Extending XCSF Beyond Linear Approximation

Pier Luca Lanzi*†, Daniele Loiacono*, Stewart W. Wilson†‡, David E. Goldberg†
*Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. P.za L. da Vinci 32, I-20133, Milano, Italy
†Illinois Genetic Algorithm Laboratory (IlliGAL)
University of Illinois at Urbana Champaign, Urbana, IL, 61801
‡Prediction Dynamics, Concord, MA 01742, USA
lanzi@elet.polimi.it, loiacono@elet.polimi.it, wilson@prediction-dynamics.com, deg@illigal.ge.uiuc.edu

ABSTRACT

XCSF is the extension of XCS in which classifier prediction is computed as a linear combination of classifier inputs and a weight vector associated to each classifier. XCSF can exploit classifiers' computable prediction to evolve accurate piecewise linear approximations of functions. In this paper, we take XCSF one step further and show how XCSF can be easily extended to allow polynomial approximations. We test the extended version of XCSF on various approximation problems and show that quadratic/cubic approximations can be used to significantly improve XCSF's generalization capabilities.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics Based Machine Learning, Learning Classifier Systems; G.1.2 [**Approximation**]: Linear approximation, Least squares approximation

General Terms

Algorithms, Performance

Keywords

LCS, XCS, Function Approximation, Least Squares

1. INTRODUCTION

XCSF [15] extends the typical concept of learning classifier systems through the introduction of a computable classifier prediction. In XCSF classifier prediction is not memorized into a parameter but computed as a linear combination of the current input and a weight vector associated to each classifier. Wilson [15] applied XCSF to simple function approximation problems showing that computable prediction can be used to evolve accurate piecewise linear approximations of a target function. In this paper we extend previous results and show that XCSF can be easily extended to include polynomial approximations. For this purpose, we consider the version of XCSF with linear least squares update recently introduced in [10] (i) to speed up the convergence of classifier weights and (ii) to improve the generalization capabilities of XCSF. We show how XCSF can be easily extended for generic polynomial approximations by adding a simple preprocessing step to the usual XCSF's weight update procedure [15]. To extend XCSF to evolve polynomial approximations of degree k, given the current input $\vec{x} = \langle x_1 \dots x_n \rangle$, classifiers weights are updated with respect to the input $\vec{y}$ obtained by enriching $\vec{x}$ with power terms, that is $\vec{y} = \langle x_1, x_1^2 \dots x_1^k, \dots, x_n \dots x_n^k \rangle$. The update of classifier weights is performed according to the usual procedures, that is, Widrow-Hoff [15] or linear least squares [10]. We apply XCSF with quadratic and cubic approximation to approximate functions taken from the literature such as the three sine and the four sine problems [9], the polynomial $1 + x + x^2 + x^3$ from [8], and other functions from [16]. We show that the generalization mechanism of XCSF can exploit quadratic and cubic approximations to evolve solutions that are significantly more accurate and significantly more compact.

The paper is organized as follows. We begin in Section 2 with a description of XCSF [15] and then in Section 3 we briefly introduce linear least squares for XCSF [10] which we use in all the experiments discussed in this paper. In Section 4 we illustrate the general approach we follow to extend XCSF to allow polynomial approximations. In Section 6, we compare XCSF with linear approximation and XCSF with quadratic and cubic approximations on several approximation problems. The results are statistically analyzed in Section 7 showing that the improvement in accuracy and generalization obtained through quadratic/cubic approximations is statistically significant. Future research directions and additional issues related to polynomial approximation with XCSF are discussed in Section 8.

2. THE XCSF CLASSIFIER SYSTEM

XCSF extends XCS in three respects [15]: (i) classifier conditions are extended for numerical inputs, as done for XCSI [14]; (ii) classifiers are extended with a vector of weights w, that are used to compute classifier's prediction; finally, (iii) the weights w are updated instead of the classifier prediction.

Classifiers. In XCSF, classifiers consist of a condition, an action, and four main parameters. The condition specifies which input states the classifier matches; it is represented

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'05, June 25–29, 2005, Washington, DC, USA.
Copyright 2005 ACM 1-59593-010-8/05/0006 ...$5.00.

by a concatenation of interval predicates, $int_i = (l_i, u_i)$, where l_i ("lower") and u_i ("upper") are integers, though they might be also real. The action specifies the action for which the payoff is predicted; when XCSF is used as a pure function approximator (like in [15] and in this paper), there is only one dummy action which has no actual effect. The four parameters are: the weight vector $\vec{w}$, used to compute the classifier prediction as a function of the current input; the prediction error ε, that estimates the error affecting classifier prediction; the fitness F that estimates the accuracy of the classifier prediction; the numerosity *num*, a counter used to represent different copies of the same classifier. The weight vector $\vec{w}$ has one weight w_i for each possible input, and an additional weight w_0 corresponding to a constant input x_0, that is set as a parameter of XCSF.

Performance Component. XCSF works as XCS. At each time step t, XCSF builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input s_t; if [M] contains less than θ_{mna} actions, *covering* takes place as in XCSI [14, 15]. The weight vector w of covering classifiers is initialized with zero values (note that originally [15], the weights were initialized with random values in [-1,1]); all the other parameters are initialized as in XCS (see [3]).

