Classifier Systems for
Continuous Payoff Environments

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. Recognizing that many payoff functions are continuous and depend on the input state x, the classifier system architecture XCS is extended so that a classifier's prediction is a linear function of x. On a continuous nonlinear problem, the extended system, XCS-LP, exhibits high performance and low error, as well as dramatically smaller evolved populations compared with XCS. Linear predictions are seen as a new direction in the quest for powerful generalization in classifier systems.

1 Introduction

This paper extends learning classifier system (LCS) architecture (using XCS [8, 3] as a basis) to environments in which the payoff is a continuous function of the input, or state, x. In most previous LCS work, the environmental payoff function $P(x, a)$ (with a the system's action) has been discontinuous. Continuous payoff functions bring new challenges but also new opportunities for classifier generalization and LCS application and permit seeing classifier systems from a broadened perspective.

The next section gives an explanation of continuous payoff environments and distinguishes them from discontinuous ones. Section 3 follows with a concrete example that will be used in experiments. Modifications of XCS for continuous payoff are given in Section 4 (see [12] for a review of XCS and [3] for an algorithmic description). Section 5 describes the experiments, followed by Section 6 with discussion.

2 Environments with Continuous Payoff Functions

2.1 Discontinuous environments

Payoff functions for environments typically used in LCS experiments are nearly always discontinuous. Consider learning a Boolean function in which a correct action (i.e., the correct value of the function) is rewarded with 1000 and an incorrect action with 0. Very often in such functions changing a single bit of x changes the function value from, say, 1 to 0. Suppose the action a does not change. Then

the reward received will change from 1000 to 0 (or vice-versa), implying that the payoff function $P(x, a)$ is locally discontinuous with respect to x. In fact, payoff functions for Boolean environments are full of such discontinuities. (They are also widely discontinuous with respect to a, but attention in this paper will focus on x.)

The discontinuity is not restricted to environments where the input is binary. Consider a data-inference problem in which exemplar attributes are real-valued and, as is typical, the correct discrete-valued decision (e.g., "malignant", "benign", "indeterminate") may change if the value of a single attribute crosses a threshold. If $P(x, a)$ assigns discrete reward values to the decisions, then $P(x, a)$ will again be discontinuous with respect to x.

These kinds of discontinuous payoff functions are well handled by the classifier syntax in a conventional LCS such as XCS. A classifier of XCS consists of a condition $t(x)$ (a truth function of x), an action a, and a scalar prediction p of the payoff to be expected if the system takes action a when the condition is satisfied by the current x. Collectively, the system's classifiers can evolve successfully to represent the environmental payoff function because discontinuities in $P(x, a)$ are simply handled by two classifiers, one for each side of the payoff "step". Conversely, where $P(x, a)$ is *not* discontinuous—i.e., the payoff is *the same* for several states (and a given action a)—XCS may evolve single classifiers in which $t(x)$ generalizes over those states (provided the syntax of $t(x)$ can express the generalization).

2.2   Continuous environments

Robotic, control, and other "real world" environments such as financial time-series prediction are often characterized by payoff functions that are continuous with respect to the input x and sometimes also with respect to the action a. Simplifying somewhat, a function $P(x, a)$ is *continuous* at $(x, a) = (x_0, a_0)$ if $\lim_{x \to x_0, a \to a_0} P(x, a)$ exists and $\lim_{x \to x_0, a \to a_0} P(x, a) = P(x_0, a_0)$ [1]. If $P(x, a)$ is continuous at all (x, a) of interest, then we will call it a continuous function. The intuition is that in a continuous function, small changes in the input result in small changes in the value of the function.

Sometimes $P(x, a)$ will only be continuous with respect to x, for example if a system is capable of just a finite set of discrete actions. Or, in principle, the function might only be continuous with respect to a. In the following, we shall consider the case where $P(x, a)$ has the form $P(x, a_j)$, where a_j is one of a finite set of discrete actions, and $P(x, a_j)$ is continuous with respect to x at every x of interest.

