Classifiers that Approximate Functions

Stewart W. Wilson
Prediction Dynamics, Concord MA 01742, USA
Department of General Engineering,
The University of Illinois at Urbana-Champaign IL 61801, USA

Abstract. A classifier system, XCSF, is introduced in which the prediction estimation mechanism is used to learn approximations to functions. The addition of weight vectors to the classifiers allows piecewise-linear approximation, where the classifier's prediction is calculated instead of being a fixed scalar. The weight vector and the classifier's condition co-adapt. Results on functions of up to six dimensions show high accuracy. The idea of calculating the prediction leads to the concept of a generalized classifier in which the payoff prediction approximates the environmental payoff function over a subspace defined by the classifier condition and an action restriction specified in the classifier, permitting continuous-valued actions.

1. Introduction

There is considerable recent research on the properties and performance of the learning classifier system XCS (Wilson, 1995; Butz and Wilson, 2001). Areas of interest include data mining (e.g. Bernadó et al. 2001), generalization over inputs (Butz and Pelikan, 2001), self-adaptation of learning parameters (Hurst and Bull, 2001), and learning in non-Markov environments (Lanzi and Wilson, 2000), among others. One area that has not received attention is function approximation, which XCS's accuracy-based fitness makes possible. This paper demonstrates XCS as a function approximator, then shows how, somewhat surprisingly, this leads to a generalized classifier structure that embraces traditional classifier formats and, for the first time, permits classifiers capable of continuous-valued actions.

XCS's classifiers estimate payoff. That is, the rules (classifiers) evolved by XCS each keep a statistical estimate of the payoff (reward, reinforcement) expected if the classifier's condition is satisfied and its action is executed by the system. Moreover, the classifiers form quite accurate payoff estimates, or *predictions*, since the classifiers' fitnesses under the genetic algorithm depend on their prediction accuracies. In effect, XCS approximates the mapping $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$, where X is the set of possible inputs, A the system's set of available actions (typically finite and discrete), and P is the set of possible payoffs. If attention is restricted to a single action $a_i \in \mathrm{A}$, the mapping has the form $\mathrm{X} \times a_i \Rightarrow$

© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

P, which is a function from input vectors x to scalar payoffs. Thus the system approximates a separate function for each a_i.

In the reinforcement learning contexts in which classifier systems are typically used, the reason for forming these payoff function approximations is to permit the system to choose, for each x, the best (highest-paying) action from A. However, there are contexts where the output desired from a learning system is not a discrete action but a continuous quantity. For instance in predicting continuous time series, the output might be a future series value. In a control context, the output might be a vector of continuous quantities such as angles or thrusts. Apart from classifier systems based on fuzzy logic (Valenzuela-Rendón, 1991; Bonarini, 2000), there are none which produce real-valued outputs. Our hypothesis was that the payoff function approximation ability of XCS could be adapted to produce real-valued outputs, as well as be used for function approximation in general applications.

To test this we adapted XCS to learn approximations to functions of the form $y = f(x)$, where y is real and x is a vector with integer components $x_1, ..., x_n$. The results demonstrated approximation to high accuracy, together with evolution of classifiers that tended to distribute themselves efficiently over the input domain. The research fed back, however, on basic classifier concepts. It was realized that thinking of a classifier as a function approximator implied a new, generalized, classifier syntax that covered most existing classifier formats and included not only discrete-action classifiers but ones with continuous actions.

The next section gives a brief description of XCS. Section 3 describes modifications to XCS for function approximation. Section 4 has results on a simple piecewise-constant approximation. In Section 5 we introduce a new classifier structure that permits piecewise-linear approximations. Results on simple functions are shown in Section 6. In Section 7 we demonstrate accurate approximation of a six-dimensional function. Section 8 presents the generalized classifier syntax and examines its use. The final section has our conclusions and suggestions for future work.

2. XCS in Brief

The following, drawn from Wilson (2000a), is an abbreviated description of XCS. (For more detail, see Wilson (1995), Kovacs (1997), and Wilson (1998); Butz and Wilson (2001) gives an algorithmic description.) XCS is designed for both single- and multiple-step tasks, but the discussion here applies only to XCS for independent single-step tasks

in which an input is presented, the system makes a decision, and the environment provides some reward.

Structurally, each classifier C_j in XCS's population [P] has a *condition*, an *action*, and a set of associated parameters. The condition is a string from {0,1,#}; the action is an integer. The three principal parameters are: (1) *payoff prediction* p_j, which estimates the payoff the system will receive if C_j matches and its action is chosen by the system; (2) *prediction error* ϵ_j, which estimates the error in p_j with respect to actual payoffs received; and (3) *fitness* F_j, computed as later explained. It is convenient to divide the description of a single operating cycle or time-step into the traditional performance, update (reinforcement), and discovery components.

2.1. Performance

Upon presentation of an input—a binary string—XCS forms a *match set* [M] of classifiers in [P] whose conditions are satisfied by the input. The condition is satisfied if and only if each of its non-# positions is the same as the corresponding bit of the input string.

[If no classifiers match, the input is "covered" by creating matching classifiers with each of the possible actions, and placing them in [M]. Each new classifier's condition is a copy of the input except that #'s ("don't cares") are inserted with probability $P_\#$ per position. Parameters are given initial values. If [P] has reached its allowable maximum N, deletion of classifiers occurs to make room for the new ones (deletion is generally not necessary; initial populations are empty and covering normally occurs only at the very beginning of a run)].

Then, for each action a_k represented in [M], the system computes a fitness-weighted average P_k of the predictions p_j of each classifier in [M] having that action: $P_k = \sum_j F_j p_j / \sum_j F_j$. P_k is termed the *system prediction* for action k.

Next, XCS chooses an action from those represented in [M] and sends it to the environment. According to the *action-selection* regime in force, the action may be picked randomly or otherwise probabilistically based on the P_k, or it may be picked deterministically—i.e., the action with the highest P_k is chosen. Finally, an *action set* [A] is formed consisting of the subset of [M] having the chosen action.

2.2. Update

In this component, the parameters of the classifiers in [A] are re-estimated according to the *reward* R returned by the environment as a consequence of the system's taking action a_k. First, the predictions are updated: $p_j \leftarrow p_j + \beta(R - p_j)$. Next, the errors: $\epsilon_j \leftarrow \epsilon_j + \beta(|R - p_j|)$.