For each action a_i in [M], XCSF computes the *system prediction*. As in XCS, in XCSF the *system prediction* of action a is computed by the fitness-weighted average of all matching classifiers that specify action a. In contrast with XCS, in XCSF classifier prediction is computed as a function of the current state s_t and the classifier vector weight w. Accordingly, in XCSF system prediction is a function of both the current state s and the action a. Following a notation similar to [3], the system prediction for action a in state s_t, $P(s_t, a)$, is defined as:

$$P(s_t, a) = \frac{\sum_{cl \in [M]|_a} cl.p(s_t) \times cl.F}{\sum_{cl \in [M]|_a} cl.F} \quad (1)$$

where cl is a classifier, $[M]|_a$ represents the subset of classifiers in [M] with action a, $cl.F$ is the fitness of cl; $cl.p(s_t)$ is the prediction of cl in state s_t, which is computed as:

$$cl.p(s_t) = cl.w_0 \times x_0 + \sum_{i>0} cl.w_i \times s_t(i)$$

where $cl.w_i$ is the weight w_i of cl. The values of $P(s_t, a)$ form the *prediction array*. Next, XCSF selects an action to perform. Note that in XCSF only one dummy action is present which has no effect. The classifiers in [M] that advocate the selected action are put in the current *action set* [A]; the selected (dummy) action is sent to the environment and a reward P is returned to the system.

Reinforcement Component. XCSF uses the incoming reward P to update the parameters of classifiers in action set [A]. The weight vector w of the classifiers in [A] is updated using a *modified delta rule* [12]. For each classifier $cl \in [A]$, each weight $cl.w_i$ is adjusted by a quantity Δw_i computed as:

$$\Delta w_i = \frac{\eta}{|s_t(i)|^2}(P - cl.p(s_t))s_t(i) \quad (2)$$

where η is the correction rate and $|s_t|^2$ is the norm the input vector s_t, (see [15] for details). The values Δw_i are used to update the weights of classifier cl as:

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (3)$$

Then the prediction error ε is updated as:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p(s_t)| - cl.\varepsilon)$$

Finally, classifier fitness is updated as in XCS.

Discovery Component. The genetic algorithm in XCSF works as in XCSI [14]. On a regular basis depending on the parameter θ_{ga}, the genetic algorithm is applied to classifiers in [A]. It selects two classifiers with probability *proportional to their fitness*, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. Crossover and mutation work as in XCSI [14, 15]. The resulting offspring are inserted into the population and two classifiers are deleted to keep the population size constant.

3. LINEAR LEAST SQUARE FOR XCSF

Recently, [10] analyzed generalization in XCSF and showed that the Widrow-Hoff update used in XCSF can converge very slowly when some conditions on the distribution of the classifier inputs hold. Therefore, XCSF may be unable to fully exploit its generalization capabilities and evolve piecewise constant approximations (like those evolved by XCSI [14]) instead of the expected piecewise linear approximations [15]. To limit the influence of the inputs distribution on XCSF and to make generalization more effective, [10] replaced Widrow-Hoff update with linear least squares update. Experimental results reported in [10] show that linear least squares significantly improves the generalization capabilities of XCSF and its robustness. In the following, we briefly describe linear least squares in XCSF and refer the reader to [10] for details and discussions.

Linear least squares is a well-known alternative to Widrow-Hoff update, that has been already used in reinforcement learning [2, 1]. Linear least squares is more efficient that Widrow-Hoff since it extracts more information from each sample and it would be expected to converge with fewer samples [6]. The Widrow-Hoff update used in the original XCSF [15] can be viewed as a gradient descent aimed at minimizing the following error function:

$$\xi(\vec{w}) = \tfrac{1}{2}e^2(t) \quad \text{where } e(t) = f(x_t) - cl.p(x_t) \quad (4)$$

linear least squares [7] replaces this error function with,

$$\xi(\vec{w}) = \tfrac{1}{2}\sum_{i=1}^{t} e^2(i) \quad \text{where } e(i) = f(x_i) - cl.p(x_i). \quad (5)$$

Let $X_t = [\vec{x_1}, \vec{x_2}, \ldots, \vec{x_t}]^T$ be the vector of inputs until time t, and let $\vec{f}(t) = [f(\vec{x_1}), f(\vec{x_2}), \ldots, f(\vec{x_t})]$ be the vector of all the desired outputs. The weight vector $\vec{w}$ that minimizes the error function in Equation 5, is computed as the solution of the following equation:

$$(X_t^T X_t)\vec{w} = X_t^T \vec{f}(t). \quad (6)$$

The weight vector $\vec{w}$ can be either determined by computing the *pseudoinverse* X^+ of X ($X^+ = (X_t^T X_t)^{-1} X_t^T$), or by applying the Singular Value Decomposition to $(X_t^T X_t)$ [11]. The least squares update requires that all the inputs used for updating are stored. But in general we are interested in an incremental update, for this purpose least squares is implemented by keeping a vector X_t^n of the latest n visited inputs

Algorithm 1 Update classifier cl with linear least squares

```
1: procedure UPDATE_PREDICTION(cl, s, P)
2:     add(cl.X,s);                    ▷ Add the current s to X_t
3:     add(cl.Y,P);                    ▷ Add the current P to Y_t
4:     A ← cl.X^t × cl.X;
5:     b⃗ ← cl.X^t × cl.Y;
6:     w⃗ ← SOLVE(A, b⃗);                ▷ Compute the new w⃗
7:     for i ∈ {0, ..., |s|} do        ▷ Update the classifiers' w⃗
8:         cl.w_i ← (1 − η) cl.w_i + η w_i
9:     end for
10: end procedure
```

and a vector f_t^n of the corresponding desired output [6]. The incremental version of least squares is performed as follows. Given,

$$\begin{aligned} X_t^n &= [\vec{x}_{t-n+1}, \vec{x}_{t-n+2}, \ldots, \vec{x_t}]^T \\ Y_t^n &= [f(\vec{x}_{t-n+1}), f(\vec{x}_{t-n+2}), \ldots, f(\vec{x_t})]^T \end{aligned}$$

The weight vector $\vec{w}$ to update the current classifier weight vector is computed as the solution of the following equation:

$$(X_t^n)^T X_t^n \vec{w} = (X_t^n)^T Y_t^n \tag{7}$$

then, the current weight vector is updated as,

$$\vec{w_t} = (1-\eta)\vec{w}_{t-1} + \eta\vec{w}$$

This formulation of the linear least squares allows us to use a small window, given an adequately small value of the learning rate η.

