Improve and Compact Population in XCSFCA using Polynomial Equation

Saeid Goodarzian[1], Ali Hamzeh[2] and Sattar Hashemi[3]
[1]CE School, CSE and IT Department, Shiraz University, Shiraz, Iran

[2] CE School, CSE and IT Department, Shiraz University, Shiraz, Iran

[3] CE School, CSE and IT Department, Shiraz University, Shiraz, Iran

Abstract

XCS is a rule-based evolutionary online learning system. XCSFCA is an extension of XCS where compute continuous actions directly from input states. In XCSFCA, computed actions of a classifier, demonstrated as a straight lines. But in very problems, the desired best action curves are not linear and there are arched; therefore a system with linear action computation needs a large population. This paper studies a new method for compute continuous actions directly from input states. In new proposed method action computes by polynomial equation. Consequently, each classifier represents a nonlinear action curve and the classifiers are more generalized. In comparison with XCSFCA, our method proves to be more efficient and smaller population size.

***Keywords**: XCSF, XCSFCA, continuous action, polynomial equation.*

1. Introduction

Learning classifier systems define a new online model of genetic based machine learning where do not create a single solution while create multiple solutions, collected into a population, called *classifiers (rules).* Classifiers adaptively change to new classifiers, to make more accurate decisions in return for inputs. The description in [4], describes three major machine learning problem types that learning classifier systems can solve them. In optimization problems, learning classifier system search problem landscape to find the best solution at hand. In classification problems, LCSs provide class labels to partition the given input patterns into different classes. Moreover, in reinforcement learning problems [17], LCSs propose an action for a situation and after the receipt of the reward, they apply reward back-propagation method to updated and improve classifiers.

Each *classifier cl* in learning classifier systems consists of a *Condition cl.C* part, an *Action cl.A* part and a *Reward Prediction cl.R* value. Classifier *cl* predicts reward *cl.R* given its condition *cl.C* is satisfied, and given further that action *cl.A* is executed.

Learning classifier system as an online method, in each time step, senses a situation of problem environment and then checks all classifiers in its population [P] to find which classifier condition can satisfy the current situation, called matching. All matched classifiers are inserted into a list called match set [M], and then according to their eligibility, one action would be selected and performed on the environment, the involved classifiers are inserted into another list called action set [A]. Regarding to the action effects, the payoff value is received and used to update the parameters of the classifiers in the action set [A].

XCS [20] is one of the most well-known Learning Classifier Systems which attracted many researchers nowadays [11]. At 2002, as a mail stone, Wilson has introduced one of the most promising extensions to XCS, called XCSF, in [23]. The most important modification in XCSF with respect to XCS is its ability to compute the environmental payoff using an approximation method instead of tuning a real number. The overall architecture of XCSF is based on XCS and is very similar to XCSI [22]; a version of XCS with continuous classifier condition. In XCSI, each classifier has interval condition instead of traditional binary ones.

Considering current researches in XCS realm, it can be said that condition and prediction parts are two basic components which are mostly been improved and investigated [5, 12, 13, 14, 15 and 21]. However, in recent researches, the action part becomes more popular due to its importance role that can directly affect very large application area such as classification, approximation, simulation, control etc. [7, 8, 9 and 16]. Briefly, it must be noted that almost in all proposed architecture for a learning classifier system, the action part consists of an integer indicating particular class or effects in the environment among a limited list of candidates. However, in a newly introduced classifier system called Generalized Classifier System [25], a novel representation for the action part, named the continuous actions, is introduced. This scheme is able to compute the action instead of selecting it and therefore extends the range of possible actions from a limited discrete set to a continuous range. To describe in brief, it is worth mentioning that the structured of each classifier in GCS is formed as:

$$t(x,a) \Rightarrow p(x,a) \quad (1)$$

Where $t(x,a)$ is the condition part of a particular rule and $p(x,a)$ is its payoff prediction. In this paper, some improvements in XCSFCA will be investigated with the aim of increasing the performance and compacting the resulted population. Here, the classifier action is computed as a polynomial combination of the input and a vector of tuned coefficients.

The rest of this paper is organized as follows: in the next section we describe XCSF in brief, and then some relevant works on continuous action are reviewed. Then we described our proposed method and benchmark problems. At last, new method's results are presented and discussed.