As a general framework, consider a robot system for which the environmental state is represented by a vector x of real-valued sensor readings and the robot can take any of a finite set of discrete motor actions such as "turn left", "take one step forward", etc. Given an input state x and an action a_j, the resulting state y may depend in a continuous way on x. That is, a slightly different x would, given a_j, result in a slightly different y. If in turn the payoff p to the system depends continuously on y, then in effect the robot would be acting in an environment

where $P(x, a_j)$ is continuous with respect to x. A classifier system designed to optimize the robot's movements in this environment would need to learn to predict $P(x, a_j)$. As will be seen in the following example environment and experiments with it, the traditional LCS architecture with its scalar predictions is inefficient and lacks transparency in learning continuous payoff functions, but these problems are greatly alleviated with a modified architecture. (For neural LCS approach to continuous payoff, see [2].)

3 Example: A 'Frog' Problem

Consider a frog-like system that learns the best-sized jump to catch a fly. The "frog" receives sensory input that is related to the fly's distance, jumps a certain distance in the fly's direction, and gets payoff that is based on the remaining distance. We shall assume that the frog has a finite set of discrete actions—in this case, jumps of certain lengths. For any fly distance (in the range allowed) the frog should learn to choose the action that lands it closest to the fly.

Let d be the frog's distance from the fly, with $0.0 \leq d \leq 1.0$. What should we take for x, the sensory input? Any quantity that monotonically decreases with d would be reasonable. For the moment we will take the simplest, a linear decrease: $x(d) = 1 - d$. For actions a_j, we will assume $k \geq 2$ equally-spaced jump lengths: $0.0, 1/(k-1), 2/(k-1), ..., (k-2)/(k-1), 1.0$.

The payoff should be any function of x and a that is bigger the smaller the distance d' that remains after jumping. That is, $P(x, a)$ should monotonically increase with smaller d'. One quite 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 shall 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

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

Payoff functions for each of the a_j discrete actions may be found by substituting a_j for a in (1). For $k = 5$ actions, the resulting functions—all continuous—are shown in Figure 1. The functions for actions 0.0 (no jump) and 1.0 are simple straight lines, but the functions for the three intermediate actions are nonlinear, forming "tents" that peak at $a_j = 1 - x$ (i.e. where the action equals the distance to the fly). For any x, the optimal action is the one corresponding to the function that is largest at that x. A classifier system would solve the frog problem by learning the functions and then, given an x, pick the action whose function is greatest there.

Because they make scalar predictions, traditional classifier systems are inefficient learners of continuous payoff functions. The reason is that the scalar

predictions amount to a piecewise-constant approximation. This is generally the least efficient approximation technique for continuous functions, in the sense that large numbers of classifiers are required in order to meet a given error criterion. [10, 12] demonstrated a classifier-system-like technique that learned continuous functions using piecewise-linear approximations, thereby exploiting the latter's far greater efficiency. In this paper we employ that technique in a standard multiple-action LCS architecture, where it is used to learn the $P(x, a_j)$.
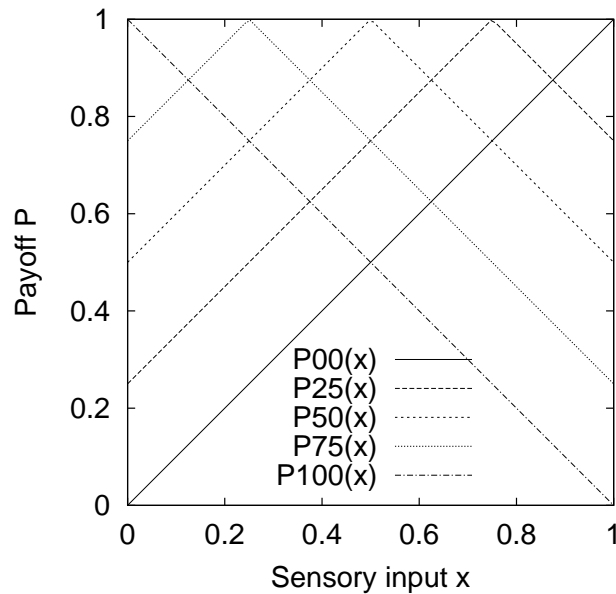

Fig. 1. Frog problem payoff functions $P(x, a_j)$ for $a_j \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$. (Functions denoted by "P00" for $P(x, 0.00)$, etc.)