Third, for each C_j, an *accuracy* κ_j is computed: $\kappa_j = 0.1(\epsilon_j/\epsilon_0)^{-\nu}$, for $\epsilon_j > \epsilon_0$, else 1.0. The parameter ϵ_0 is termed the *error threshold*; ν is a positive integer. Note that classifiers in [A] with $\epsilon_j \leq \epsilon_0$ have equal accuracies. Fourth, from the κ_j, each classifier's *relative accuracy* κ'_j is computed: $\kappa'_j = \kappa_j / \sum_j \kappa_j$. Finally the fitnesses F_j are updated according to: $F_j \leftarrow F_j + \beta(\kappa'_j - F_j)$.

2.3. Discovery

On some time-steps, XCS executes a *genetic algorithm* (GA) within [A]. Two classifiers are chosen probabilistically based on their fitnesses and copied. The copies are crossed (two-point crossover) with probability χ, and then mutated with probability μ per allele. The resulting offspring are inserted into [P]; if the population size is already at its maximum value, N, two classifiers are deleted. The probability of deletion of a classifier is determined by Kovacs's 1999 method and is designed to preferentially remove low-fitness classifiers that have participated in a threshold number of action sets—that is, have had sufficient time for their parameters to be accurately estimated.

Whether or not to execute the GA on a given time-step is determined as follows. The system keeps a count of the number of time-steps since the beginning of a run. Every time a GA occurs, the classifiers in that [A] are "time-stamped" with the current count. Whenever an [A] is formed, the time-stamp values of its members are averaged and subtracted from the current count; if the difference exceeds a threshold θ_{GA}, a GA takes place.

A *macroclassifier* technique is used to speed processing and provide a more perspicuous view of population contents. Whenever a new classifier is generated by the GA (or covering), [P] is scanned to see if there already exists a classifier with the same condition and action. If so, the *numerosity* parameter of the existing classifier is incremented by one, and the new classifier is discarded. If not, the new classifier is inserted into [P]. The resulting population consists entirely of structurally unique classifiers, each with numerosity ≥ 1. If a classifier is chosen for deletion, its numerosity is decremented by 1, unless the result would be 0, in which case the classifier is removed from [P]. All operations in a population of macroclassifiers are carried out as though the population consisted of conventional classifiers; that is, the numerosity is taken into account. In a macroclassifier population, the sum of numerosities equals N, the traditional population size. [P]'s actual size in macroclassifiers, M, is of interest as a measure of the population's space complexity.

2.4. Overall Picture

XCS's overall objective is to maximize the rate of reward it obtains from the environment. If the environment (or trainer, teacher, etc.) gives high reward for actions (decisions) that it regards as "correct" or "best", then XCS's achieving a high reward rate will be equivalent to learning or solving the problem presented by the environment. This is the framework of *reinforcement learning* (Sutton and Barto, 1998). Decision problems (e.g., data inference) can be cast in terms of reinforcement learning if the rewards given the system correlate with the correctness of its decisions.

XCS is designed to evolve classifiers that accurately predict environmental reward in all the situations (inputs) the system might encounter. Having these it can choose the actions that maximize reward. XCS evolves accurate classifiers by a process of trial and error in which, given an input and a match set, a particular action is tried, the reward (if any) is received, and the associated classifiers are updated as previously described. Classifiers that predict accurately tend to be reproduced, while inaccurate classifiers are eventually deleted.

XCS not only evolves accurate classifiers, it evolves ones that are *maximally general*. That is, if several *different* input situations result in the same reward when a particular action is made by the system, then XCS will tend to evolve a *single* (macro)classifier having that action that matches all of those situations. As a result, the system achieves compactness of representation, and the evolved classifiers display the structure of the environment or problem in a way that is often readily interpretable by users. This important tendency of XCS occurs because if there are two equally accurate classifiers but one matches a subset of the states matched by the other, the more general classifier will win out since it occurs in more action sets and will have more reproductive opportunities (Wilson, 1995; Kovacs, 1997; Wilson, 1998; Wilson, 2000b).

2.5. Subsumption Deletion

The generalization mechanism just mentioned, while powerful, may sometimes not go as far as one would wish. Winning classifiers will be those that match more inputs—environmental *states*—than equally accurate rivals. Usually, a classifier whose condition is formally more general (has more #'s) will match more states. However, in certain environments (termed *sparse*), not all states may actually be present, with the result that some formally more general classifiers, while still perfectly accurate, cannot "drive out" their rivals. The resulting population will be larger than is strictly necessary to solve the problem, and

its knowledge will not be as perspicuous. To deal with sparseness and evolve classifiers that are as formally general as possible without sacrificing accuracy, an optional procedure called "subsumption deletion" was introduced (Wilson, 1998; Wilson, 2000b; Butz and Wilson, 2001).

Two forms of subsumption deletion are used. In *GA subsumption*, a new offspring is examined to see if its condition is logically subsumed by the condition of either of its parents. If so, and if the subsuming parent is both highly accurate and sufficiently experienced (its parameters have been updated a threshold number of times), the offspring is not added to the population but the parent's numerosity is incremented. *Action-set subsumption* is different from and independent of GA subsumption. Each action set is searched for the most general classifier that is both accurate and sufficiently experienced. Then all other classifiers in the set are tested against the general one to see if it subsumes them. Any that are subsumed are deleted from the population.

3. Modification for Function Approximation

For function approximation, XCS was modified in two respects (later, a third). The first was to adapt the program for integer instead of binary input vectors. The second, very simple, was to make the program's payoff predictions directly accessible at the output and to restrict the system to a single (dummy) action. The resulting program was called XCSF.

The changes to XCS for integer inputs were as follows (drawn from Wilson (2001)). 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 integers. 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.

Crossover (two-point) in XCSF 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 integer, *rand* picks an integer uniform randomly from $(0, m_0]$, and the sign is chosen uniform randomly. (Mutation by a fixed amount was tried, but a bounded random amount as above gave better performance.) If a new value of l_i is less than the minimum possible input value, in the present case 0, the new value is set to 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 integer from $[0, r_0]$, with r_0 a fixed integer.