To add linear least squares to XCSF classifiers are enriched with two new parameters, the vector X the last n inputs, and the vector Y storing the n outputs of the elements in X; both X and Y are FIFO queues of n elements, implemented as circular arrays. The update of the classifiers prediction with the linear least squares is reported as Algorithm 1 using the typical algorithmic notation used for XCS [3]. The procedure SOLVE encapsulate the algorithm for solving the linear system (Equation 7) which in our implementation is based on the Singular Value Decomposition (SVD) [11] as implemented in the GSL/BLAS package [4].

In this paper, all the experiments are performed using XCSF modified with the linear least squares for the weight update discussed here; for this purpose, classifiers keep a vector of the latest 50 visited inputs and of the corresponding outputs (i.e., $n = 50$).

4. BEYOND LINEAR APPROXIMATION

We now extend XCSF to allow polynomial approximations. Given the current input x, in XCSF the classifier prediction is computed as,

$$cl.p(\vec{x}) = w_0 x_0 + w_1 x,$$

where $\vec{x} = \langle x_0, x\rangle$, and x_0 is the usual fixed constant. We can introduce a quadratic term in the approximation evolved by XCSF so that,

$$cl.p(\vec{x}) = w_0 x_0 + w_1 x + w_2 x^2 \tag{8}$$

To learn the new set of weights we use the usual update procedure (e.g., least squares) applied to the input vector $\vec{y}$ obtained by preprocessing the actual input $\vec{x}$, as $\vec{y} = \langle x_0, x, x^2\rangle$.

The same approach can be generalized to allow the approximation of any polynomials. Given a function $f(x)$ of one variable, to use XCSF to evolve an approximation $\hat{f}(x)$ based on a polynomial of order k, we define

$$\vec{y} = \langle x_0, x, x^2, \ldots, x^k\rangle$$

and apply XCSF to the newly defined input space. When more variables are involved we have two possibilities. Either we can use XCSF to evolve a solution built as a sum of different polynomials, one for each variable. In this case, given n input variables $(x_1, \ldots, x_n)$, so that $\vec{x} = \langle x_0, x_1, \ldots, x_n\rangle$, we define

$$\vec{y} = \langle x_0, x_1, x_1^2, \ldots, x_1^k, \ldots, x_n, x_n^2, \ldots, x_n^k\rangle$$

Either, we can use the *multivariate polynomial* in n variables and define the new input $\vec{y}$ accordingly. In this case with two input variables, we would have

$$\vec{y} = \langle x_0, x_1, x_2, x_1x_2, \ldots, x_1^k x_2^k\rangle.$$

When using this approach we obtain approximations in which also mixed terms, involving multiplication of different variables, appear. Nevertheless, as a major drawback, with this approach the size of $\vec{y}$ dramatically grows as the number of actual variables in $\vec{x}$ increases.

4.1 Preprocessing vs Processing

To extend XCSF with quadratic approximation we added a preprocessing step so to enrich the classifier inputs with powers terms. Then we used the usual update introduced for linear prediction to evolve the coefficients of the polynomial. There is an alternative approach that we could have taken. Viewing classifiers in XCSF as "sort of" perceptrons with a linear activation function, we could as well change such an activation function to be quadratic [6]. Accordingly, we may define classifier prediction as,

$$cl.p(\vec{x}) = (\vec{w}\vec{x})^2$$

which, in the case of one input, becomes,

$$cl.p(\vec{x}) = x_0^2 w_0^2 + 2w_0 w_1 x_0 x + w_1^2 x^2.$$

In this case, classifier prediction needs less parameters, two instead of three in Equation 8, but classifier prediction is not linear with respect to the weights. To have a rough idea, consider the gradient descent in the case of a problem with one input. In this case, the gradient for the two weights is

$$[\frac{\partial cl.p(\vec{x})}{\partial w_0}, \frac{\partial cl.p(\vec{x})}{\partial w_1}] = [2(x_0w_0 + xw_1)x_0, 2(x_0w_0 + xw_1)x]$$

which is non linear. In addition, we need an update specific for the new prediction computation which on the other hand makes it more difficult to control the convergence ratio [6]. Accordingly, in this paper we follow the former approach and we implement quadratic approximations through the preprocessing of classifier inputs.

5. DESIGN OF EXPERIMENTS

All the experiments discussed in this paper involve single step problems and are performed following the standard design used in the literature [13, 15]. In each experiment XCSF has to learn to approximate a target function $f(x)$; each experiment consists of a number of problems that XCSF must

$$f_p(x) = 1 + x + x^2 + x^3 \quad (9)$$
$$f_{s3}(x) = \sin(x) + \sin(2x) + \sin(3x) \quad (10)$$
$$f_{s4}(x) = \sin(x) + \sin(2x) + \sin(3x) + sin(4x) \quad (11)$$
$$f_{abs}(x) = |\sin(x) + |\cos(x)|| \quad (12)$$

(a)

$$F_p(x) = 1000 \times f_p(\frac{x}{1000}), x \in [-1000, 1000] \quad (13)$$
$$F_{s3}(x) = 1000 \times f_{s3}(\frac{2\pi x}{1000}), x \in [0, 1000] \quad (14)$$
$$F_{s4}(x) = 1000 \times f_{s4}(\frac{2\pi x}{1000}), x \in [0, 1000] \quad (15)$$
$$F_{abs}(x) = 1000 \times f_{abs}(\frac{2\pi x}{1000}), x \in [0, 1000] \quad (16)$$

(b)

Table 1: Functions used to test XCSF: (a) original functions; (b) functions adapted to integers.