4 XCS-LP: XCS Modified for Linear Predictions

To address the frog problem and, generally, to make an LCS capable of predicting efficiently in continuous payoff environments, XCS was modified in two respects. The first was to adapt the program for real instead of binary input vectors. The second was to change XCS's classifier architecture so that the fixed scalar prediction was replaced by a linear polynomial that would *calculate* the prediction from x. The resulting program was called XCS-LP ("linear prediction"). XCS-LP differs from XCS *only* as described in the next two subsections.

4.1 Real inputs

The changes to XCS for real inputs were as follows [9, 11]. The classifier condition was changed from a string from $\{0,1,\#\}$ to 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 attributes x_i if and only if $l_i \leq x_i \leq u_i$ for all x_i. In the experiments reported here, the input value range was [0.0,1.0].

Crossover (two-point) in XCS-LP operates in direct analogy to crossover in XCS. A crossover point can occur between any two alleles, i.e., within an interval predicate or between predicates, and also at the ends of the condition (the action is not involved in crossover). Mutation, however, is different. An allele is mutated by adding an amount $\pm rand(m_0)$, where m_0 is a fixed real, $rand$ picks a real number uniform randomly from $(0.0, m_0]$, and the sign is chosen uniform randomly. If a new value of l_i is less than the minimum possible input value, in the present case 0.0, the new value is set to 0.0. If the new value is greater than u_i, it is set equal to u_i. A corresponding rule holds for mutations of u_i.

The condition of a "covering" classifier (a classifier formed when no existing classifier matches an input) has components $l_0, u_0, ..., l_n, u_n$, where each $l_i = x_i - rand_1(r_0)$, but limited by the minimum possible input value, and each $u_i = x_i + rand_1(r_0)$, limited by the maximum possible input value; $rand_1$ picks a random real number from $[0.0, r_0]$, with r_0 a fixed real.

For the subsumption deletion operations, we defined subsumption of one classifier by another to occur if every interval predicate in the first classifier's condition subsumes the corresponding predicate in the second classifier's condition. An interval predicate subsumes another one if its l_i is less than or equal to that of the other and its u_i is greater than or equal to that of the other. For purposes of action-set subsumption, a classifier is more general than another if its *generality* is greater. Generality is defined as the sum of the widths $u_i - l_i + 1$ of the interval predicates, all divided by the maximum possible value of this sum.

4.2 Linear predictions

As mentioned, a classifier in XCS-LP calculates its prediction using a linear polynomial in x (i.e., in the components of x). [10] introduced a system, XCSF, in which the classifiers similarly used linear polynomials in x to learn, collectively, a piecewise-linear approximation to a given continuous function $f(x)$. However, XCSF had no actions (or just the "null" action) because its purpose was to use its predictions to approximate just one function. In XCS-LP, exactly the same piecewise-linear prediction technique is used, except that now there are k functions to be approximated, the $P(x, a_j)$ for each of the k actions. The following, reproduced substantially from [12], explains the prediction mechanism.

Besides its condition and action a_j, each classifier in XCS-LP has an associated *weight vector* $w = (w_0, w_1, ..., w_n)$, where n equals the number of components in x. To calculate its prediction, the classifier forms $p(x) = w \cdot x'$, 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, $p(x)$ in effect computes a *hyperplane approximation* to the payoff function $P(x, a_j)$ over the subspace defined by the classifier's condition. Classifiers will have different weight vectors w since in general the subspaces of their conditions differ.

Of course, the classifiers' weight vectors must be adapted. If classifiers are to predict with a given accuracy, the coefficients w_i of their weight vectors must be appropriate. Following [10] we used a modification of the *delta rule* [5]. The delta rule is given by

$$\Delta w_i = \eta(t - o)x_i \qquad (2)$$

where w_i and x_i are the ith components of w and x', respectively. In the quantity $(t - o)$, o is the output, in the present case the classifier prediction, and t is the *target*, in this case the current payoff $P(x, a_j)$. Thus $(t-o)$ is the amount by which the prediction should be corrected (the negative of the classifier's instantaneous error). Finally, η is the *correction rate*. The delta rule says to change the weight proportionally to the product of the input value and the correction.