2. XCSF in brief

XCSF [23] is a model of Learning Classifier System that extends the typical concepts of classifiers through the introduction of a computed classifier prediction. To develop XCSF, XCS has to be modified in three respects: (i) classifier conditions are extended for numerical inputs, as done in XCSI [22]; (ii) classifier are extended with a vector of weights$\vec{w}$, that are used to compute the classifier prediction; finally, (iii) The original update of the classifier prediction must be modified so that the weights are updated instead of the classifier prediction. These three modifications result in a version of XCS, XCSF [23] that maps numerical input into actions with an associated calculation prediction.

Classifiers: In XCSF, classifiers consist of a condition, an action and four main parameters. The condition specifies which input states the classifier matches; as in XCSI [22], it is represented 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. The four parameters are: (i) The weight$\vec{w}$, used to compute the classifier prediction as a function of the current input; (ii) The prediction error ε, that estimates the error affecting the classifier prediction; (iii) The fitness *F* that estimates the accuracy of the classifier prediction; (iv) 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 inputx_0, which 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 and creates a new classifier that matches the current inputs and has a random action. Each interval predicates$int_i = (l_i; u_i)$ in the condition of a covering classifier is generated as $l_i = s_t(i) - rand_1(r_0)$ and $u_i = s_t(i) - rand_1(r_0)$, where $s_t(i)$ is the input value of state s_t matched by the interval [0,r_0] with r_0 fixed integer. The weight vector $\vec{w}$ of covering classifiers is initialized with zero values (note in the original paper, weight are initialized with values in [0, 1]); all the other parameters are initialized as in XCS [20]. For each action a_i in [M], XCSF computes the system prediction that estimates the payoff that XCSF expects when action $a_i$is performed. 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. However, in contrast with XCS, in XCSF the classifier prediction is computed as a function of the current state s_t and the classifier weight vector$\vec{w}$. Accordingly, in XCSF system prediction is a function of both the current state s and the action a. Following a notation similar to [6], the system prediction for action a in state s_t, $P(s_t;a)$, is defined as equation 2:

$$P(s_t;a) = \frac{\sum_{cl \in [M]_a} cl.p(s_t) \times cl.F}{\sum_{cl \in [M]_a} cl.F} \quad (2)$$

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 computed in the state s_t. In particular, $cl.p(s_t)$ is computed as equation3

$$cl.p(s_t) = cl.w_0 \times x_0 + \sum_{t>0} cl.w_i \times s_t(i) \quad (3)$$

Where $cl.w_i$ is the weight w_i of cl and x_0 is a constant input.The values of $P(s_t;a)$ form the prediction array. Next, XCSF select s an action to perform. The classifiers in [M] that advocate the selected action are put in the current action set [A]; the selected action in sent to the environment and a reward r is returned to the system together with next input state s_{t+1}.

Reinforcement Component: XCSF uses the incoming reward r to update the parameters of classifiers in action set [A]. First, the reward r is used to update the weight vector $\vec{w}$ using a modified delta rule [19] as follows for each classifier $cl \in [A]$, each weight $cl.w_i$ is adjusted by a quality Δw_i computed as equation (4):

$$\Delta w_i = \frac{\eta}{\|\vec{x}_{t-1}\|^2}\left(r - cl.p(s_{t-1})\right)x_{t-1}(i) \quad (4)$$

Where η is a correction rate and $\vec{x}_{t-1}$ is defined as the input state vector s_{t-1} augmented by a constant x_0 (i.e.$x_{t-1} = \langle x_0, s_{t-1}(1), s_{t-1}(2), \dots, s_{t-1}(n)\rangle$) and

$||\vec{x}_{t-1}||^2$ is the norm of vector $\vec{x}_{t-1}$ for further details refer to [23]. The values $\Delta w_i$are used to update the weights of classifier cl as equation 5.

$$cl.w_i \leftarrow cl.w_i + \Delta w_i \quad (5)$$

Then the prediction error $\varepsilon$is updated as equation 6:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|r - cl.p(s_{t-1})| - cl.\varepsilon) \quad (6)$$

Where β is the learning rate. Classifier fitness is updated as in XCS. First, the raw accuracy κ of the classifiers in [A] is computed as equation 7.

$$cl.\kappa = \begin{cases} 1 & if\ cl.\varepsilon < \varepsilon_0 \\ \alpha(\frac{cl.\varepsilon}{\varepsilon_0})^{-v} & otherwise \end{cases} \quad (7)$$

