Evolving Prediction Weights Using Evolution Strategy

Trung Hau Tran, Cédric Sanza, Yves Duthen
IRIT-UPS-CNRS, University of Toulouse, Toulouse, France
{hau, sanza}@irit.fr

ABSTRACT
The evolution strategy is one of the strongest evolutionary algorithms for optimizing real-value vectors. In this paper, we study how to use it for the evolution of prediction weights in XCSF in order to make the computed prediction more accurate. Our version of XCSF shows to be able to evolve more accurate linear approximations of functions. It is more efficient than the original XCSF and slightly better than XCSF with recursive least squares, in spite of its simple structure and its low complexity.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning, Parameter learning*

General Terms
Algorithms, Performance

Keywords
XCSF, function approximation, evolution strategy

1. INTRODUCTION
The learning classifier system XCSF [15][16] is an extension to XCS [13]. The major development of XCSF is the use of a computed classifier prediction instead of a fixed scalar one, which allows the system to evolve classifiers with more accurate values of the prediction. Thus, XCSF improves its performance and can yield more compact solutions.

The efficiency of XCSF in solving reinforcement learning problems has been demonstrated [5]. In addition, XCSF can be used as an approach to approximate a target function [4][15][16]. In this case, it has only one action (or dummy action) and the prediction represents the value of this approximated function. Wilson [15][16] proposed a linear prediction, in which the classifier prediction $p(x,w)$ is computed as a linear combination of the input x and a vector of prediction weights w associated to each classifier. Lanzi [4] extended XCSF beyond linear prediction to polynomial predictions. The accuracy and generalization in the modified version of XCSF is significantly better than the original XCSF. XCSF with neural prediction [7] is another extension to XCSF, where the prediction is the output of a multi-layer neural network. A key importance of XCSF with neural prediction is that the structure of the network is also evolved, along with the classifier condition, by a genetic algorithm to improve the approximation accuracy. This new XCSF version showed to be more powerful than XCSF on functions highly non linear.

Besides the type of function approximators used to compute the prediction, the performance of XCSF also depends on the type of algorithms used to update prediction weights. In [15][16], prediction weights w are updated using a *modified delta rule* [12]. In [6], different update algorithms were proposed (*recursive least square* (RLS), *Kalman filter* and *gain adaptation*). XCSF with RLS and Kalman filter achieve similar performance and outperform the other versions.

In our work, we will focus on prediction weight update algorithms to extend XSCF performance. Indeed, we will investigate an approach in which prediction weights are evolved by using an evolutionary algorithm. An evolution strategy [9] is more preferable than a genetic algorithm [2] for this task because the former maintains a small population of individuals (i.e. prediction weight vectors) and can learn fast. Its efficiency has been demonstrated to solve the frog problem (one-dimensional problem) in [10] and double integrator and pendulum problems (two-dimensional problems) in [11]. The system that we used, called XCSFCA, evolved action weights to make the computed continuous action more accurate in the absence of action feedback. In this paper, we apply our version of XCSF with evolution strategy, called XCSFES, to approximate a set of continuous functions taken from [6]. The main difference with XCSFCA is that XCSFES evolves prediction weights while the action part is not involved. The results show that XCSFES outperforms the original XCSF and obtains more accurate approximations than XCSF with RLS on the functions. XCSFES produces more compact solutions than these versions.

The next section gives a brief description of the evolution strategy. Section 3 summarizes the functioning of XCSF. Section 4 presents XCSFES. Section 5 presents the results obtained when applying it to approximate the continuous functions. A discussion in section 6 concludes this paper.

2. EVOLUTION STRATEGY
The evolution strategy (ES) was developed by Rechenberg [8] and Schwefel [9] for optimizing real-vectors. It is based on the ideas of evolution and adaptation. Each individual in the ES is a real vector of object parameters $x^{(g)} \in \Re^n$. The simplest version of the ES is a (1+1)-ES. In this elitist version, in every generation one parent $x^{(g)}$ produces one offspring $x'^{(g+1)}$ by mutation, the offspring is assigned a fitness value evaluating its quality with respect to the problem, the offspring competes with the parent based on the fitness, and the better individual becomes the parent of the next generation. Thus, the selection of individuals for the next generation g+1 is deterministic. Note that, it is a minimization problem and thus the direction in Eq.(1) is "≤".