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. Piecewise-Constant Approximation

The simplest way to approximate the function $y = f(x)$ with XCSF is to let x be the input and y the payoff. After sufficient sampling of the input space, the system prediction (the fitness-weighted average of matching classifiers' predictions (Sec. 2.1)) should, given an x, more or less accurately predict the corresponding y. In all XCS-like systems the fitness of a classifier depends on its accuracy of prediction, so that XCSF should converge to a population of classifiers that, over their respective input ranges, predict payoff well. The closeness of the approximation should be controllable with the error threshold ϵ_0 (Sec. 2.2), as follows. A classifier with prediction error ϵ_1 has higher accuracy than a classifier with prediction error ϵ_2 if $\epsilon_1 < \epsilon_2$ (Sec. 2.2). Since classifier *fitness* depends on accuracy, classifiers with lower errors will win out in the evolutionary competition. However, by definition, classifiers with errors *less* than ϵ_0 have accuracy 1.0, so no further fitness pressure applies. Thus ϵ_0 should limit the closeness of the approximation.

It is important that besides evolving accurate classifiers, the system employ the classifiers efficiently over the input domain. In slowly-changing or low-gradient regions of the function we would hope to evolve classifiers with relatively large interval predicates. The function value, and thus a given classifier's error, would change relatively little over such regions; so, as in XCS, a tendency toward accurate, maximally general conditions (Sec. 2.4) should cause the interval predicates to expand. Conversely, we should expect classifiers with small interval

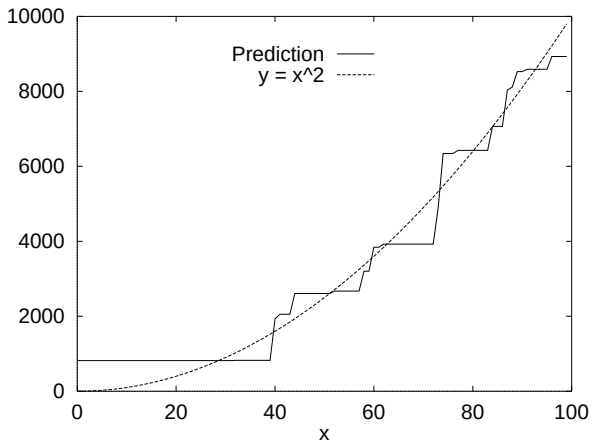

Figure 1. Piecewise-constant approximation to the parabola function $y = x^2$. $\epsilon_0 = 500$.

predicates where the function is changing rapidly. We also hope for an efficient *distribution* of classifiers over the domain in the sense that the tiling minimizes overlaps between classifier predicates.

Figure 1 shows typical results from an experiment in which XCSF approximated the function $y = x^2$, a parabola, over the interval $0 \leq x < 100$. (In all experiments reported here, the input value range was [0,99].) Plotted are XCSF's prediction for each possible x as well as the function itself.

The system learned from a dataset consisting of 1000 x, y pairs, with x chosen randomly and y the corresponding function value. In the experiment, a pair was drawn randomly from the dataset, its x value presented to XCSF as input, and the y value used as reward or reinforcement. XCSF formed a match set [M] of classifiers matching x and calculated the system prediction for each possible action in the usual way; since there was only one, dummy, action, just one system prediction was calculated and that became the system's output. An action set [A] was formed consisting of classifiers in [M] having the dummy action (i.e., all classifiers in [M]), and the predictions of the classifiers in [A] were adjusted using the reward, y, in the usual way; the other classifier parameters were also adjusted. A genetic algorithm was run in [A] if called for. This cycle was repeated 50,000 times after which the plot in Figure 1 was obtained by sweeping through all possible values of x and recording the resulting system predictions.

Parameter settings for the experiment were as follows, using the notation of Butz and Wilson (2001): population size $N = 200$, learning rate $\beta = 0.2$, error threshold $\epsilon_0 = 500$, 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 $\delta = 0.1$. In addition: mutation increment $m_0 = 20$ and covering interval $r_0 = 10$. GA subsumption was enabled, with time threshold $\theta_{GAsub} = 50$.

Several aspects of Figure 1 are of interest. The prediction curve has a "staircase" appearance typical of a piecewise-constant approximation. The height of the major "steps" varies between about 1000 and 2000. Examination of individual steps indicates an average error roughly consistent with the value of ϵ_0, suggesting that ϵ_0 is controlling the closeness of the approximation. The width of the steps is, again roughly, wider in the "flatter" part of the function and narrower in the steep part. Finally, it is significant that the prediction curve indeed takes the form of a staircase, instead of being smoother. Long flat "steps" suggest that one set of classifiers is in control (forms [M]) after which, on the next step, another set takes over. This in turn suggests a tendency toward efficient distribution of classifier resources over the domain.

Figure 2 gives another perspective. It is a listing of the (macro)-classifiers of the population at the end of the experiment, 15 in all. Shown are each classifier's condition, prediction, error, fitness, and numerosity. Note that most classifiers with substantial fitnesses have errors less than ϵ_0. A special graphic notation is used to represent the condition. Since x has just one component, the condition contains just one interval predicate. The possible range of x, 0-99, is divided into 20 equal subranges. An interval predicate is indicated by a cluster of "`0`"s that covers its range. If an interval predicate entirely covers a subrange, e.g., 35-39, an "`0`" is placed at that range's position. If the interval predicate covers some of but less than the whole of a subrange, a small "`o`" is put there. This notation has been found more perspicuous than using the raw numbers.

Note how, consistent with Figure 1, the classifier conditions are larger toward the beginning of the domain, where the function slope is lower. It is also interesting that the classifiers with higher fitnesses and numerosities cover the domain without a great deal of overlap. These classifiers dominate the calculation of the system prediction, since the latter is a fitness-weighted average of the predictions of matching classifiers. Because they dominate, the prediction curve takes the form of a staircase. Apart from the presence of the remaining, lower fitness, classifiers, the distribution of resources over the domain is thus relatively efficient.