Where ε_0 is a constant that controlled the acceptable values of prediction error ε. If $cl.\varepsilon$ is less than ε_0 the error is accepted and classifier is accurate ($cl.\kappa = 1$), otherwise the accuracy of classifier cl is controlled by parameters α and v. For represent efficient accuracy of each classifier, $cl.\kappa'$calculated respect to other classifier accuracies and repetition, so the raw accuracy $\kappa$is used to calculate the relative accuracy κ' as equation 8.

$$cl.\kappa' = \frac{cl.\kappa \times cl.uum}{\sum_{j\in[A]} cl_j.\kappa \times cl_j.num} \quad (8)$$

Finally, the relative accuracy $\kappa'$is used to update the classifier fitness as equation 9.

$$cl.F = cl.F + \beta(cl.\kappa' - cl.F) \quad (9)$$

An algorithmic description of the overall update procedure is reported in [13].

Discovery Component: The genetic algorithm in XCSF [23] works as in XCSI [22]. On a regular basis depending on the parameterθ_{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 [22]. The resulting offspring are inserted into the population, if the population size is exceeding the maximum size of population, deletion method performed to delete the excessive classifiers.

3. Related works on continuous actions in LCS

In LCSs, actions are typically fixed, discrete, and encoded by a set of symbols, e.g. {0, 1}, {“Left”, “Right”, and “Top”, “Down”}, {“12”,”15”…”63”}. However, continuous real-valued actions are desirable in many applications especially where fine reactions are more important. This is very complicated for a LCS with discrete actions to handle the continuous real-valued action range; therefore, LCSs that can generate the actions according to sensory inputs are interested. This section reviews two notable investigations in this area.

3.1 Generalized Classifier System (GCS)

In [25], Wilson described three distinct classifier system architectures for continuous action. Generalized classifier system (GCS) is more applicable and remarkable architecture in comparison with the other one. Many basic parts of GCS inherited from XCSF however extensions where permit continuous action describes here. Each classifier in GCS structured in format of:

$$t(x,a) \Rightarrow p(x,a) \quad (10)$$

Where $t(x,a)$ is the condition part of a particular rule, and $p(x,a)$ is its payoff prediction computed as a linear combination of weight vector$\vec{w}$ and collected vector of input *x*, action a and a constant value namedx_0. Satisfying $t(x,a)$ is related to both values x and a. since for each time step t best action for each satisfied classifier is desirable, equation (11) was used to compute the best action:

$$a^*(x) = max_{a\in A}\, P(x,a)|\, t(x,a) = true \quad (11)$$

Where $a^*(x)$ is the best action for input x and A is a continuous range for valid actions.

In GCS, author inspired the idea from an investigation on XCSF where the condition parts of classifiers are represented as general hyper-ellipsoidal [5].

Satisfying $t(x,a)$ is depends on the values of *x* and a. So, although to form the match set [M] the values of both *x* and *a* are needed, the action *a* is the system output and not available yet. So in exploration phase a random action a generated, but in exploitation phase a different methodology is used to find the a_{best} (the best action of a particular classifier) which is desirable for the learner. In exploitation phase, a classifier would be a member of [M] if its condition part matches the input *x* and has any value of $a \in A$. Suppose thata_l and$a_u$is the minimum and maximum value of$a \in A$. Since,$p(x,a)$is computed linearly. Thus, it is clear that through all possible values of a either a_l or$a_u$can lead to maximize$p(x,a)$, in other words one of $p(x,a_l)$ or $p(x,a_u)$ have the highest prediction value. So, either a_l ora_u will be selected as the a_{best}. Finally, fora^*, the system picksthe best of all a_{best} that yield higher prediction among the others.

IJCSI
www.IJCSI.org

3.2 XCSF with computing continuous actions (XCSFCA)

XCSFCA is an extension of XCSF that can be applied on the environments where the action could be assumed as a computable function with respect to the environmental input. However, in the recent GCS [25], actions are selected from a continuous range; also in XCSFCA, actions are directly computed from a continuous function of the input. In XCSFCA, the classifier action $cl.a$is computed as equation (12) suggested:

$$cl.a(x,\zeta) = \zeta.x'$$
$$x' = (x_0, x_1, \dots, x_n) \quad (12)$$

Where ζ is a vector of action weights and x_0 is a constant, also a vector of mutation rate $cl.\sigma$is added to each classifier to be used in action weight updating phase. An evolutionary strategy (ES [10]) evolves the action weights to compute actions that are more accurate. The XCSFCA principal changes as follow:

The process of building the match set [M]:XCSFCA builds a match set [M] containing the classifiers in the population [P] whose condition matches the current sensory input and its computed action $cl.a$ belongs to the range of actiona_{range}, where $a_{range}$is a range of acceptable values for actions.