solve. For each problem, an example $\langle x, f(x)\rangle$ of the target function $f(x)$ is randomly selected; x is input to XCSF whom computes the approximated value $\hat{f}(x)$ as the expected payoff of the only available dummy action; the action is virtually performed (the action has no actual effect), and XCSF receives a reward equal to $f(x)$. XCSF learns to approximate the target function $f(x)$ by evolving a mapping from the inputs to the payoff of the only available action. Each problem is either a *learning* problem or a *test* problem. In *learning* problems, the genetic algorithm is enabled while it is turned off during *test* problems. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate. XCSF performance is measured as the accuracy of the evolved approximation $\hat{f}(x)$ with respect to the target function $f(x)$. To evaluate the evolved approximation $\hat{f}(x)$ we measure the *mean absolute error* (MAE) defined as:

$$MAE = \frac{1}{n}\sum_x |f(x) - \hat{f}(x)|,$$

where n is the number of points for which $f(x)$ is defined. In particular we use the average MAE over the performed experiments, dubbed $\overline{MAE}$. All the statistics reported in this paper are averaged over 50 experiments.

Note that, all the experiments discussed in this paper are performed using XCSF modified with the linear least squares; for this purpose, in all the experiments, classifiers maintain a vector of the latest 50 visited inputs and of the corresponding outputs, i.e., the parameter n for linear least squares is set to 50.

6. EXPERIMENTAL RESULTS

We now compare XCSF with linear prediction to the version of XCSF extended with quadratic and cubic predictions. For this purpose, we have consider problems taken from the literature [8, 9, 16] and adapted them to integers, following the approach of [15] for the sine function. The functions we use are reported in Table 1a: Equation 9 is from [8], Equation 10 and Equation 11 are the Koza's Sinus Three and Sinus Four [9], finally Equation 12 is from [16]. Table 1b reports the functions in Table 1a adapted to integers that we use in this section to test XCSF with quadratic and cubic approximations.

For the experiments discussed here we always use the same parameter settings: $N = 1000$; $\beta = 0.2$; $\alpha = 0.1$; $\nu = 5$; $\chi = 0.8$, $\mu = 0.04$, $\theta_{nma} = 1$, $\theta_{del} = 50$; $\theta_{GA} = 50$; $\delta = 0.1$; GA-subsumption is on with $\theta_{sub} = 50$; while action-set subsumption is off; the parameters for integer conditions are $m_0 = 200$, $r_0 = 100$, and $x_0 = 1000$ [14]; the learning rate for weight update η is 0.2; each experiment consists of 50000 learning problems; as in [10], for the linear square update we use a window of the last 50 input output pairs; for each problem we use different values of the error threshold ϵ_0.

6.1 Quadratic Approximation

First, we compare XCSF with linear and quadratic prediction on the functions reported in Table 1b for different values of the error threshold ϵ_0. Table 2 reports, for each experiment (i) the error threshold ϵ_0 used, and for each version of XCSF, (ii) the average mean absolute error with the standard deviation (column $\overline{MAE}\pm\sigma$), (iii) the average number of macroclassifiers in the final populations with the standard deviation (column $|[P]| \pm \sigma$), and (iv) the generality of classifiers in the final populations with the standard deviation (column G([P])$\pm\sigma$), computed as the average number of inputs that classifiers in the populations match.

As can be noted, when using quadratic approximation for classifier prediction, XCSF evolves more accurate solutions, in fact, the average error is usually (though not always) smaller with quadratic prediction. In addition, when using linear approximation in the sine three and four, XCSF reaches an average absolute error ($\overline{MAE}$) that is higher than the error threshold ϵ_0. In contrast, XCSF with quadratic approximation reaches an average error below the target error threshold ϵ_0 on both problems. Most interesting the evolved solutions are generally smaller when quadratic approximation is used, in fact, the average size of the evolved populations with quadratic approximations are generally smaller than those evolved with linear approximations. This of course happens because the classifiers evolved with quadratic approximation are more general, as showed by the values of the average classifiers' generality (column $G([P]) \pm \sigma$ in Table 2 and Table 3).

In Figure 1, we report the performance of XCSF for the sine three $F_{s3}(x)$ with ϵ_0=50 with linear (Figure 1a) and quadratic prediction (Figure 1b), compared against the target function; vertical bars in the plots indicate the variance computed over the 50 runs. As can be noted, on the average, XCSF with quadratic prediction evolves solutions that are more accurate along all the problem range, in fact the error bars have almost the same size over all the plot (Figure 1b). When using linear prediction, XCSF is still rather accurate though the accuracy slightly drops around the smooth sections of the target function.

As an example, we also report the most inaccurate populations (corresponding to the highest values of $\overline{MAE}$) evolved with linear and quadratic prediction in Figure 2a and Figure 2b respectively. In the linear case (Figure 2a) each classifier in the final population is represented by the segment it approximates; in the quadratic case (Figure 2b) each classifier in the final population is represented by the section of the curve it approximate; note that in both figures no informa-