In sum, XCSF succeeds in approximating the function in accordance with a stated error criterion (confirmed for additional values of ϵ_0) and the classifiers are employed reasonably well. Still, a piecewise-constant approximation is primitive compared with an approximation where the

	CONDITION	PRED	ERR	FITN	NUM
0.	|.....oO|	8992.	283.	.796	31
1.	|.....Oo.|	7876.	322.	.906	32
2.	|.....oOO...|	6162.	503.	.564	20
3.	|.....OOO...|	5990.	570.	.188	9
4.	|.....oOOO...|	5988.	569.	.028	1
5.	|.....oOOo...|	5668.	413.	.075	3
6.	|.....oOO....|	5534.	281.	.090	3
7.	|.....oOo....|	5423.	205.	.298	10
8.	|.....oOOOo...|	5053.	524.	.002	1
9.	|.....oOOo....|	4861.	465.	.054	5
10.	|.....OOOo.....|	3854.	468.	.862	30
11.	|.....oOOOO.....|	1481.	364.	.135	7
12.	|.....OOOOO.....|	1359.	377.	.358	18
13.	|OOOOOOOOo.....|	821.	378.	.372	15
14.	|OOOOOOOo.....|	820.	377.	.371	15

Figure 2. Classifiers from experiment of Figure 1. (PREDiction, ERRor, FITNess, NUMerosity.)

approximating segments more closely follow the function's contour. The simplest such approach is a *piecewise-linear* approximation. But how could a piecewise-linear approximation be done with a classifier system?

5. Piecewise-linear Approximation

Traditionally, a classifier's prediction is a number intended to apply for all inputs x that satisfy its condition. However, for function approximation, it would be desirable if the prediction could vary over the condition's domain, since the function being approximated generally varies. In effect, the prediction itself should be a function, the simplest form of which would be a linear polynomial in the input components, call it $h(x)$. The function $h(x)$ would substitute for the classifier's traditional (scalar) prediction, p. Then, given an input x, each matching classifier would *calculate* its prediction by computing $h(x)$.

For approximating a one-dimensional function $f(x)$, $h(x)$ would be a two-term polynomial $h(x) = w_0 + w_1 x_1$. In this case, w_1 can be thought of as the slope of an approximating straight line, with w_0 its intercept. For an n-dimensional $f(x)$, $h(x) = w \cdot x'$, where w is a *weight vector* $(w_0, w_1, ..., w_n)$ and x' is the input vector x augmented by a constant x_0, i.e., $x' = (x_0, x_1, ..., x_n)$. In this case $h(x)$ computes a *hyperplane approximation* to $f(x)$. Classifiers would have different weight vectors w since in general the domains 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. One approach would be to use an evolutionary algorithm. The weight vector would be evolved along with the classifier condition. For this, the w_i could be concatenated with the interval predicates of the condition and the whole thing evolved as a unit. However, optimizing the weights of a linear network is unimodal, so adapting the weight vector separately using a simple gradient technique seemed likely to be faster than an evolutionary algorithm. We used a modification of the *delta rule* (Mitchell, 1997), since we had some experience with that method.

The delta rule is given by

$$\Delta w_i = \eta(t - o)x_i, \tag{1}$$

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 correct value of y according to $y = f(x)$. 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

$$\Delta o = \Delta w \cdot x' = \eta(t - o)|x'|^2. \tag{2}$$

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. After some experimentation with this issue, we noticed that in its original use (Widrow and Hoff, 1988), 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:

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

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 Widrow and Hoff. In the experiments that follow, we used the modified delta rule with various values of $\eta \leq 1.0$.

Use of a delta rule requires selection of an appropriate value for x_0, the constant that augments the input vector. In tests, 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

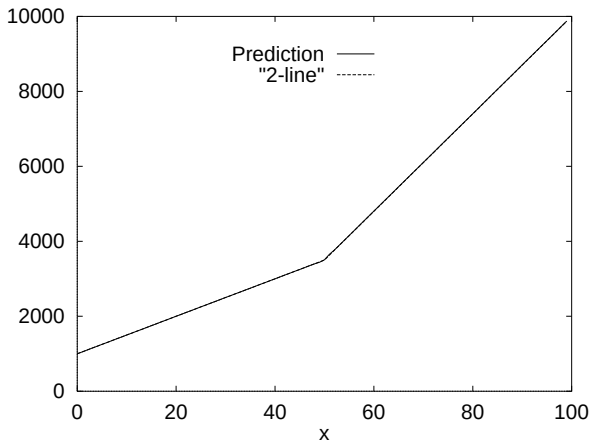

Figure 3. Piecewise-linear approximation to piecewise-linear function "2-line". $\epsilon_0 = 10$.

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 $x_0 = 100$—the same order of magnitude as the other x_i—appeared to solve the problem.

For piecewise-linear approximation, no changes were necessary to XCSF except for addition of the weight vectors to the classifiers, and provision for calculation of the predictions and application of the modified delta rule to the action set classifiers on every time-step. 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. Both policies yielded performance improvements over other initializations. In the experiments, most parameter settings were the same as those given in Section 4; differences will be noted. Settings of the new parameter, η, will be given.

6. Tests on Simple Functions

Preliminary testing was carried out approximating functions that were themselves linear or piecewise linear. For example, tests were done on the function "2-line", defined as

$$\begin{aligned} y &= 50x + 1000, \quad 0 \leq x < 50 \\ &= 130x - 3000, \quad 50 \leq x < 100. \end{aligned}$$

Parameters for the experiment were the same as previously, except for $N = 800$, $\epsilon_0 = 10$, $\theta_{GAsub} = 100$, and $\eta = 0.4$. The approximation

```
          CONDITION          PRED   ERR  FITN NUM

0. |.........oOOOOOOOOOO| 4705.  19. .158  62
1. |..........oOOOOOOOOO| 4670.   0. .706 274
2. |..........oOOOOOOOOO| 4670.   0. .044  18
3. |..........OOOOOOOOOO| 4670.   0. .090  29
4. |OOOOOOOOOo.........| 3057.  16. .005   6
5. |OOOOOOOOOo.........| 3050.   0. .676 247
6. |OOOOOOOOOo.........| 3050.   0. .377 164
```

Figure 4. Classifiers from experiment of Figure 3

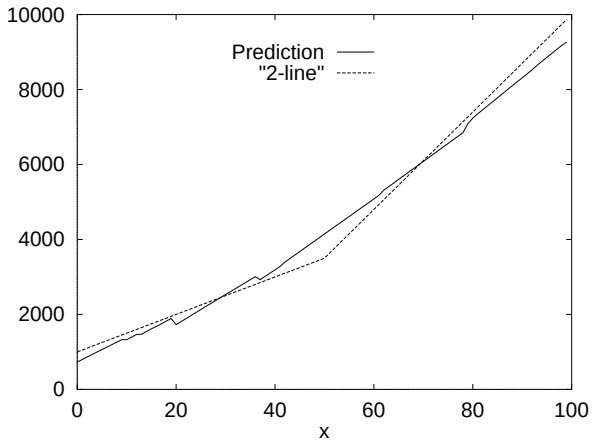

Figure 5. Approximation to "2-line" with $\epsilon_0 = 500$

obtained (Figure 3) was so close that the plots of the prediction and the function itself appear identical.