Covering operator: classifier $cl$is accepted and inserted into population [P] and consequently into match set [M] if the computed action $cl.a$ belongs to the allowed action range.

Action selection in exploration: the action with the highest prediction is selected.

4. Extension to XCSFCA by polynomial function

As described in section 3.2, XCSFCA uses a linear combination of action weight vector $cl.\zeta$and x' to compute actions in reply to specified environmental state *x*. The length of weight vector ζ is related to problem landscape dimension. In XCSFCA classifiers action curve represented as straight line. Since the classifier action *cl.A* must align to desired action curve, while the problem is not complex and the relation between input *x* and best action $a*$ is uniformly ascending or descending, the linear actions is efficient. In more complex problems the curve of best action $a*$is not uniformly ascending or descending and it is not linear like and it is arched. Therefore, XCSFCA must evolve a large population to handle the problem landscape entirely, because a particular classifier in XCSFCA just works finely in a small area of problem landscape. If the classifiers activation area becomes undersized, more classifiers need to cover problem landscape. For these types of problems, in one side, producing a suitable population and in the other side evolving action weight vector $cl.\zeta$for a particular classifier is time consuming processes, also population size, proportionally grown up.

In this section, we describe that the action computation function can change to reach another mapping type of input *x* to action *a*.

We replaced action weight vectorζ with action computation coefficients $\upsilon = (\upsilon_0, \upsilon_1, \upsilon_2, \dots, \upsilon_n)$. If our modification of XCSFCA, the classifier action $cl.a$ is computed as a polynomial equation of υ and the current input x. Degree of polynomial is related to problem, e.g. for a polynomial with degree 2 the classifier action computed as equation 13:

$$cl.a(x,\upsilon) = \upsilon_2 * x^2 + \upsilon_1 * x + \upsilon_0 \quad (13)$$

It is clear that, for a low dimensional problem, a polynomial function with opportunely degree is more powerful to compute actions because the number of action computation coefficients $\upsilon$is related to polynomial degree and we can adjust the polynomial degree considering the problem complexity. In the other point of view, through using polynomial function for compute actions, a particular classifier is able to cover a larger subsection of problem landscape because this is more flexible and better to be aligned with best actions curve.

We expected that, modified XCSFCA using polynomial function can solve problems that are more complex, also it can compact the population size in the simplest problems. The promising results presented in this paper approve our claim.

5. Experimental Setup

This section described *frog1* and *frog2* problems from [24] and [18]. Also for better examination, our proposed method a new *frog* problem introduced. In comparison with *frog1* and *frog2*, the new *frog* problem is more difficult.

5.1 *Frog* problems

The *frog* problem introduced in [24] is a one-dimensional problem describing a frog who wants to catch a fly that is located at the distance *d*. The frog senses the distance *d* through a value named *x* calculated as equation (15), and jumps with respect to *x* then the frog receive the payoff using the function given as equation (14):

$$P(x,a) = \begin{cases} x + a & if\ x + a \le 1 \\ 2 - (x + a) & otherwise \end{cases} \tag{14}$$

$$x(d) = 1 - d \tag{15}$$

As an extension to the original *frog* problem, the *frog2*problem is introduced in [12] where both payoff function and transformation of d to x are modified as equation (16) and (17):

$$P(x,a) = \begin{cases} xe^{a} & if\ a \le -lnx \\ x^{-1}e^{-a} & otherwise \end{cases} \tag{16}$$

$$x(d) = e^{-d} \tag{17}$$

Fig. 1 and Fig. 2 show the best action $a*$ of the *frog1* and *frog2* with respect to x. We introduce an extension of *frog* problem called *Frog3*, which is demonstrated in fig. 3(a, b). In *Forg3*, the payoff function is still continuous and nonlinear which is composed of two nonlinear forms, also the relation between x and $a*$ is not uniformly ascending or descending. The payoff function is computed as equation (18) and (19):

$$(x,a) = \begin{cases} \dfrac{a}{\frac{1}{2}(\sin(2\pi x)+1)} & if\ a \le \frac{1}{2}(\sin(2\pi x)+1) \\ \dfrac{a-\frac{1}{2}(\sin(2\pi x)+1)}{\frac{1}{2}(\sin(2\pi x)+1)-1} + 1 & otherwise \end{cases} \tag{18}$$