$f(x)$	N	ϵ_0	Linear			Quadratic		
			$\overline{MAE} \pm \sigma$	$\|[P]\| \pm \sigma$	G([P])$\pm\sigma$	$\overline{MAE} \pm \sigma$	$\|[P]\| \pm \sigma$	G([P])$\pm\sigma$
$F_p(x)$	1000	5	2.56 ± 0.28	65.68 ± 8.55	198.67 ± 126.70	2.00 ± 0.26	58.04 ± 9.52	502.20 ± 129.72
$F_p(x)$	1000	50	24.39 ± 3.29	64.78 ± 11.07	585.46 ± 310.89	21.15 ± 3.98	57.38 ± 13.91	811.37 ± 433.47
$F_p(x)$	1000	100	42.08 ± 7.79	62.14 ± 12.14	722.87 ± 346.67	53.33 ± 10.47	47.70 ± 9.05	928.30 ± 666.92
$F_{s3}(x)$	1000	5	11.86 ± 8.03	71.16 ± 7.51	30.08 ± 22.84	2.93 ± 0.73	46.06 ± 7.01	71.66 ± 27.13
$F_{s3}(x)$	1000	50	33.77 ± 7.65	48.06 ± 6.12	82.19 ± 38.86	24.31 ± 3.18	46.70 ± 7.00	176.48 ± 31.72
$F_{s3}(x)$	1000	500	292.31 ± 24.75	64.24 ± 9.69	285.17 ± 219.38	282.45 ± 29.60	57.82 ± 10.95	405.39 ± 252.26
$F_{s4}(x)$	1000	5	28.67 ± 45.55	78.44 ± 6.19	27.34 ± 21.89	5.02 ± 8.21	48.18 ± 5.95	54.13 ± 23.74
$F_{s4}(x)$	1000	50	37.83 ± 12.78	49.66 ± 6.45	63.34 ± 31.53	26.72 ± 8.75	44.80 ± 7.74	130.83 ± 28.16
$F_{s4}(x)$	1000	500	311.76 ± 18.33	46.72 ± 8.40	253.02 ± 234.78	280.81 ± 24.84	53.20 ± 11.06	332.83 ± 233.75
$F_{abs}(x)$	1000	5	4.67 ± 6.76	52.9 ± 5.22	54.17 ± 23.09	1.92 ± 0.22	35.96 ± 5.78	149.31 ± 73.57
$F_{abs}(x)$	1000	50	21.21 ± 2.22	46.8 ± 5.47	139.50 ± 30.44	17.00 ± 3.07	47.22 ± 6.57	207.70 ± 96.70
$F_{abs}(x)$	1000	100	44.03 ± 3.84	54 ± 8.28	182.24 ± 63.66	37.80 ± 4.98	53.84 ± 10.45	255.82 ± 81.61

Table 2: XCSF with linear and quadratic approximation: $f(x)$ is the target function; N is the population size; ϵ_0 is the error threshold. Statistics are averages over 50 runs.

tion regarding the classifier fitness or numerosity is reported, thus it is difficult to understand the contribution of each classifier to the overall solution that the two populations represent. Noticeably, XCSF can successfully exploit quadratic approximation to produce solutions which smoothly approximate large sections of $F_{s4}(x)$ whereas XCSF linear approximation tend to be rather inaccurate around the smooth function peaks. When the error threshold is smaller the difference in approximation between linear and quadratic approximation is more evident.

As a second example, we report in Figure 3 the performance of XCSF for the sine four $F_{s4}(x)$ with ϵ_0=5 with linear (Figure 3a) and quadratic prediction (Figure 3b), compared against the target function; we also report an example of population evolved with linear and quadratic prediction in Figure 4a and Figure 4b. When the error is smaller, ϵ_0, is more difficult for XCSF with linear approximation to evolve an accurate solution, accordingly the variance around the peaks in Figure 3a is larger than for the quadratic case (Figure 3b). Note however, even with such a small error threshold XCSF with linear approximation is still capable of evolving very accurate solutions such as the one depicted in Figure 4a. On the other hand, quadratic approximation allows XCSF to evolve an even smoother approximation (Figure 4b).

6.2 Cubic Approximation

The approach we applied to extend XCSF for quadratic approximation (discussed in Section 4) is rather general and in principle it can be used to extend XCSF further, beyond quadratic approximations. For instance, we can use it to extend XCSF with cubic approximations to compute classifier prediction. Nevertheless, we shall consider how much advantage in terms of approximation accuracy and generalization such an extension might provide with respect to the overhead introduced. In fact, the additional power terms added to classifiers inputs must be stored in a vector with linear least squares [10], or used to compute additional data structure with the recursive least squares [10], since Widrow-Hoff update would be too slow [6]. Most important, additional power terms introduce bias into classifier weights since classifier prediction error (and thus classifier fitness) will be influenced more by the weights corresponding to high power terms. As an example, consider a classifier that matches