Figure 4 shows the seven classifiers at the end of the run[1]. They are clearly divided into one group for the upper segment of the function and another for the lower segment. The dominant classifier in the upper group, no. 1, has error zero and a predicate covering the interval 52-99 (inclusive). In the lower group, the dominant classifier, no. 5, also has error zero and covers the interval 0-50.

Raising the error threshold caused the approximation to deteriorate. In Figure 5, $\epsilon_0 = 500$. The prediction curve seems to ignore the break at $x = 50$, reflecting the change in slope only gradually. The classifier list showed several that bridged the break point, with predictates from about 30 to 70. Evidently, with a larger error threshold, the system

[1] The prediction values represent the most recent weight-vector calculation and are not particularly significant. As in Section 4, this and all following experiments were run for 50,000 input cycles.

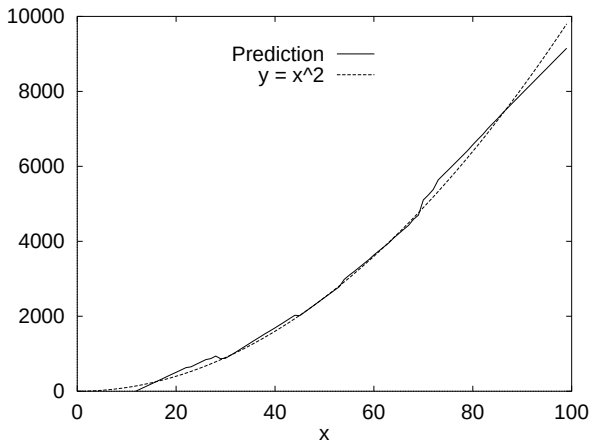

Figure 6. Piecewise-linear approximation to parabola with $\epsilon_0 = 500$.

```
        CONDITION          PRED   ERR   FITN NUM

0. |.........OOOOOOOOOOO| 9375. 246. .286 151
1. |......OOOOOOOOOOOOOO| 9014. 431. .122  54
2. |.....oOOOOOOOOOOOOOO| 8985. 432. .176  80
3. |.....oOOOOOOOOOOOOOO| 8906. 456. .037  37
4. |....oOOOOOOOOOOOOOOO| 8744. 456. .035  33
5. |OOOOOOOOOOOOOo.....| 4184. 379. .077  35
6. |OOOOOOOOOOOOO......| 3756. 247. .258 122
7. |OOOOOOOOOOOOo......| 3564. 256. .066  32
8. |OOOOOOOOOOOo.......| 3193. 262. .035  18
9. |OOOOOOOOOo.........| 2250. 204. .211 125
```

Figure 7. High fitness classifiers from experiment of Figure 6.

was not forced, as in Figure 3, to evolve classifiers that corresponded closely to the two function segments.

Figures 6 through 12 show typical results on parabola and sine functions. Parameters were the same as in the "2-line" experiments except $\eta = 0.2$ (an insignificant difference). Values for ϵ_0 are given in the captions. The two values chosen for each function are equivalent to 5% and 1% of the functions' ranges. To highlight the dominant classifiers and save space, the classifier lists include only the highest-fitness classifiers; the full populations are three or four times larger.

While the parabola figures show good linear approximations with a small number of classifiers, it is somewhat surprising that—unlike the piecewise-constant case—the sizes of the interval predicates do not seem to reflect the function's slope. Perhaps for piecewise-linear approximation a different analysis is in order: predicate length may be more related to curve straightness than steepness. For the sinewaves,

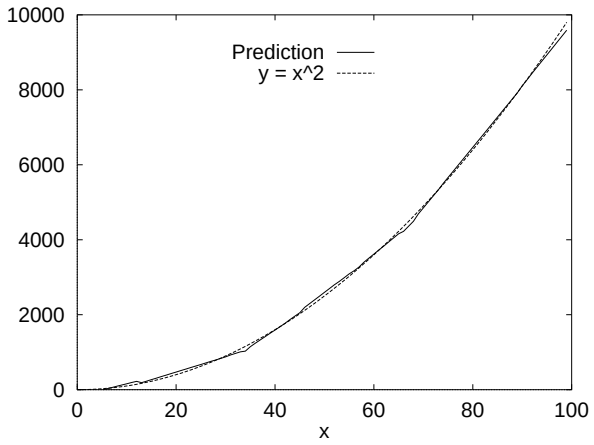

Figure 8. Piecewise-linear approximation to parabola with $\epsilon_0 = 100$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|.....oOOOOOO|	9591.	77.	.823	212
1.	|.....oOOOOOO..|	7734.	75.	.081	21
2.	|.....oOOOOOO..|	7687.	89.	.035	9
3.	|.....oOOOOOO.....|	4438.	79.	.731	184
4.	|.....oOOOO.....|	3149.	26.	.123	41
5.	|..oOOOOOO.....|	1868.	79.	.228	39
6.	|..oOOOOO.....|	1621.	61.	.072	13
7.	|OOOOOO.....|	967.	104.	.607	219
8.	|OOOOOO.....|	749.	85.	.074	16

Figure 9. High fitness classifiers from experiment of Figure 8.

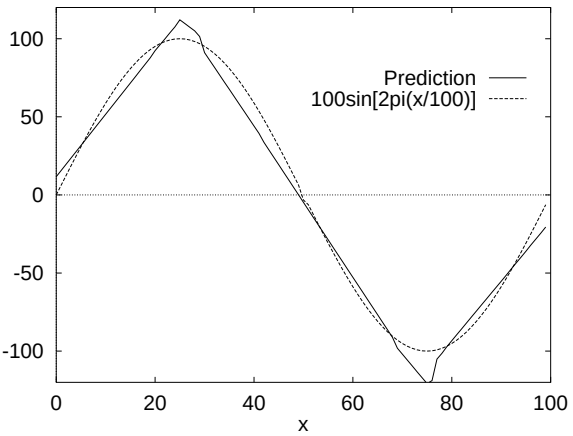

Figure 10. Piecewise-linear approximation to sine function with $\epsilon_0 = 10$.