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

$$x^{(g+1)} = \begin{cases} x'^{(g+1)} & \text{if } fitness(x'^{(g+1)}) \le fitness(x^{(g)}) \\ x^{(g)} & \text{otherwise} \end{cases} \quad (1)$$

Mutation on $x^{(g)}$ is performed by adding a random value $N(0,\sigma^{(g)2})$ from a normal distribution with mean of zero and standard deviation of $\sigma^{(g)}$ (Eq.(2)). The global step-size (or standard deviation) $\sigma^{(g)} \in \Re^+$ used by mutation is itself adapted by using a *1/5th-rule* [8] based on the rate of successful mutations over a number of G generations. A mutation is called successful if the offspring dominates its parent in the fitness sense (i.e. $fitness(x'^{(g+1)}) \le fitness(x^{(g)})$). [3] proposed an alternative simple implementation of the 1/5th-rule. Indeed, after the evaluation of a new generation, the success or the failure of the mutation is directly used in the adaptation of the global step-size (Eq.(3)). The new implementation can achieve faster and more precisely performance than the original ES.

$$x'^{(g+1)} = x^{(g)} + \sigma^{(g)} \cdot N(0,1) \quad (2)$$

$$\sigma^{(g+1)} = \begin{cases} \sigma^{(g)} \cdot \alpha_{ES} & \text{if } fitness(x'^{(g+1)}) \le fitness(x^{(g)}) \\ \sigma^{(g)} \cdot \alpha_{ES}^{-1/4} & \text{otherwise} \end{cases} \quad (3)$$

Where $N(0,1)$ is a normal distribution with mean of zero and standard deviation of one. α_{ES} is the change rate of the global step-size. Values for α_{ES} are recommended between $2^{1/n}$ and 2, where n is the dimension of the problem. Each element $x_i^{(g)}$ is initialized to a value $x_i^{(0)}$. $\sigma^{(g)}$ is initialized to a constant value $\sigma^{(0)}$. The value for this constant is problem dependent.

3. REVIEW OF XCSF

XCSF is an extension to XCS for learning environments where the input is continuous and the payoff landscape is continuous with respect to the input. A new classifier condition structure adapting to real value inputs is introduced that allows XCSF to evolve more appropriate conditions. The use of a computed prediction as a function of the input allows XCSF to evolve classifiers with more accurate values of the prediction. Thus, the system performance is improved, XCSF can yield more compact solutions and the convergence time may increase.

XCSF runs like XCS. The system receives an input message x via the detectors, forms a match set M of classifiers whose condition part matches x, determines a winner action a^* between possible actions in M according to an action selection strategy, forms an action set A of the classifiers advocating a^* and sends a^* to the effectors. The system receives a reward r at the next step to update the parameters of the classifiers in A. A new component updates the prediction weights in order to make the predictions more accurate. Covering operator is triggered according to a covering strategy. Either no existing classifier matches x as in [13] or M contains less than ϕ_{mna} discrete actions as in [1]. A niche genetic algorithm is invoked to generate potential classifiers adapting to the problem.

Classifier condition. XCSF extends XCS to deal with continuous input. The components of a classifier condition are the interval predicates $int_i=(l_i,u_i)$ where l_i("lower") and u_i("upper") are real values. A classifier cl matches an input message $x=(x_1,...,x_n)$ if each element x_i belongs to the corresponding interval predicate $cl.int_i$ at the position i i.e. $l_i \le x_i \le u_i$. The system with the real input was studied in [14] and one can use again the methods manipulating interval predicates.

Computed classifier prediction. The new concept in XCSF is the computation of the classifier prediction instead of using a parameter estimating its value. A vector of weights $w=(w_0,w_1,...,w_n)$, called the prediction weight vector, is added to each classifier. The classifier prediction $cl.p$ is computed as a linear combination of the prediction weight vector $cl.w$ and the input message $x=(x_1,...,x_n)$ concatenated with a constant x_0. Wilson's studies [16] proposed that the value of x_0 should be the same order of the values of the elements of x.

$$\begin{aligned} cl.p(x) &= w \cdot x' \\ w &= (w_0, w_1, ..., w_n) \\ x' &= (x_0, x_1, ..., x_n) \end{aligned} \quad (4)$$

Covering operator. When a new covered classifier is created, each interval predicate $int_i=(l_i,u_i)$ is generated as $l_i = x_i - rand(r_0)$ and $u_i = x_i + rand(r_0)$, where $rand(r_0)$ is a value uniform randomly from $[0,r_0]$ and r_0 is a real constant, and each element w_i of the weight vector is initialized to a value uniform randomly from [-1,1].

Discovery mechanism. A genetic algorithm works as in XCS. Crossover (with probability χ) exchanges alleles of two parents between two crossover points. Since an allele is a real value, a new mutation operator is introduced. Mutation (with probability μ) modifies an allele by adding an amount $\pm rand(m_0)$ where m_0 is a real constant.

Domain control. Covering and mutation operators can generate illegal intervals. If it happens, l_i and u_i of an interval predicate are brought back to the condition bounds and they are possibly permuted in order to respect the predicate constraint $l_i \le u_i$.