Notice that correcting the w_i in effect changes the output by

$$\Delta o = \Delta w \cdot x' = \eta(t - o)|x'|^2. \qquad (3)$$

Because $|x'|^2$ is factored in, it is difficult to choose η so as to get a well-controlled overall rate of correction: η too large results in the weights fluctuating and not converging; if η is too small the convergence is unnecessarily slow. We noticed that in its original use [7], the correction rate was selected so that the entire error was corrected in one step; this was possible, however, because the input vector was binary, so its absolute value was a constant. In our problem, reliable one-step correction would be possible if a *modified delta rule* were employed:

$$\Delta w_i = (\eta/|x'|^2)(t - o)x_i. \qquad (4)$$

Now the total correction would be strictly proportional to $(t - o)$ and could be reliably controlled by η. For instance, $\eta = 1.0$ would give the one-step correction of [7]. In the experiments that follow, we used the modified delta rule with $\eta = 0.2$.

Use of a delta rule requires selection of an appropriate value for x_0, the constant that augments the input vector. We found that if x_0 was too small, weight vectors would not learn the right slope, and would tend to point toward the origin—i.e. $w_0 \approx 0$. Note that x_i is a factor in the above equation for Δw_i. If x_0 is small compared with the other x_i, then adjustments of w_0 will tend to be swamped by adjustments of the other w_i, keeping w_0 small. Choosing x_0 to be about the same order of magnitude as the other x_i solved the problem in [10] and is adopted here.

To change XCS from the traditional fixed predictions to linear predictions only required appending the weight vectors to the classifiers, plus providing for calculation of the predictions and application of the modified delta rule to the weight vectors of the action set classifiers (instead of employing and updating

scalar predictions as in XCS). In a classifier created by covering, the weight vector was randomly initialized with weights from $[-1.0, 1.0]$; GA offspring classifiers inherited the parents' weight vectors. In [10] both policies yielded performance improvements over other initializations and are followed here.

5 Experiments

In this section we report three experiments on the frog problem. The first applies XCS-LP as described above. The second applies (standard) XCS. The third changes the sensory and payoff functions of the problem and again applies XCS-LP. The idea was first to test XCS-LP, then compare with the XCS solution, then check XCS-LP's performance when the functions were less simple than those so far defined.

In each experiment, the system alternated between learning problems and test problems. In a learning problem, d was chosen randomly from $[0.0, 1.0]$, the corresponding x was input to the system, the system chose one of the $k = 5$ actions a_j at random, and the corresponding payoff was received from the environment and used for updating parameters (the GA was also enabled). In a test problem, again beginning with a random d, the system chose the action whose system prediction was highest and the corresponding payoff was received (with no updating or GA). The probability that a problem would be a learning problem was 0.5. Each experiment consisted of 5 runs each with a different random seed. Each run began with an empty classifier population and was carried out to 300,000 learning problems to be sure results had stabilized. The results were recorded as moving averages over the previous 50 test problems of: payoff, system error, population size (in macroclassifiers), and population generality. The curves given in the figures are averages over the 5 runs.

Common parameter settings for the experiments were: 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 $m_0 = 0.1$ and covering interval $r_0 = 0.1$. GA subsumption was enabled, with $\theta_{GAsub} = 100$. For XCS-LP, $\eta = 0.2$ and $x_0 = 1.0$.

5.1 XCS-LP on the frog problem

Results for XCS-LP are shown in Figure 2. From (1) and Figure 1 we can calculate that if the system chooses the best action every time it should receive an average payoff of 0.9375. The payoff curve appears to reach such a value, as was confirmed directly by the data. Thus on the time scale of the experiment, the system almost immediately learned to pick the best action. The system error also quite quickly went to a low value, which from the data was approximately 0.004 by the end of the experiment, actually less than the experiment's error threshold, $\epsilon_0 = 0.01$. During the first 50,000 problems the population size and

generality moved to values that remained fairly steady subsequently. By the end of the experiment, the average population size was 24.6 classifiers.