	CONDITION	PRED	ERR	FITN	NUM
0.	|000000.....|	120.	10.	.292	76
1.	|00000o.....|	118.	10.	.083	22
2.	|00000.....|	111.	7.	.572	158
3.	|....0000o.....|	66.	5.	.032	15
4.	|.....o0000|	-20.	4.	.792	173
5.	|.....o000000|	-23.	12.	.101	78
6.	|.....0000000000o....|	-130.	8.	.888	234

Figure 11. High fitness classifiers from experiment of Figure 10.

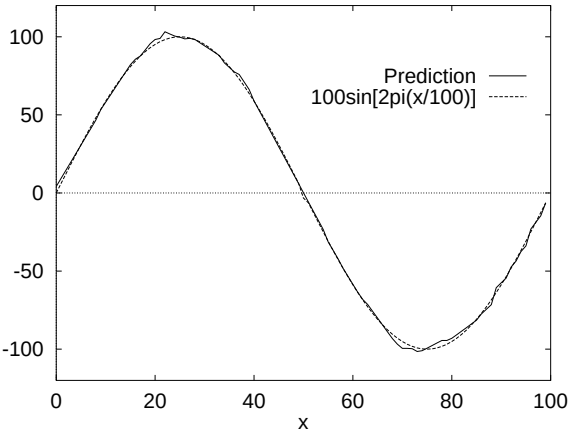

Figure 12. Piecewise-linear approximation to sine function with $\epsilon_0 = 2$.

the approximation overshoots the peaks when ϵ_0 is large (Figure 10), but this effect disappears with smaller values and the curve is quite nicely matched (Figure 12). Figure 11 suggests that the system divides the approximation into classifiers for the beginning, middle, and end of the curve.

7. Multi-dimensional Input

XCSF was initially tested on functions of more than one variable by letting $y = f(x_1, ..., x_n)$ be a linear function, i.e., a hyperplane function. The system rapidly evolved solutions with one or a few classifiers and arbitrary accuracy. This was expected, since the classifiers' weight vectors are effectively linear functions. To test XCSF on a multi-dimensional nonlinear function, we chose, somewhat arbitrarily, $y = [(x_1^2 + ... + x_n^2)/n]^{1/2}, 0 \leq x_i < 100$, a sort of "rms" function. Experiments with $n = 3$ went well, so $n = 6$ was tried. Parameters were the same as previously, except $N = 3200$, $\beta = 0.1$, $\theta_{GAsubs} = 200$, and

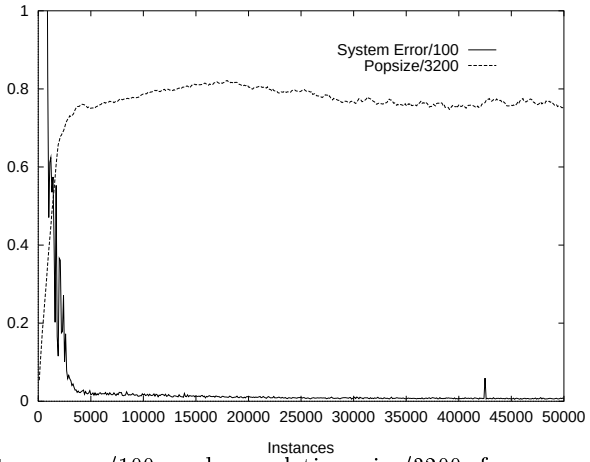

Figure 13. System error/100 and population size/3200 for approximation to six-dimensional "rms" function. $\epsilon_0 = 1$.

$\eta = 1.0$. The error threshold $\epsilon_0 = 1$ (1% of the range). In contrast to previous experiments in which instances were chosen randomly from a fixed data set, instances were picked randomly from the domain. Figure 13 plots the system error and population size.

Starting initially very high, the system error (a moving average of the absolute difference between XCSF's prediction and the actual function value) fell rapidly to less than 1 (or .01 as plotted on this graph). The population size—in macroclassifiers—rose quickly to about 2400 and stayed there. The system seemed to have little difficulty approximating the function to within 1%, though quite a few classifiers were required.

8. Generalized Classifiers

8.1. Definition

In XCS, as in other classifier systems, the classifier prediction is a scalar, and the system adapts the classifier conditions and the prediction scalars to find accurate classifiers that are as general as possible. In XCSF, the prediction was replaced by a weight vector computing a linear function, leading to a more powerful and subtle co-adaptation of the condition and the prediction. As an extreme but instructive example, XCSF can approximate a very high-dimensional linear function with $O(1)$ classifiers, far less than required using scalar predictions.

What do the results with XCSF suggest for classifier architecture? The essential novelty in XCSF is that the prediction is *calculated*, instead of being a fixed scalar. The prediction is a linear function of x,

the input vector. More precisely, the prediction function *approximates* a desired output function over the input subdomain defined by the classifier's condition. Other classifiers approximate other parts of the desired output function. Taken together, the classifiers evolve to approximate, within a specified error criterion, the input-output mapping defined by $y = f(x)$.

In the Introduction it was noted that XCS approximates the mapping $\mathrm{X}\times\mathrm{A} \Rightarrow \mathrm{P}$, which includes actions as well as inputs. But this is just another way of saying that the desired output y (a payoff prediction) is a function of both the input x and the action a taken by the system. We can certainly imagine approximating y with a linear function of both x and a. Further, we can define a *generalized classifier*

$$t(x) : r(a) \Rightarrow p(x, a) \tag{4}$$

that expresses these ideas. The generalized classifier says: "Within the subdomain of $\mathrm{X} \times \mathrm{A}$ defined by the truth function $t(x)$ and the *action restriction* $r(a)$, the payoff is approximated by the *prediction function* $p(x, a)$."

The components of (4) are as follows. The truth function $t(x)$ defines the subdomain of X within which the classifier applies ("matches", in the traditional terminology). For example in a binary input space $t(x)$ could be simply "01#0", meaning of course the set of strings {0110, 0100}. Or $t(x)$ could be an S-expression such as $(> x_1\ x_3)$ (Lanzi, 1999). It could also be a representation of a neural network (Bull and O'Hara, 2001). Whatever its specific form, a classifier's $t(x)$ determines, given an x, whether or not the classifier applies.

