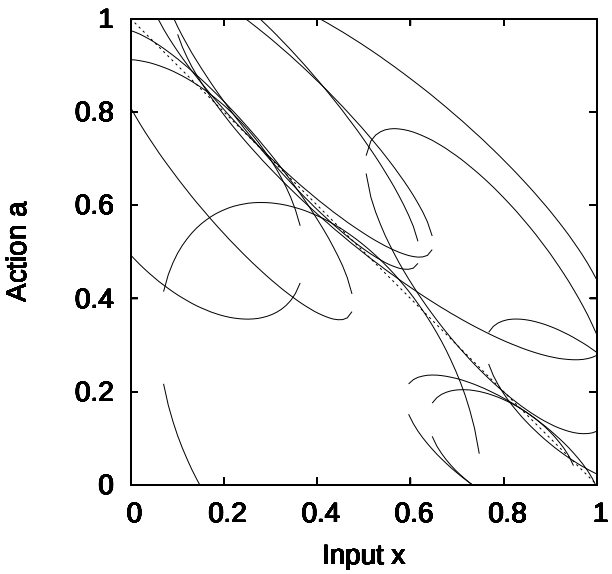

Fig. 7. Elliptical conditions of the 10 highest-numerosity classifiers in GCS frog problem experiment. Diagonal ($a = 1 - x$) corresponds to "ridge" in payoff landscape. Breaks in ellipses are a plotting artifact.

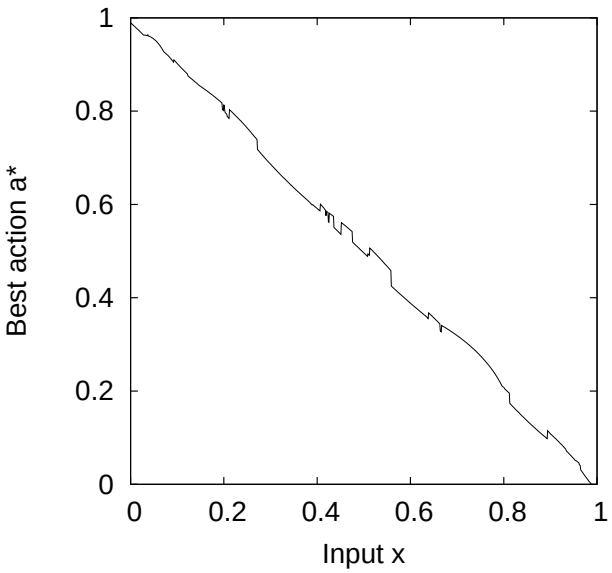

Fig. 8. Scan of best action a^* vs. input x for GCS frog problem experiment (x increment 0.001).

6.3 GCS Discussion

The limited experimental results with GCS are mixed but quite promising. The condition ellipses indeed seem to be evolving to align with the diagonal (oblique) feature of the frog problem environment. Their sizes are reasonable: not so small as to require large numbers to approximate the diagonal nor so large—or fat—as to produce a poor approximation. At the same time, we did not observe the evolution of just *two* dominant ellipses, one covering each landscape region, each very long so that its side was practically a straight line. This would have been the ideal result, and it is not clear why the system's generalization pressure [1] did not cause larger- and particularly longer-condition accurate classifiers to win out until just two dominant ones remained.

In the binary domain, generalization involves substituting #'s (don't-cares) for specified alleles in the condition. Each additional # in fact doubles the subspace that the condition matches. If corresponding inputs occur, the classifier becomes twice as active and thus has twice the reproductive opportunity compared with classifiers that are just one # less general. In the real-valued domain, in contrast, mutation of a single allele normally results in a condition that is at most only slightly more general than the next-more-specific classifiers, so that the evolutionary pressure over them is considerably less than in the binary case. It may be that this weakness of generalization pressure explains the fact that the population did not appreciably "condense" (shrink) in our experiment, and may explain other similar real-domain classifier system results. One of the most attractive features of accuracy-based classifier systems is their ability, under the right conditions, to generalize, i.e., condense down to a set of classifiers that quite directly expresses the regularities of the payoff environment. It seems important for future research to understand how this process can be as effective in the real domain as it is in the binary—clearly the present frog problem offers the possibility of very substantial generalization: to just two classifiers!