Figure 3 gives insight into the evolved population. Shown are the classifiers of a typical population, at 300,000 learning problems. A special graphic notation is used to represent the conditions. The range of x is divided up into subranges of length 0.1. A "." appearing in a subrange means that the condition's interval predicate will accept no x value in that subrange. "0" means that every x will be accepted, while "o" means some will be accepted because one or both boundaries of the predicate fall somewhere in that subrange. The condition of classifier 0 accepts every x—it is completely general. Classifier 1 is almost so—in fact the value of u is 0.999. The actual ranges for classifiers 2 and 3 are (0.474, 1.0) and (0.0,0.527), and so forth.

The first 13 classifiers have high numerosity and fitness compared with the remainder. Because they dominate the calculation of the system prediction, they can be regarded as the system's solution to the task. Inspection shows that the conditions of the classifiers quite accurately correspond to the x ranges of the straight-line segments of $P(x, a_j)$ for each action. Furthermore, the weight vectors have slopes and intercepts that also reflect the $P(x, a_j)$. For instance, classifiers 2 and 3 reflect $P(x, 0.5)$ in that they approximately equally divide the x range and their weight-vector intercepts (first components) and slopes are close to those of the ideal weight vectors which are (1.5, -1.0) and (0.5, 1.0). Similarly for the classifiers corresponding to actions 0.0, 0.75, and 1.0. However the upper straight-line segment for $P(x, 0.25)$ seems to be weakly defined (classifiers 4, 7, and 10) in that the slope is around -0.68 whereas it should ideally be -1.0.
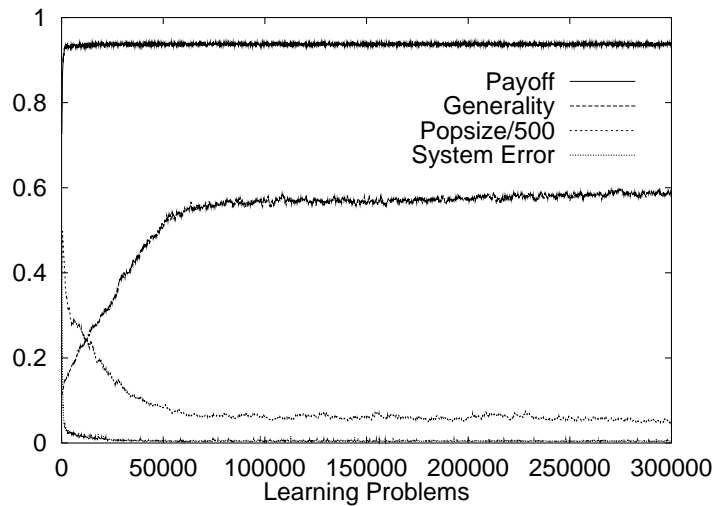

Fig. 2. Results for XCS-LP on frog problem (legend shows curve order at right).

```
         CONDITION      ACT      WTVECTOR       ERR     FITN   NUM  EXPER

  0.  |0000000000|    1.00    1.00  -1.00     0.000    .997    76  28224
  1.  |000000000o|     .00    0.00   1.00     0.000    .970    67   8274
  2.  |....o000000|     .50    1.49  -0.98     0.003    .916    56   7463
  3.  |000000o....|     .50    0.50   0.98     0.003    .769    48   3057
  4.  |........oO|     .25    1.46  -0.69     0.010    .902    40   2584
  5.  |0000000o..|     .25    0.25   0.99     0.007    .567    38  21004
  6.  |00o.......|     .75    0.76   0.95     0.003    .942    38   1450
  7.  |........oo|     .25    1.47  -0.68     0.013    .260    28   2999
  8.  |..o00000000|     .75    1.23  -0.98     0.005    .555    26   5231
  9.  |0000000o..|     .25    0.25   0.99     0.007    .364    23    801
 10.  |.......oo.|     .25    1.50  -0.68     0.007    .478    21    880
 11.  |..o00000000|     .75    1.23  -0.97     0.013    .044    20   1088
 12.  |..o00000000|     .75    1.25  -1.00     0.000    .335    11   3776
 13.  |.........o|    1.00   -0.99   1.01     0.013    .000     1      6
 14.  |.........o|     .25    1.36  -0.65     0.198    .000     1     12
 15.  |.........o|     .50    1.23  -0.78     0.180    .000     1     10
 16.  |........oO|     .25    1.47  -0.69     0.014    .001     1     39
 17.  |000000000o.|     .25    0.25   0.99     0.007    .017     1    133
 18.  |.........o|     .75   -0.33  -0.34     1.658    .001     1      4
 19.  |.........o|     .00   -1.00  -1.00     0.005    .002     1      0
 20.  |000000o....|     .50    0.50   0.98     0.013    .008     1     22
```

Fig. 3. Entire population from one run of the experiment of Figure 2 ordered by decreasing numerosity. (ACTion, WeighTVECTOR, ERRor, FITNess, NUMerosity, EXPERience.)