The action restriction $r(a)$ defines a *range of effector values*. For example, $r(a)$ might specify a range of rudder angles between -10 and +34 degrees. More generally, if there were more than one effector (e.g. rudder, aileron, throttle) $r(a)$ is an expression that defines a subdomain of the effector space: a set of effector vectors each taking values, e.g., rudder and aileron angles and throttle positions. In a very simple special case, $r(a)$ would just name an action, such as "turn left", as occurs in traditional classifier systems. Whatever its specific form, a classifier's $r(a)$ specifies an allowed set of effector values should the system decide to act according to this classifier, as will be described.

Finally, the prediction function $p(x, a)$ serves to calculate the classifier's payoff prediction just in case $t(x)$ is satisfied and the system intends to take an action a (generally a vector) from $r(a)$. According to its form—e.g. a linear expression in x and a or a combination of other basis functions—$p(x, a)$ calculates a prediction, which is of course an approximation of expected payoff to within a given error criterion. In XCS, of course, $p(x, a)$ is simply a constant. Consider, however, a robot

taking fixed-length steps in a plane, but able to turn through any angle. If the robot were supposed to learn to orient and move toward a goal, environmental payoff would probably depend on a (and maybe x), so that the prediction function $p(x, a)$ would depend at least on a. More generally, the prediction function of a given classifier would depend on some or all of the input and effector variables.

8.2. Operation

How would generalized classifiers be used in the system's performance, reinforcement, and discovery cycles? In performance, the main step would be to calculate a payoff prediction for each classifier that matches the input x. Since the prediction function $p(x, a)$ also depends on a, what value for a should be used? The answer, if the system is in an *exploit* mode (i.e., seeking to act on its best information so as to maximize payoff), is to choose a such that this classifier's payoff is maximized. Thus

$$a^* = \max_{a \in r(a)} p(x^*, a), \tag{5}$$

where a^* is the chosen action and x^* is the current input. In many situations this is not a lengthy calculation. For instance, if $p(x, a)$ is linear in x and a, e.g. of the form

$$w_0 x_0 + w_1 x_1 + ... + w_n x_n + w_{n+1} a_1 + ... + w_{n+m} a_m,$$

where the a_i correspond to separate effectors, the maximum is obtained by choosing each a_i so that the associated term is maximized. Once it had calculated a^* and the associated $p(x^*, a^*)$ for each classifier in the match set [M], the system would carry out the action whose prediction was highest.

If, on the other hand, the system is in an *explore* mode (seeking to try new actions in hopes of gaining new information about environmental payoff), a^* for each classifier in [M] would be selected at random from $r(a)$, or according to some other exploration regime. The system would then randomly choose one of the resulting actions for execution in the environment.

The reinforcement or update cycle would consist of updating the classifier whose action was taken (often, updating occurs only in explore mode). The update would occur exactly as in XCSF: the prediction function would be adjusted according to the difference between its prediction and the actual payoff received; then the error and fitness values would be adjusted as in XCS.

The discovery cycle would also occur as in XCSF, except that genetic operators, besides acting on the truth function $t(x)$, would also act on

the action restriction $r(a)$. For this the syntax of $r(a)$ must of course be amenable to the genetic operators. For example, if the action restriction was of the form

$$(v_{11} \leq a_1 \leq v_{12}) \ ... \ (v_{m1} \leq a_m \leq v_{m2}),$$

and this was encoded by concatenating the (scalars) v_{ij}, the genetic operators would apply exactly as they do in XCSF.

8.3. A Special Case

To illustrate the generality of (4), it is interesting to see how the Boolean multiplexer problem (Wilson, 1995) might be represented. As is quite well known, in a typical learning experiment on the 6-multiplexer, XCS evolves pairs of classifiers such as

$$10\#\#1\# : 1 \ \Rightarrow \ 1000$$
$$10\#\#1\# : 0 \ \Rightarrow \ 0$$

Each is accurate and maximally general (i.e. can't be generalized further without losing accuracy). There are *two* classifiers for the same condition because the two possible actions—1 or 0—result in different payoffs, and XCS evolves accurate classifiers whether or not they are "correct" (payoff 1000) or "wrong" (payoff 0).

In the generalized representation, however, the above pair would be covered by a single classifier

$$10\#\#1\# : \# \ \Rightarrow \ 1000a,$$

where $r(a)$ is "#", meaning a can take on either 1 or 0, and $p(x, a)$ is the linear expression $1000a$. It can be seen that substituting 1 or 0 in $p(x, a)$ results in the correct payoff for that action. Thus, in this example, the binary-input, discrete action, step-wise payoff case is fully captured by the generalized representation.

8.4. Continuous Actions

In what sense does the generalized classifier of (4) lead to a system capable of continuous actions? As discussed in Sec. 8.2 a classifier is in principle capable of advocating any action value permitted by $r(a)$. But in practice, when the system is in exploit mode, the classifier will choose just the action that maximizes its prediction. Thus for a given state of the classifier system a certain discreteness of actions is maintained. However, the degree of discreteness depends on the quality of the prediction function approximations: the better the approximations,

the less discretized the action space, since each classifier will cover a smaller portion of that space. Therefore, one can perhaps say that the generalized classifier format permits continuous actions in the limit.

Why are continuous actions desirable in the first place? It is desirable for a system to be able to choose actions that maximize the payoff available from the environment. In continuous environments, payoff-maximizing actions (precise angles of turns, etc.) will be difficult for a system designer to anticipate, and so it is important that the system can choose them itself to within the limits of its resources—e.g. the quality of approximation permitted by its population size. In general this will be better than any predefined set of fixed actions. However, at some point the actions chosen by the generalized classifier system will be fine-grained enough, so that further resolution—or perfect continuity—will not add significantly to payoff returns.

9. Summary and Conclusions

This paper introduced a classifier system, XCSF, designed to learn approximations to functions. The prediction estimation mechanism was used to form the approximations: given an input vector x, the value y of the function to be approximated was treated as a payoff to be learned. In its first incarnation, XCSF produced piecewise-constant approximations. A more advanced version added a weight vector to each classifier, permitting the approximation to be piecewise-linear. Tests on simple one-dimensional functions yielded arbitrarily close approximations, according to the setting of an error parameter. The system tended to evolve classifiers that distributed themselves reasonably efficiently over the function's domain, though some overlap occurred together with the presence of a moderate number of redundant low-fitness classifiers. In limited tests on a six-dimensional nonlinear function, XCSF rapidly formed highly accurate approximations, though the number of classifiers required was much larger than for the one-dimensional functions.