Reinforcement. The prediction weight vector $cl.w$ of each classifier in A is updated using a *modified delta rule* [12]:

$$\Delta w_i = \left(\frac{\eta}{\|x'\|^2} \right) \cdot \left(P - cl.p(x) \right) \cdot x_i \quad (5)$$

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (6)$$

Where η is the correction rate. The classifier prediction error $cl.\varepsilon$ and the classifier fitness $cl.F$ are then updated as follows:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta \cdot \left(\left| P - cl.p(x) \right| - cl.\varepsilon \right) \quad (7)$$

$$cl.\kappa = \begin{cases} 1 & \text{if } cl.\varepsilon < \varepsilon_0 \\ \alpha \cdot \left(\frac{cl.\varepsilon}{\varepsilon_0} \right)^{-\nu} & \text{otherwise} \end{cases} \quad (8)$$

$$cl.\kappa' = \frac{cl.\kappa \cdot cl.num}{\sum_{j \in [A]} cl_j.\kappa \cdot cl_j.num} \quad (9)$$

$$cl.F \leftarrow cl.F + \beta \cdot \left(cl.\kappa' - cl.F \right) \quad (10)$$

Where β is the learning rate. The constant ε_0 represents the threshold of tolerance for the prediction error $cl.\varepsilon$. If $cl.\varepsilon$ is below ε_0, the error is accepted and the classifier cl is considered to be accurate ($cl.\kappa$=1). Otherwise its accuracy drops off and is controlled by an accuracy function with the parameters α and ν. The relative accuracy $cl.\kappa'$ is calculated with respect to the other classifiers cl_j of

A. *cl.num* is the numerosity of the classifier *cl*. Notice that, *cl.w* is updated first and then *cl.*ε and *cl.F* are updated.

4. XCSF WITH EVOLUTION STRATEGY

The integration of the evolution strategy into XCSF is straightforward. Prediction weights in XCSF are updated by algorithms in Eq.(5)&Eq.(6). We replace them by the corresponding algorithm (Algorithm 1). The vector of object parameters *x* in Eq.(2) corresponds to the prediction weight vector *cl.w* and the function evaluation *fitness*(*x*) of an individual *x* in Eq.(3) corresponds to the calculation of the absolute error between the current reward and the computed prediction. The modified classifier system is called XCSFES.

The classifier structure is slightly extended. Indeed, one global step-size parameter σ is added to each classifier. An (1+1)-ES is applied on each member of action set *A* to evolve prediction weights during explore problems. It will mutate the current prediction weight vector of each action set classifier to produce one mutant vector and evaluate it. The better prediction weight vector will be kept to the next time-step. Consider a classifier *cl* in *A* with its prediction weights *cl.w* and its step-size *cl.*σ, the mutant classifier *cl′* is a copy of the parent classifier *cl* except for its prediction weight vector represented by *cl′.w* generated by mutation on *cl.w* using *cl.*σ. This corresponds to step 1 (Algorithm 1). The evaluation of the mutant prediction weight vector *cl′.w* consists of the absolute error calculation between the current payoff *P* and the computed prediction *cl′.p*(*x*,*cl′.w*). The evaluation of the parent vector is similar to the one of the mutant, i.e. the mutant and the parent are evaluated in the context of the current payoff *P*. That corresponds to step 2 (Algorithm 1). If the mutant is better than the parent (i.e. lower error), it will replace its parent (step 3 in Algorithm 1). If replacement does not occur, the mutant is discarded and a new mutant will be generated in next generations. After the evaluation of a generation, the global step-size is itself updated by taking into account the success or the failure of the mutation (step 4 in Algorithm 1).

Algorithm 1. Evolve prediction weights by (1+1)-ES

1. Mutate one parent prediction weight vector represented by *cl.w* to produce one mutant vector *cl′.w* by Eq.(2)
2. Evaluate the mutant vector *cl′.w*, i.e. compute the absolute error between the reward *P* and the computed prediction *cl′.p*

$$eval(cl'.w) = \left| P - cl'.p\left(x, cl'.w\right) \right|$$

Evaluate the parent vector *cl.w*:

$$eval(cl.w) = \left| P - cl.p\left(x, cl.w\right) \right|$$

3. Replace the parent prediction weight vector if the mutant vector dominates the parent vector

IF (*eval*(*cl′.w*) < *eval*(*cl.w*))
replace cl by cl′

4. Update the global step-size *cl.*σ by Eq.(3)

In term of memory usage XCSFES requires only one additional parameter (i.e. the global step-size). The complexity of XCSFES's update algorithm is *O*(*n*). The global step size may prematurely converge. A lower bound σ_{min} may be necessary to ensure a minimum global step size.