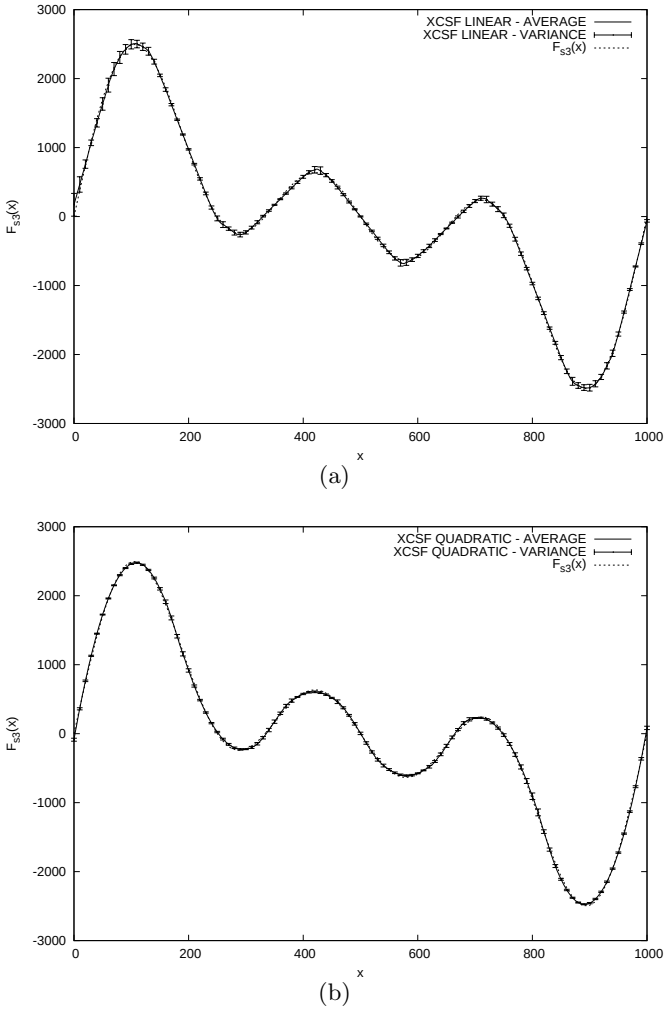
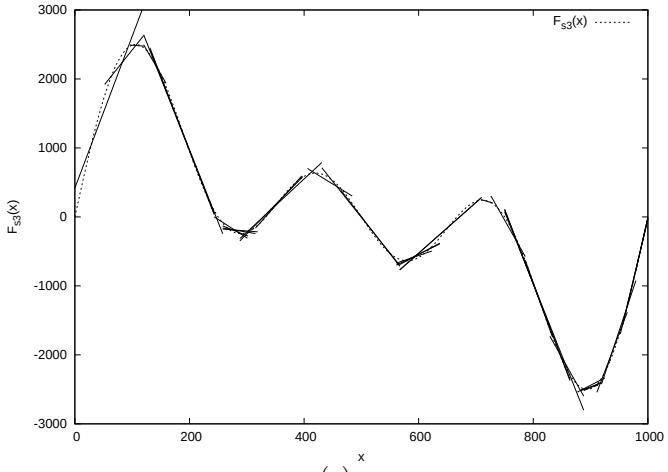
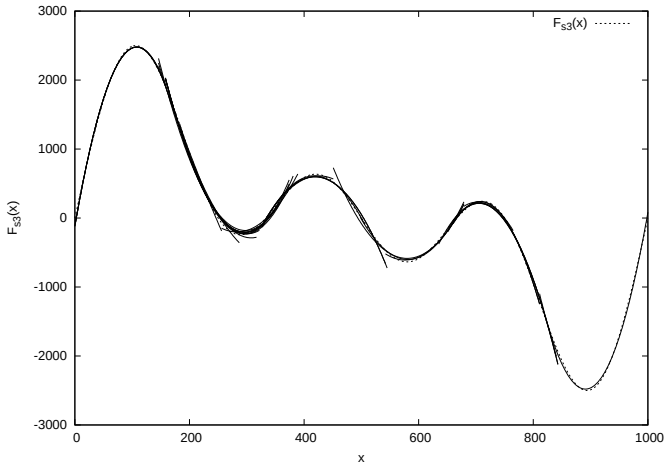


Figure 1: Average performance of XCSF with (a) linear and (b) quadratic approximation applied to $F_{s3}(x)$ with $N = 1000$ and $\epsilon_0 = 50$. Curves are averages over 50 runs.

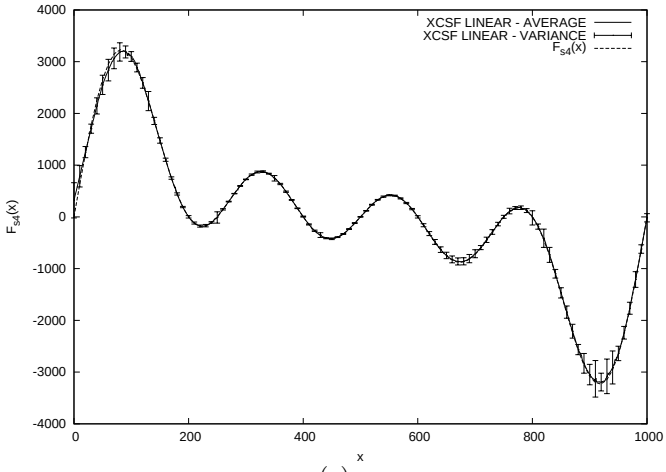


(a)

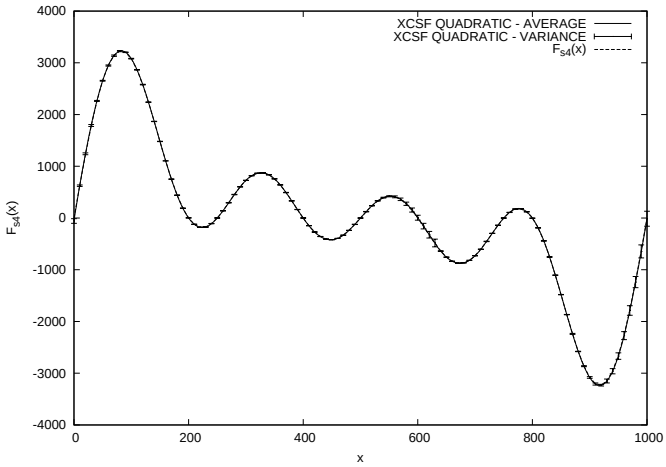


(b)

Figure 2: Worst populations evolved by XCSF for $F_{s3}(x)$ with $N = 1000$ and $\epsilon_0 = 50$: (a) linear prediction; (b) quadratic prediction.



(a)



(b)

Figure 3: Average performance of XCSF with (a) linear and (b) quadratic approximation applied to $F_{s4}(x)$ with $N = 1000$ and $\epsilon_0 = 5$. Curves are averages over 50 runs.