5.2   XCS on the frog problem

Figure 4 shows frog problem results using standard XCS. Compared with XCS-LP, the payoff curve is essentially the same, reaching its steady value slightly quicker. The system error is significantly higher, approximately 0.03 vs. 0.004. The population size is much higher, averaging 192 classifiers at the end vs. 24.6, and does not decrease significantly over time. Generality is about 0.11 vs. 0.59 and also does not change over the experiment.

Figure 5 suggests the reason for these changed results. Shown are the 11 classifiers at the top of the numerosity ranking at the end of a typical run. They are very specific (interval predicates are small) as were the other 170 classifiers in this population. If all classifiers are quite specific, average population generality will be low, and the population size must be high in order to cover all inputs.

Thus XCS was capable of learning the frog problem, but at the cost of a large population of highly specific classifiers. In this problem the payoff function varied significantly with small changes in x. The scalar prediction of an XCS classifier can only stay accurate over a short interval, so that large numbers of them were required. Moreover, in this experiment the system error substantially exceeded ϵ_0, suggesting that the population was actually too small for good coverage. To check this we re-ran the experiment with $N = 2000$. Then the average system error fell to about 0.01, equaling ϵ_0. However, the classifiers were even more specific than before and the population size was 478.
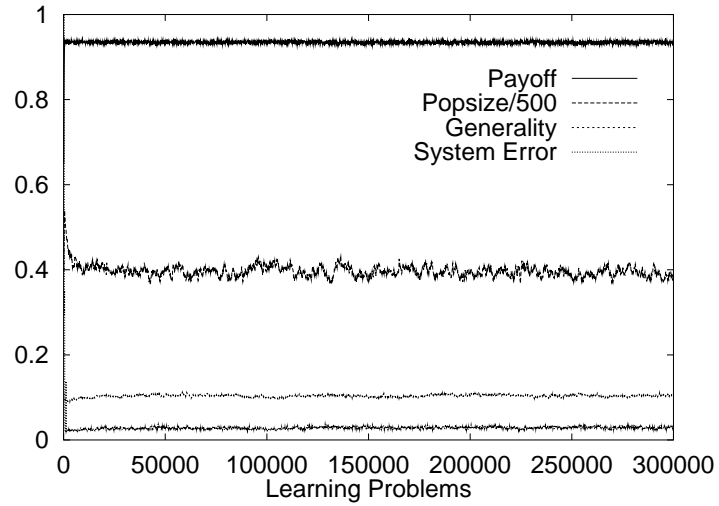
Fig. 4. Results for XCS on frog problem.

```
        CONDITION     ACT   PRED   ERR   FITN  NUM EXPER
 0.  |o.........|    1.00   .948  .011  .601   11  167
 1.  |oo........|     .50   .608  .020  .252    8  232
 2.  |..o.......|     .00   .238  .008  .551    7  46
 3.  |........oO|     .75   .332  .022  .148    7  83
 4.  |......o...|     .75   .587  .008  .461    7  23
 5.  |........oo|     .25   .845  .026  .208    7  37
 6.  |....oo....|     .50   .982  .013  .286    7  39
 7.  |......oOo.|     .25   .940  .023  .224    7  46
 8.  |oOOOo.....|     .75   .911  .036  .424    7  82
 9.  |..oo......|     .00   .319  .018  .238    7  15
10.  |..oOo.....|     .25   .552  .035  .385    7  11
```

Fig. 5. Top 11 of 181 classifiers from one run of experiment of Figure 4. (PRED = scalar prediction)