5. EXPERIMENTS

5.1 Experiment Settings

XCSF learns to approximate a function *f*(*x*) from examples (*x*, *f*(*x*)), where *x* is randomly selected on the domain [0,1] and *f*(*x*) is the corresponding function value. The following set of functions is taken from [6]. XCSF receives a vector *x* as input, computes the approximated value $\hat{f}(x)$ as the system prediction for the only available action, executes the action and then receives the real function value *f*(*x*) used as reward.

Table 1. Target functions with $x \in [0,1]$

$$f_p(x) = 1 + x + x^2 + x^3 \tag{11}$$

$$f_{abs}(x) = \left| \sin(x) + \left|\cos(x)\right| \right| \tag{12}$$

$$f_{s3}(x) = \sin(x) + \sin(2x) + \sin(3x) \tag{13}$$

$$f_{s4}(x) = \sin(x) + \sin(2x) + \sin(3x) + \sin(4x) \tag{14}$$

We use the same parameter settings and measures described in [6]. These are: N=800, β=0.2, θ_{GA}=50, χ=0.8, μ=0.04, r_0=0.1, m_0=0.2, x_0=1, α=0.1, ν=5, θ_{del}=50, δ=0.1 and three values 0.05, 0.10 and 0.20 for ε_0. XCSFES uses the extra settings: the initial standard deviation $\sigma^{(0)}$=3, σ_{min}=0.01, the change rate $\alpha_{ES}=2^{1/n}$ [3], where n=2 is the vector length of prediction weights. Each element w_i of the prediction weight vector is initialized to 1. GA-subsumption is enabled with θ_{gasub}=50 while action set subsumption is disabled. Explore and exploit problems are alternated. One run is stopped after 100,000 explore problems. The system error measures the difference between the approximated value and the real function value. The system error curve is plotted by using a 50-point running average from exploit problems, averaged over 50 runs. We use the root mean square error (RMSE) defined in [6] to evaluate the approximation accuracy. $\overline{RMSE}$ represents the average RMSE over all runs.

$$RMSE = \sqrt{\frac{1}{n_{rmse}} \sum_{x} (f(x) - \hat{f}(x))^2} \tag{15}$$

Where n_{rmse} is the number of samples used for this measure. In our experiment, *x* is chosen from 0 to 1, increased by 0.001.

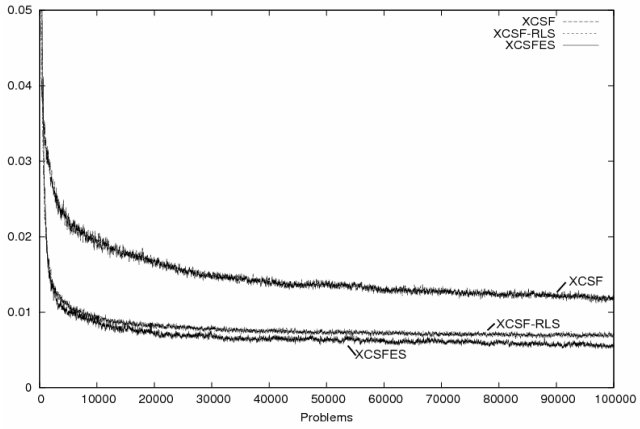
5.2 Results

Table 2 reports the performance of XCSFES, XCSF with RLS [6] and XCSF on the target functions. Over 50 runs, approximations obtained by XCSFES are more accurate than the original XCSF on the functions. In comparison with XCSF with RLS, XCSFES performs slightly better. For the average population size, XCSFES produces more compact solutions than XCSF and XCSF with RLS.

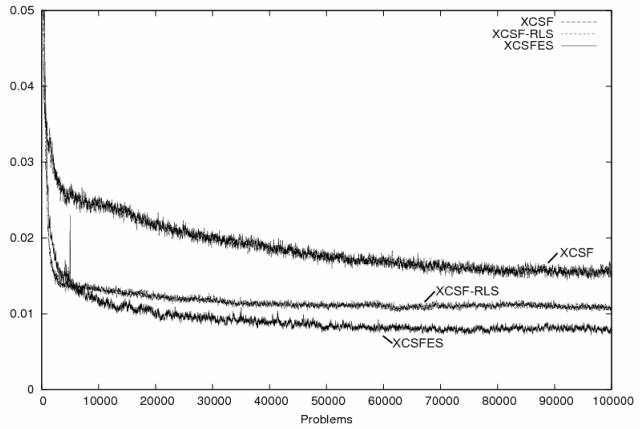
Fig. 1 shows the system error of XCSFES, XCSF with RLS and XCSF, respectively on f_p. The three systems have their system error below ε_0. The first two systems are significantly better than XCSF and there is little difference between XCSFES and XCSF with RLS. For ε_0=0.05, after about 50,000 problems, XCSF with RLS performance stays at 0.007 while XCSFES continues to evolve prediction weights and stabilizes performance at 0.005. For ε_0=0.10 and ε_0=0.20, XCSF with RLS performance is about

0.011 and 0.022, respectively while XCSFES performance is about 0.008 and 0.011, respectively.