$$x(d) = \tfrac{1}{2}(\sin(2\pi d)+1) \tag{19}$$

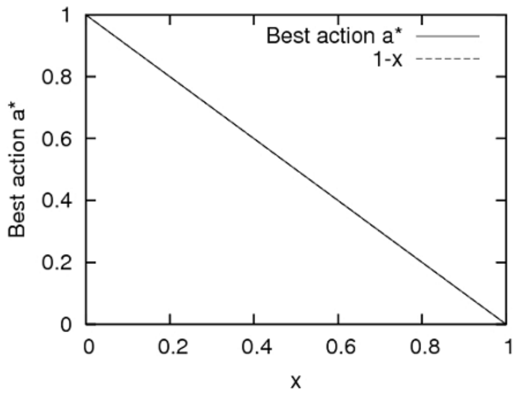

Figure 1: Best action a* of *frog1*

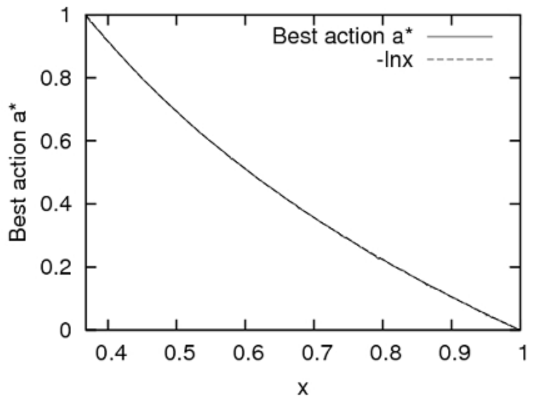

Figure 2: Best action a* of *frog2*

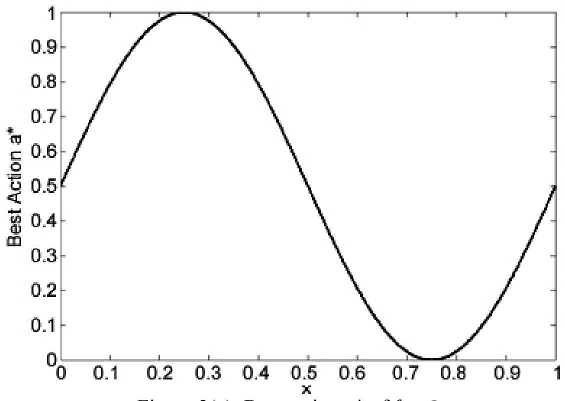

Figure 3(a): Best action a* of *frog3*

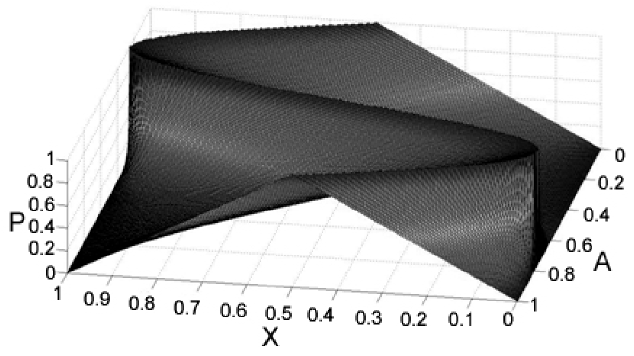

Figure 3(b): Payoff for all input *x* and action *a* of *frog3*

5.2 Simulation Setting

All the experiments discussed in this paper are performed following the standard design used in the literature [25]. To apply fair comparison of our results with other related work results, we used the minimal change to parameter setting. The parameters setting for the experiment were as follows:$N = 2000, \beta = 0.5,\ \alpha = 0.1,\ \eta = 0.2,\ \delta = 0.1,$

$\theta_{GA} = 48$, $\nu = 5$, $\varepsilon_0 = 0.01$, $\mu = 0.04$, $\chi = 0.8$, $\theta_{del} = 50$. Both GA-subsumption and action set subsumption were not activated. Each run would stop after 100,000 explore problems. Explore and exploit problems are experimented one after one. As a polynomial with degree 2is used to compute the actions, each classifier has an array with 3elements $\upsilon = (\upsilon_0, \upsilon_1, \upsilon_2)$ as action computation coefficients. The payoff would be received after applying the selected action. The fly positions were randomly selected from continuous range [0, 1]. The actions were computed within continuous range [0, 1].To plot the best action a, x was scanned from 0 to 1 and from e^{-1} to 1 and from 0 to 1 in *frog1*, *frog2* and *frog3*problems respectively, increased by 0.001. The best action curve is plotted, averaged over ten runs.