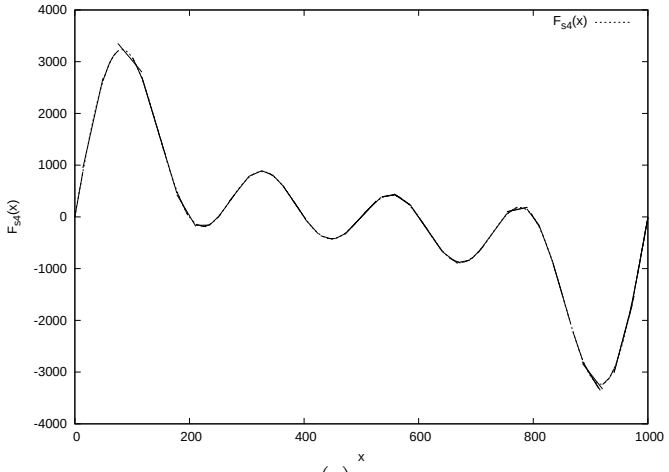
the input value 1000, with a cubic approximation, such an input will become $\langle 10^3, 10^6, 10^9 \rangle$ making weight w_3 (corresponding to input 10^9) way more relevant that weight w_1 (corresponding to input 10^3).

When we apply XCSF with cubic approximation to the problems previously considered with the same settings we obtain the results reported in Table 3. The solutions evolved by the version of XCSF with cubic approximation are (i) generally more accurate than those evolved with quadratic approximation (column $\overline{MAE} \pm \sigma$), and they are also (ii) generally smaller in that they contain less classifiers (column $|[P]| \pm \sigma$), which are more general in that on the average they match more inputs (column $G([P]) \pm \sigma$). Figure 5 shows the average approximation evolved by XCSF with linear (Figure 5a) and cubic (Figure 5b) approximations. Again, XCSF can exploit the available (cubic) approximation to evolve an extremely accurate solution.

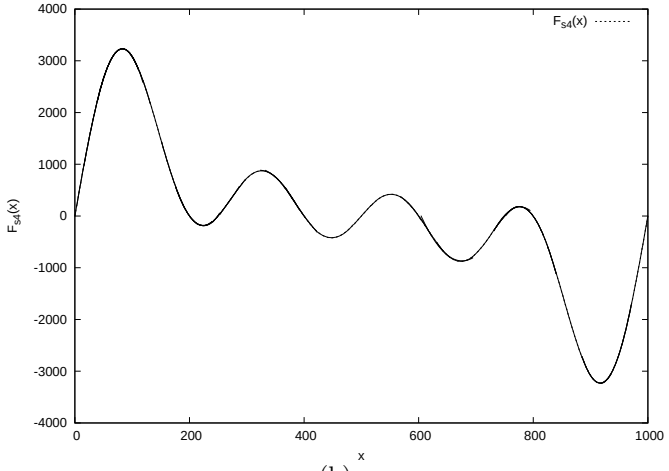
7. STATISTICAL ANALYSIS

We apply a *one-way* analysis of variance or ANOVA [5] to test whether the differences in approximation accuracy and generalization capability observed for XCSF with linear, quadratic, and cubic approximation (Table 2, Table 3) are statistically significant. We also apply typical post-hoc procedures (Tukey, Scheffé, SNK, and Bonferroni) to analyze the differences among different versions of XCSF. We apply ANOVA twice, one over the data collected for small values of ϵ_0 (i.e., 5 and 50), one over all the data.

Analysis for ϵ_0=5 and ϵ_0=50. The ANOVA test performed on the data of mean absolute error shows that some versions of XCSF perform significantly different, with a confidence level of the 99.99%. All the subsequent *post-hoc* procedures show that the mean absolute error obtained by linear approximation is significantly different from all the other two versions; the same holds for XCSF with quadratic and cubic approximations. When we apply the same analysis to the data of the population size (column $|[P]|$ in Table 2 and Table 3) and to the data of classifiers generality (column $G([P])$ in Table 2 and Table 3) we obtain similar results, that is, the difference observed for all the three versions of XCSF is statically significant.



(a)



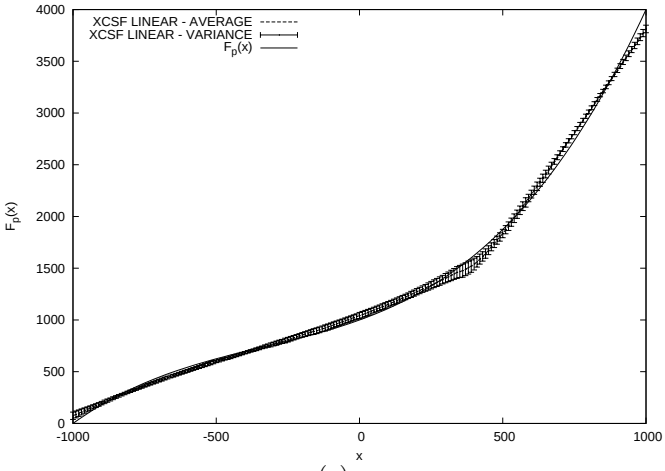
(b)

Figure 4: Example of accurate approximations evolved by XCSF for $F_{s4}(x)$ with $N = 1000$ and $\epsilon_0 = 5$: (a) linear prediction; (b) quadratic prediction.