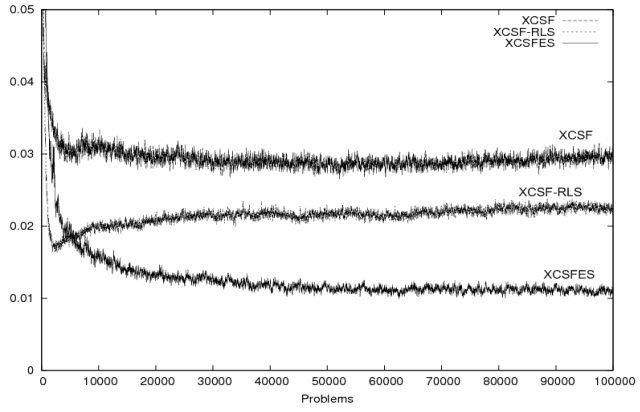
On f_{abs}, the system error of XCSFES, XCSF with RLS and XCSF is below ε_0 (Fig. 2). XCSFES is better than the others for each ε_0.



(a) ε_0=0.05

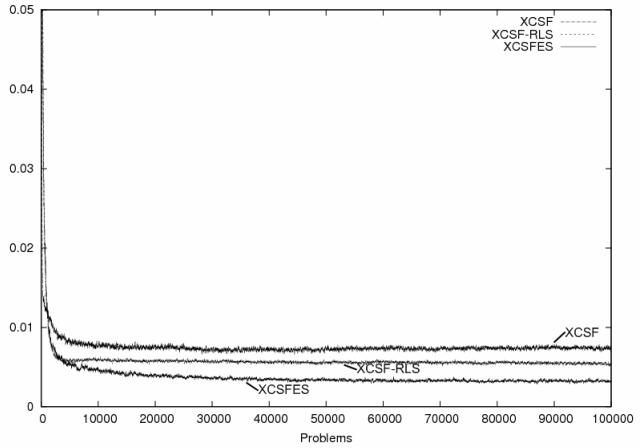


(b) ε_0=0.10

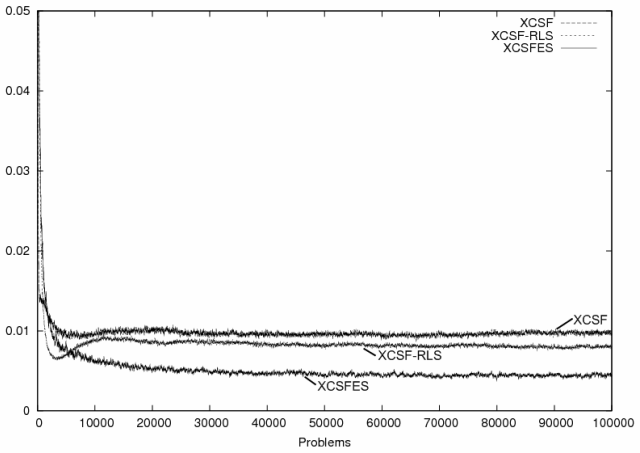


(c) ε_0=0.20

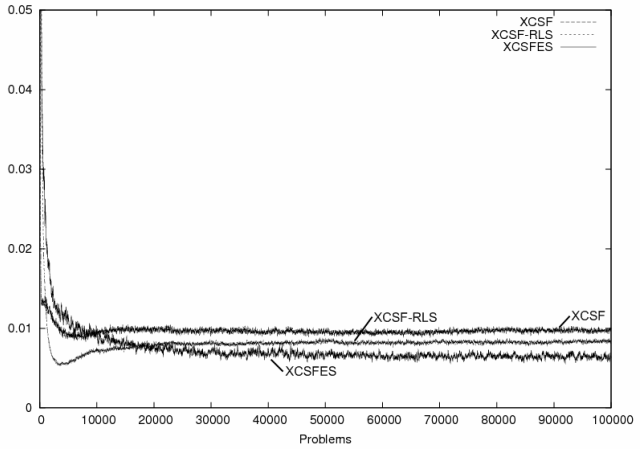
Figure 1. System error of XCSFES, XCSF with RLS and XCSF on f_p. (a), (b) and (c) correspond to 3 values 0.05, 0.10 and 0.20 for ε_0.