5.3 XCS-LP on a modified frog problem

The third experiment returned to XCS-LP but changed the sensory function to $x(d) = e^{-d}$. This changed the payoff function to

$$P(x,a) = \begin{cases} xe^a & : & a \leq -ln(x) \\ (1/x)e^{-a} & : & a \geq -ln(x) \end{cases} \tag{5}$$

Now the function—still continuous—is both nonlinear in having a tent-like structure and nonlinear in half of the tent "sides". However, the results shown in Figure 6 are similar to those of Figure 2. The evolution is slower and the final population somewhat larger (57.8). This is apparently due to the curvature of the tent sides, which requires additional classifiers to approximate accurately, as can be seen in listings of the population (omitted here for lack of space).
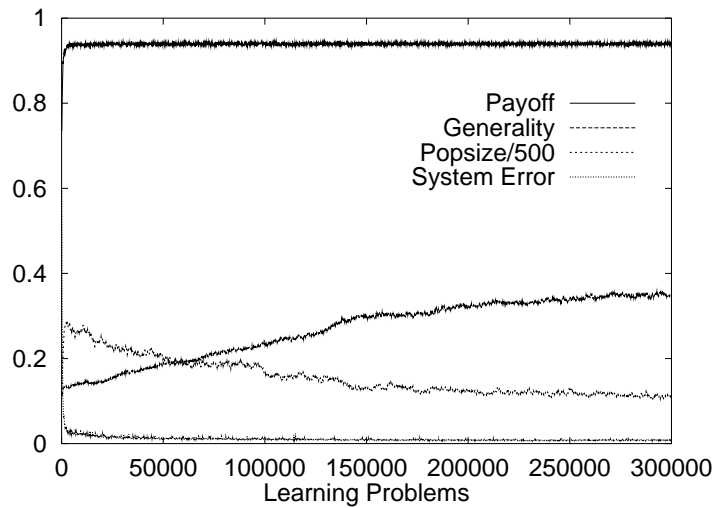

Fig. 6. Results for XCS-LP on modified frog problem.

6 Discussion

The experiments demonstrated a feasible classifier system technique for learning in environments where the payoff for a given action is a continuous and nonlinear function of the state x. In addition, the first and second experiments showed that, compared with traditional scalar predictions, *calculating* the predictions via a linear approximation to the payoff function can yield a substantial reduction in population size while increasing the transparency of the system's knowledge.

Further work is needed, in a variety of environments, to increase understanding of the technique. In particular, the evolution is quite slow, and there may be

other methods of updating the weight vectors that make it faster. The approximation technique needs also to be extended to cover the action variable(s), since in many environments (including the frog problem) the payoff is a continuous function of the action (Reynolds's "Hoverbeam" task [6] is an interesting further example). Moreover, besides linear functions, other approximation bases should be looked at.

In dealing with continuous payoff, XCS-LP advances the ability of classifier systems to *generalize*, that is, to evolve compact and at the same time readable representations of complex payoff functions. Traditionally, effort has focused on making classifier condition syntax more expressive. The present research points out a separate path to compact representation via viewing the prediction as a function of the input and then approximating it.

References

1. `mathworld.wolfram.com/ContinuousFunction.html`.
2. Larry Bull and Toby O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In W. B. Langdon et al, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911. Morgan Kaufmann, 2002.
3. Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Lanzi et al. [4], pages 253–272.
4. Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*. Springer-Verlag, Berlin, 2001.
5. Tom M. Mitchell. *Machine Learning*. WCB/McGraw Hill, Boston, MA, 1997.
6. Stuart I. Reynolds. A description of state dynamics and experiment parameters for the hoverbeam task. Technical report, University of Birmingham, School of Computer Science, 2000.
7. Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pages 126–134. The MIT Press, Cambridge, MA, 1988.
8. Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
9. Stewart W. Wilson. Get Real! XCS with Continuous-Valued Inputs. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Learning Classifier Systems. From Foundations to Applications*, volume 1813 of *LNAI*, pages 209–219, Springer-Verlag, Berlin, 2000.
10. Stewart W. Wilson. Function approximation with a classifier system. In Lee Spector et al, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 974–981. Morgan Kaufmann, 2001.
11. Stewart W. Wilson. Mining Oblique Data with XCS. In Lanzi et al. [4], pages 158–174.
12. Stewart W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1(2-3):211–233, 2002.