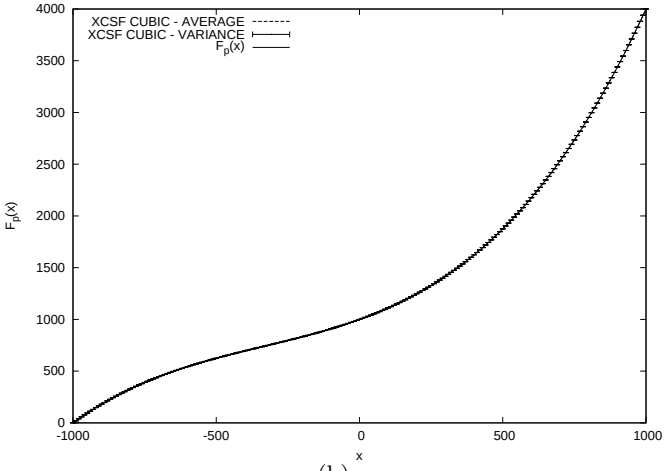
Analysis on all the data. When we consider also higher values of error threshold (ϵ_0=100 and ϵ_0=500) the difference among the different versions becomes more blurred. As before, ANOVA shows a significant difference both in performance (with a confidence level of 99.8%), in the size of the final populations, $|[P]|$ (with a confidence level of 99.99%), and in the classifiers' average generality, $G([P])$ (with a confidence level of 99.99%). The subsequent post-hoc procedures (Scheffé, Tukey, and SNK) for the mean absolute error report two homogeneity groups: one including XCSF with linear and quadratic approximation, one including XCSF with quadratic and cubic approximation. This suggests that when we consider higher error thresholds the performance of linear and quadratic approximation (in terms of approximation accuracy) is similar; the same holds for quadratic and cubic approximation. Therefore, with high error thresholds is not convenient to improve the approximation class. With respect to the size of the final solutions evolved, the ANOVA test shows that the results for the three versions are significantly different with a confidence level of 99.99%; the following post-hoc procedures show that all the three versions are

			Cubic		
$f(x)$	N	ϵ_0	$MAE \pm \sigma$	$\|[P]\| \pm \sigma$	G([P])$\pm\sigma$
$F_p(x)$	1000	5	1.11 ± 7.76	25.90 ± 5.33	1474.29 ± 800.40
$F_p(x)$	1000	50	0.000 ± 0.000	27.92 ± 7.15	1438.17 ± 814.89
$F_p(x)$	1000	100	0.00 ± 0.03	27.38 ± 7.17	1426.41 ± 819.49
$F_{s3}(x)$	1000	5	2.62 ± 0.70	46.06 ± 6.81	122.82 ± 46.31
$F_{s3}(x)$	1000	50	18.02 ± 3.21	56.42 ± 7.81	210.66 ± 54.13
$F_{s3}(x)$	1000	500	250.05 ± 37.026	54.94 ± 8.49	423.99 ± 260.51
$F_{s4}(x)$	1000	5	2.98 ± 0.88	45.24 ± 5.49	91.72 ± 36.91
$F_{s4}(x)$	1000	50	29.78 ± 76.94	53.12 ± 6.28	164.40 ± 41.42
$F_{s4}(x)$	1000	500	247.2 ± 25.67	56.56 ± 9.05	384.58 ± 223.83
$F_{abs}(x)$	1000	5	1.54 ± 0.39	34.16 ± 6.01	163.49 ± 85.03
$F_{abs}(x)$	1000	50	20.01 ± 3.02	53.34 ± 8.94	238.96 ± 76.40
$F_{abs}(x)$	1000	100	42.82 ± 7.11	59.24 ± 9.22	294.40 ± 107.21

Table 3: XCSF with cubic approximation. Statistics are averages over 50 runs.



(a)



(b)

Figure 5: Average performance of XCSF with (a) linear and (b) cubic approximation applied to $F_p(x)$ with $N = 1000$ and $\epsilon_0 = 100$. Curves are averages over 50 runs.

significantly different one from another. Finally, the same tests applied to the data from classifiers' generality show that the three versions of XCSF (linear, quadratic, and cubic) performs significantly different (at the 99.99%) though the post-hoc procedures report different results: Scheffé and Bonferroni show that the difference in classifiers' generality of the three systems is significantly different, in contrast, Tukey tests report no difference among the three systems. The difference in the results of the statistical tests for classifiers' generality is mainly due to the data distribution. While the average absolute error and the population size are basically averages and therefore they are more or less normally distributed, values of classifiers' generality have a distribution that is completely problem dependent. In our experience, the distribution of classifiers' generality has usually one peak for each section of the problem space that allows large generalizations (e.g., in the sine three, the distribution of classifiers' generality one peak for each one large slope).

8. SUMMARY

We have shown how XCSF can be easily extended beyond linear approximation so as to allow polynomial approximations for computing classifier predictions. We have compared the original XCSF with linear prediction, to the versions of XCSF extended with quadratic and cubic prediction on different problems taken from the literature. The results we report show that the generalization mechanism of XCSF can effectively exploit quadratic and cubic approximations to evolve solutions that are usually more accurate and more compact than those evolved by the original XCSF with linear approximation. The statistical analysis of the results shows that such improvements are statistically significant, both in accuracy and in generalization capability. Finally, we note that the approach we discussed here is generalizable to higher order polynomials. However, our statistical analysis suggests that increasing the degree of the polynomials used to approximate classifier prediction might not lead to statistically significant improvements unless small error thresholds are involved.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF (F49620-03-1-0129), and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for SupercomputingApplications (NCSA) and funded by the Office of Naval Research (N00014-01-1-0175). The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

9. REFERENCES

[1] J. A. Boyan. Least-squares temporal difference learning. In *Proc. 16th International Conf. on Machine Learning*, pages 49–56. Morgan Kaufmann, San Francisco, CA, 1999.

[2] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

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

[4] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual – Second Edition*. Network Theory Ltd., 2003. (paperback).

[5] S. A. Glantz and B. K. Slinker. *Primer of Applied Regression & Analysis of Variance*. McGraw Hill, 2001. second edition.

[6] S. Haykin. *Adaptive Filter Theory*. Prentice-Hall, 1986.

[7] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[8] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[9] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, May 1994.

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

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, editors. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, Feb. 2002.

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

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

[14] S. W. Wilson. Mining Oblique Data with XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems. Third International Workshop, IWLCS 2000, Paris, France, September 15-16*, volume 1996 of *Lecture notes in Computer Science*, pages 158–174. Springer-Verlag, Apr. 2001.

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

[16] I. Zelinka. Analytic programming by means of soma algorithm. In *Proceedings of the 8th International Conference on Soft Computing, Mendel'02*, pages 93–101, Brno,Czech Republic, 2002.