(a) ε_0=0.05



(b) ε_0=0.10

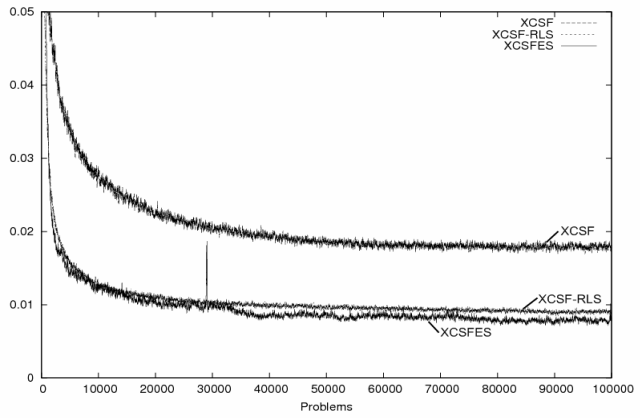


(c) ε_0=0.20

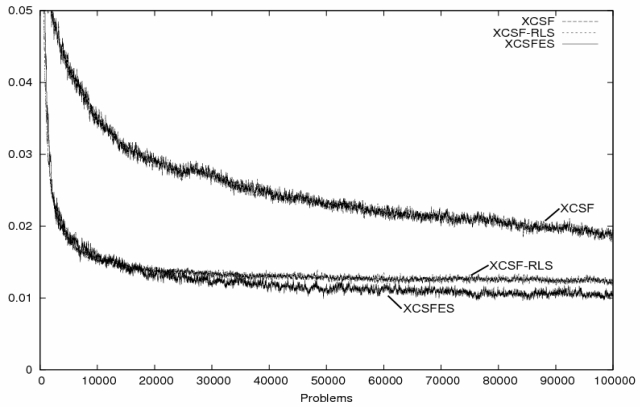
Figure 2. System error of XCSFES, XCSF with RLS and XCSF on f_{abs}

On f_{s3}, for ε_0=0.05 and ε_0=0.10, XCSFES performance is little difference with XCSF with RLS. For ε_0=0.20, XCSFES has less error than XCSF with RLS (Fig. 3).

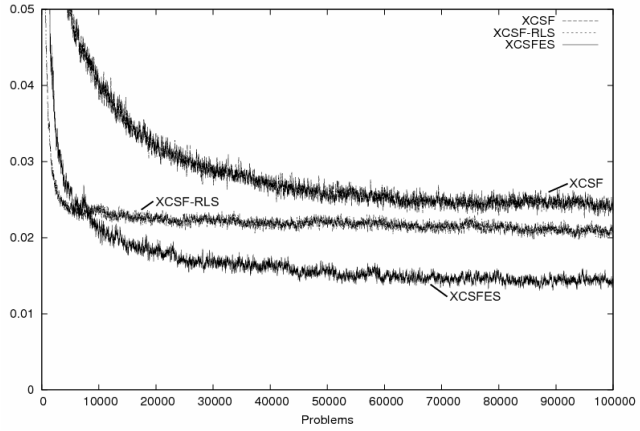
On f_{s4}, XCSFES and XCSF with RLS significantly outperform XCSF (Fig. 4). The system error of XCSFES and XCSF with RLS falls more quickly than the one of XCSF. For ε_0=0.05, after about 30,000 problems, the system error of XCSF with RLS decreases more slowly than the one of XCSFES. The same behaviors are obtained for ε_0=0.10 and ε_0=0.20.