Future work should continue with multi-dimensional functions, to determine the technique's general viability and estimate its complexity in terms of learning time and resources (classifiers) required. Since XCSF approximates linear functions effortlessly, regardless of dimensionality, it is likely that the complexity will relate to the degree of "smoothness" or "flatness" in hyperspace that the function exhibits. Comparisons should be made with fuzzy classifier systems, which appear to be quite different in concept: the output of a fuzzy system

is computed jointly by more than one classifier, whereas in XCSF an accurate output can in principal be computed by just one.

Function approximation with XCSF could be useful for on-line learning of any function or mapping from a vector of input values to an output value. An example would be financial time-series prediction, where a future price is presumably an approximable function of known prices or other quantities at earlier times in the series.

Piecewise-linear function approximation in XCSF is based on the idea of *calculating* a classifier's prediction, and this leads to the concept of a generalized classifier in which the condition is a truth function $t(x)$ of the input x and the prediction is an approximation function $p(x, a)$ that depends on x and an action a. Such a classifier would apply in the subspace of the $\mathrm{X} \times \mathrm{A} \Rightarrow \mathrm{P}$ mapping defined by $t(x)$ and an action restriction $r(a)$ specified in the classifier. Given x, the best action to take would be determined by maximizing $p(x, a)$ over the restriction's range. This architecture would appear to permit the system to evolve classifiers that take actions that optimally match the environment's payoff landscape—instead of generally suboptimal actions from a prespecified finite set.

Since we have only presented generalized classifiers conceptually, the next step is to do experiments, especially in domains where they would appear to have advantages. The first advantage is of course continuous vs. discrete actions, so tests in continuous robotic environments, either actual or simulated, are in order. The second advantage lies in the potential generalization power of generalized classifiers. If the payoff function relates to x and a in a way that is readily handled by piecewise-linear approximation, then a generalization advantage should be expected compared with conventional XCS systems. This means savings in space complexity (population size) as well as increased transparency in the system's model. Furthermore, approximation bases other than linear should be tried where promising.

Acknowledgement

This work was supported in part by NuTech Solutions Inc.

References

Banzhaf, W., J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith (eds.): 1999, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99).* Morgan Kaufmann: San Francisco, CA.

Bernadó, E., X. Llorà, and J. M. Garrell: 2001, XCS and GALE: A Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. In (Spector, 2001), pp. 337–341.

Bonarini, A.: 2000, An Introduction to Learning Fuzzy Classifier Systems. In (Lanzi et al., 2000), pp. 83–104, Springer-Verlag.

Bull, L. and T. O'Hara: 2001, A Neural Rule Representation for Learning Classifier Systems. In (Spector, 2001).

Butz, M. V. and M. Pelikan: 2001, Analyzing the Evolutionary Pressures in XCS. In: L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke (eds.): *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. pp. 935–942, Morgan Kaufmann: San Francisco, CA.

Butz, M. V. and S. W. Wilson: 2001, An Algorithmic Description of XCS. In (Lanzi et al., 2001), Springer-Verlag.

Hurst, J. and L. Bull: 2001, A Self-Adaptive XCS. In (Spector, 2001), pp. 357–361.

Kovacs, T.: 1997, XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions.. In: Roy, Chawdhry, and Pant (eds.): *Soft Computing in Engineering Design and Manufacturing*. pp. 59–68, Springer-Verlag, London.

Kovacs, T.: 1999, Deletion Schemes for Classifier Systems.. In (Banzhaf et al., 1999), pp. 329–336, Morgan Kaufmann: San Francisco, CA.

Lanzi, P. L.: 1999, Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In (Banzhaf et al., 1999), pp. 345–352, Morgan Kaufmann: San Francisco, CA.

Lanzi, P. L., W. Stolzmann, and S. W. Wilson (eds.): 2000, *Learning Classifier Systems. From Foundations to Applications*, Vol. 1813 of *LNAI*. Berlin: Springer-Verlag.

Lanzi, P. L., W. Stolzmann, and S. W. Wilson (eds.): 2001, *Advances in Learning Classifier Systems: Third International Workshop, IWLCS-2000*, Vol. 1996 of *LNAI*. Berlin: Springer-Verlag.

Lanzi, P. L. and S. W. Wilson: 2000, Toward Optimal Classifier System Performance in Non-Markov Environments. *Evolutionary Computation* **8**(4), 393–418.

Mitchell, T. M.: 1997, *Machine Learning*. Boston, MA: WCB/McGraw Hill.

Spector, L. (ed.): 2001, *Proceedings of the 2001 Genetic and Evolutionary Computation Conference Workshop Program*. San Francisco, CA: .

Sutton, R. S. and A. G. Barto: 1998, *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press/Bradford Books.

Valenzuela-Rendón, M.: 1991, The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables. In: *Proceedings of the 4th International Conference on Genetic Algorithms (ICGA91)*. pp. 346–353.

Widrow, B. and M. E. Hoff: 1988, Adaptive Switching Circuits. In: J. A. Anderson and E. Rosenfeld (eds.): *Neurocomputing: Foundations of Research*. Cambridge, MA: The MIT Press, pp. 126–134.

Wilson, S. W.: 1995, Classifier Fitness Based on Accuracy. *Evolutionary Computation* **3**(2), 149–175.

Wilson, S. W.: 1998, Generalization in the XCS classifier system. In: J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo (eds.): *Genetic Programming 1998: Proceedings of the Third Annual Conference*. pp. 665–674, Morgan Kaufmann: San Francisco, CA.

Wilson, S. W.: 2000a, Get Real! XCS with Continuous-Valued Inputs. In (Lanzi et al., 2000), pp. 209–219, Springer-Verlag.
Wilson, S. W.: 2000b, State of XCS Classifier System Research. In (Lanzi et al., 2000), pp. 63–82, Springer-Verlag.
Wilson, S. W.: 2001, Mining Oblique Data with XCS. In (Lanzi et al., 2001), Springer-Verlag.