Apart from the important issue of generalization, the GCS concept appears, from the experiments so far, to be relatively sound. It is not understood why the performance became more volatile with time but more intimate tracking of populations should yield clues. Often deterioration of this sort is due to increasing overgeneralization—which in this instance would mean classifiers whose conditions fall across the landscape diagonal. Yet, as seen, the highest-numerosity classifiers at 100,000 problems were fine in this respect.

While volatile, the performance did rise quickly to near-optimal. Why it did not go all the way to optimal (payoff 1.0) appears to depend at least in part on the specific classifier that ends up providing its a_{best} for a^*. The edge of this classifier's condition lies close to the landscape diagonal: it may fall slightly short of the diagonal, or go over a bit. In some cases the classifier may be newly generated and inexperienced and thus not accurate; choosing its a_{best} could mean a large error. In earlier experiments it was found that system error improved if classifiers with prediction errors greater than a threshold were excluded from the a^* competition; this criterion was in force in the experiment reported.

Unlike the frog problem, most practical problems involve more than one input dimension. With a two-dimensional input, GCS would need ellipsoidal conditions, which require three (Euler) angles to specify their orientation. Above that, direct visualization is lost and (if keeping an ellipsoid-like basis) one would need to move to general hyperellipsoids of the kind already investigated by Butz [2]. It is interesting, however, that even for higher input dimensionality, the calculation of a_{best} still only requires solution of a quadratic equation.

7 General conclusion

Three architectures for continuous action in classifier systems have been introduced and examined. None appears unreasonable, and the third, GCS, appears to be perhaps the most interesting direction since it treats the action homogeneously with the input—reflecting the nature of the payoff landscape—and it is the only one that learns the landscape and not just its highest-paying region. Yet all three architectures as presented are quite crude and unrefined and need much more work (including, probably, correction of errors!). Perhaps the most useful perspective on the offering here is as a smorgasbord from which with careful selection and combination new insights toward continuous action might be gleaned.

References

1. Martin Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8:28–46, 2004.
2. Martin V. Butz. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. 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 1835–1842, Washington DC, USA, 25-29 June 2005. ACM Press.
3. Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, pages 253–272. Springer-Verlag, Berlin, 2001.
4. Ali Hamzeh and Adel Rahmani. An evolutionary function approximation approach to compute prediction in XCSF. In *Proceedings of the European Conference on Machine Learning*, pages 584–592, 2005.
5. Lanzi P. L. and S. W. Wilson. Classifier conditions based on convex hulls. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2005.
6. P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Generalization in the XCS classifier system: analysis, improvement, and extension. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2005.

7. Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Extending XCSF beyond linear approximation. 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 1827–1834, Washington DC, USA, 25-29 June 2005. ACM Press.
8. Pier Luca Lanzi and Stewart W. Wilson. Toward optimal classifier system performance in non-Markov environments. *Evolutionary Computation*, 8(4):393–418, 2000.
9. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press/Bradford Books, Cambridge, MA, 1998.
10. C. J. C. H. Watkins. *Learning From Delayed Rewards*. PhD thesis, Cambridge University, 1989.
11. Stewart W. Wilson. Continuous action. Slides presented at the Eighth International Workshop on Learning Classifier Systems (IWLCS-2005). Available at prediction-dynamics.com.
12. Stewart W. Wilson. Optimal continuous policies: a classifier system approach. Extended Abstract, International Workshop on Learning Classifier Systems (IWLCS-2004). Available at prediction-dynamics.com.
13. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
14. Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
15. 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.