(a) ε_0=0.05

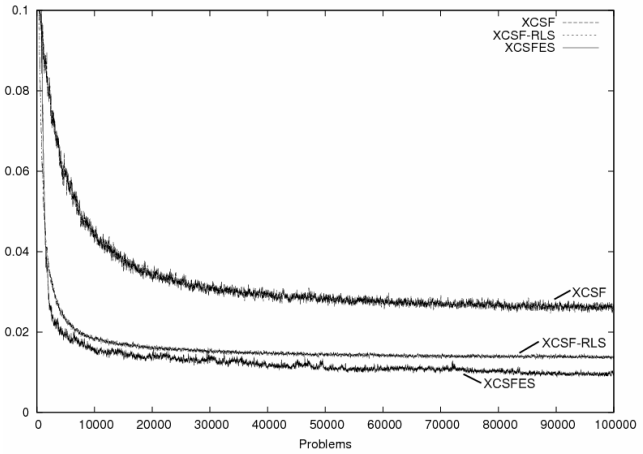


(b) ε_0=0.10

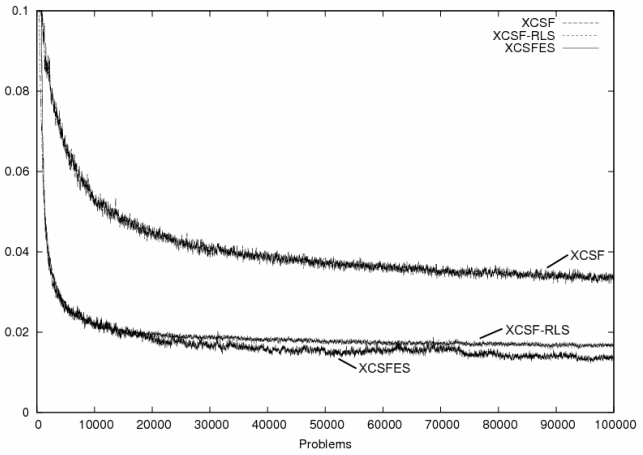


(c) ε_0=0.20

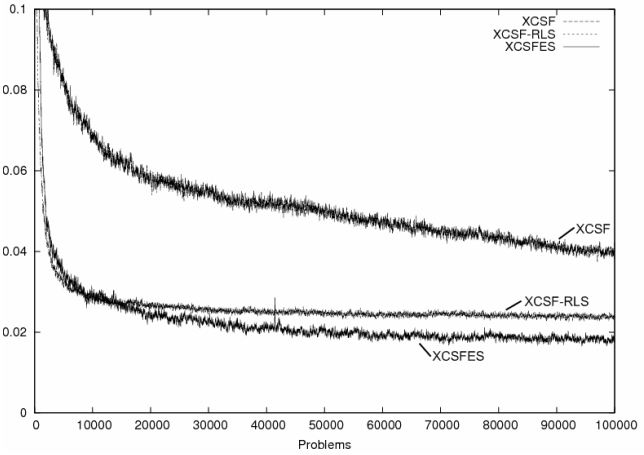
Figure 3. System error of XCSFES, XCSF with RLS and XCSF on f_{s3}



(a) ε_0=0.05



(b) ε_0=0.10



(c) ε_0=0.20

Figure 4. System error of XCSFES, XCSF with RLS and XCSF on f_{s4}

6. DISCUSSION AND CONCLUSION

The evolution strategy is an evolutionary method for evolving the prediction weights and is also considered as a kind of updating the

prediction weights, in the sense of modifying. It differs from "error-correcting" methods such as the modified delta rule and the recursive least square method, which directly correct the prediction weights.

The ES requires less computation time than the RLS method. The former requires $O(n)$ while the latter requires $O(n^2)$ [6]. If the input space is high, the ES needs more time to evolve prediction weights and thus it can be slower than the error-correcting methods. This is a disadvantage of the ES.

In this paper, we have presented a modification of XCSF, in which prediction weights of each classifier are not updated, instead they are evolved by using an evolution strategy. We use XCSFES to approximate a set of target functions. Although XCSFES uses the simple structure for the prediction weight vector and requires less computation time than XCSF with RLS, the results show XCSFES is able to evolve more accurate approximations. Our system is significantly more efficient than the original XCSF and obtains more accurate approximations than XCSF with RLS. More compact solutions are obtained by XCSFES.

Even though the functions tested have one dimensional input, the success obtained could lead to a future work that extends experiments to difficult target functions (e.g. high dimensional input space) and investigates the integration of other versions of the evolution strategy (e.g. covariance matrix adaptation evolution strategy) into XCSF.

Acknowledgements

The authors wish to acknowledge helpful and constructive comments of Stewart W. Wilson and the anonymous reviewer.

7. REFERENCES

[1] Butz, M. V., and Wilson, S. W., An Algorithmic Description of XCS, *Soft Computing*, 6(3-4), pp. 144-153, 2002.

[2] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[3] Kern, S., Müller, S. D., Hansen, N., Büche, D., Ocenasek, J., and Koumoutsakos, P., Learning Probability Distributions in Continuous Evolutionary Algorithms - A Comparative Review, *Natural Computing: an international journal*, Vol. 3, pp. 77-112, 2004.

[4] Lanzi, P. L., Loiacono, D., Wilson, S. W., and Goldberg, D. E., Extending XCSF beyond Linear Approximation, *GECCO'2005, Genetic and Evolutionary Computation Conference*, pp. 1827-1834, ACM Press, New York, USA, 2005.

[5] Lanzi, P. L., Loiacono, D., Wilson, S. W., and Goldberg, D. E., Classifier Prediction based on Tile Coding, *GECCO'2006, Genetic and Evolutionary Computation Conference*, pp. 1497–1504, ACM Press, New York, USA, 2006.

[6] Lanzi, P. L., Loiacono, D., Wilson, S. W., and Goldberg, D. E., Prediction Update Algorithms for XCSF: RLS, Kalman filter, and Gain Adaptation, *GECCO'2006, Genetic and Evolutionary Computation Conference*, pp. 1505-1512, ACM Press, New York, USA, 2006.

[7] Loiacono, D., and Lanzi, P. L., Evolving Neural Networks for Classifier Prediction with XCSF, *ECAI'2006, Workshop on Evolutionary Computation*, pp. 36-40, ACM Press, New York, USA, 2006.

[8] Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme und Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.

[9] Schwefel, H.-P., *Numerical Optimization of Computer Models*, Chichester: Wiley, 1981.

[10] Tran, T. H., Sanza, C., Duthen, Y., and Nguyen, D. T., XCSF with Computed Continuous Action, *GECCO'2007, Genetic and Evolutionary Computation Conference*, London, England, pp. 1861-1869, 2007.

[11] Tran, T. H., *Approches évolutionnaires pour le comportement adaptatif d'entités autonomes*, PhD thesis, University of Toulouse, IRIT lab., France, 2007.

[12] Widrow, B., and Hoff, M. E., Adaptive Switching Circuits, *Neurocomputing: Foundations of Research*, pp. 126-134, MIT Press, Cambridge, MA, 1988.

[13] Wilson, S. W., Classifier Fitness Based on Accuracy, *Evolutionary Computation*, 3(2), pp. 149-175, 1995.

[14] Wilson, S. W., Get Real! XCS with Continuous-Valued Inputs, *Festschrift in Honor of John H. Holland*, pp. 111-121, 1999.

[15] Wilson, S. W., Function Approximation with a Classifier System, *GECCO'2001, Genetic and Evolutionary Computation Conference*, San Francisco, CA, pp. 974-981, 2001.

[16] Wilson, S. W., Classifiers that Approximate Functions, *Natural Computing*, 1(2-3), pp. 211-234, 2002.

Table 2. (a), (b), (c) and (d) report XCSFES performance on the target functions f_p, f_{abs} f_{s3}, and f_{s4}, respectively. The first column is the error threshold ε_0, the second and third columns are the average RMSE with the standard deviation ($\overline{RMSE} \pm \sigma$) obtained by XCSFES, XCSF with RLS and the original XCSF, respectively, and the fourth and last columns are the average population size with the standard deviation ($|P| \pm \sigma$) of XCSFES, XCSF with RLS and XCSF, respectively.

ε_0	RMSE±σ (XCSFES)	RMSE±σ (RLS)	RMSE±σ (XCSF)	|P|±σ (XCSFES)	|P|±σ (RLS)	|P|±σ (XCSF)
0.05	0.008±0.002	0.010±0.001	0.015±0.002	94.560±10.112	124.680±7.004	120.340±10.184
0.10	0.011±0.003	0.015±0.002	0.023±0.004	105.580±9.833	130.660±7.522	134.600±7.017
0.20	0.014±0.006	0.029±0.003	0.037±0.005	115.360±10.033	134.120±7.727	136.680±7.145

(a) f_p

ε_0	RMSE±σ (XCSFES)	RMSE±σ (RLS)	RMSE±σ (XCSF)	|P|±σ (XCSFES)	|P|±σ (RLS)	|P|±σ (XCSF)
0.05	0.004±0.001	0.007±0.001	0.009±0.001	107.020±8.885	132.620±7.502	136.280±7.037
0.10	0.005±0.001	0.010±0.001	0.012±0.001	115.380±10.020	131.060±7.245	133.840±7.963
0.20	0.008±0.003	0.010±0.001	0.012±0.001	122.700±8.455	133.320±7.098	132.980±6.793

(b) f_{abs}

ε_0	RMSE±σ (XCSFES)	RMSE±σ (RLS)	RMSE±σ (XCSF)	|P|±σ (XCSFES)	|P|±σ (RLS)	|P|±σ (XCSF)
0.05	0.011±0.008	0.012±0.002	0.022±0.004	88.520±9.603	118.640±6.962	115.860±7.754
0.10	0.013±0.004	0.016±0.002	0.025±0.004	100.300±9.669	127.180±6.893	126.560±7.489
0.20	0.018±0.004	0.027±0.004	0.033±0.004	107.900±10.445	130.480±6.932	135.680±7.875

(c) f_{s3}

ε_0	RMSE±σ (XCSFES)	RMSE±σ (RLS)	RMSE±σ (XCSF)	|P|±σ (XCSFES)	|P|±σ (RLS)	|P|±σ (XCSF)
0.05	0.008±0.007	0.016±0.003	0.035±0.006	95.813±17.384	111.120±6.728	110.360±7.485
0.10	0.011±0.009	0.021±0.004	0.041±0.007	104.567±18.155	120.080±8.197	119.800±7.965
0.20	0.014±0.010	0.030±0.003	0.050±0.008	112.280±15.231	127.000±7.909	129.180±7.979

(d) f_{s4}