6. Results

6.1. Frog1`s results.

Fig. 4 shows the system performance (in black), population size (in blue) and system error (in red).As it Fig.4 shows, the system performance is greater than 99% and system error drops to smaller than 1% after 21000 explore problems. The final population size of classifiers N is smaller than 19% of N, and it is worth to mentioning that the final population size is about half of XCSFCA population size showed in [18].

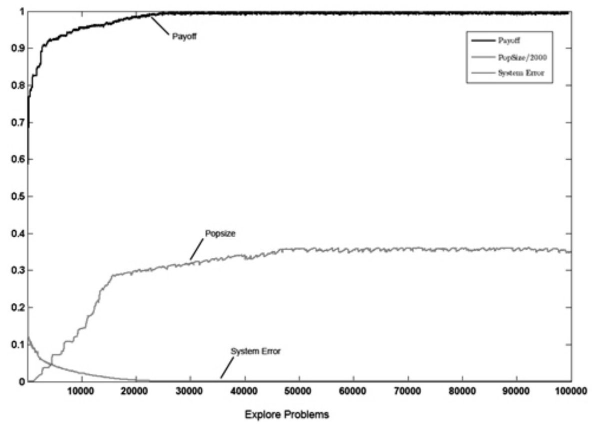

Figure 3: Results of *frog1* average over ten runs

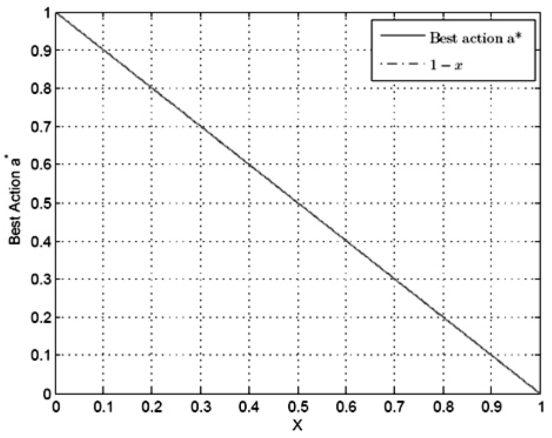

Figure 4: Best action $a*$ of *frog1* is plotted by scanning the values of x from 0 to 1, increased by 0.001 averaged over ten runs. Best action $a*$ is slightly borken at some input x

Fig. 5 shows the best action $a*$ of *frog1* which is very similar to diagonal *1-x*. The *frog* problem seems simple, however the mapping from real distance d to transformed distance x makes it as a benchmark because this mapping from d to x hide the real position of fly and the frog-like system to be compelled to catch the fly using the payoff values. In the finalized population, the vectors of action computation coefficients υ_0, υ_1 *and* υ_2 are close to 1, -1 and 0 respectively, and the standard deviations σ_0, σ_1 *and* $\sigma_2$are very close to zero so in late phases the action computation coefficients $\upsilon = (\upsilon_0, \upsilon_1, \upsilon_2)$ have little change.

Table1, shows 20 more activated classifiers from one run of *frog1*, where selected from the finalized population. In Table 1, l_0 and $l_1$are the values for the interval predicate; w_0, $w_1$and w_2 are the prediction weights for constant x_0 and the input x and the action *a* respectively; υ_0, υ_1 andυ_2 are the action computation coefficients for x_0, x and x^2 respectively; σ_0, σ_1 *and* σ_2 are the standard deviations used by mutation on υ_0, υ_1 andυ_2 respectively;ε, *fit* and *num* indicate prediction error, classifier fitness and numerosity respectively. For example in the last row of Table 1, classifier condition starts from 0.302 and spreads to 0.935, so from this interval x = ***0.1*** and*Cl.a*is calculated by Eq.(13) as follows: $Cl.a = \upsilon_2 * 0.01 + \upsilon_1 * 0.1 + \upsilon_3 = -0.0021 * 0.01 + -1.0023 * 0.1 + 0.9995 =$ **0.8992**

6.2. Frog2`s results.

IJCSI
www.IJCSI.org

However *frog2* is more difficult than *frog1*; Fig. 6 shows the system performance, population size and system error of *frog2* where averaged over ten runs. The system performance is greater than 99% and the system error drops to smaller than 1% after 22000 explore problems. The final population size of classifiers N is smaller than 20% of N where it is about half of XCSFCA population size showed in [18]. Fig. 7 shows the best action $a*$ of *frog2* which is very similar to curve *–ln(x)*.

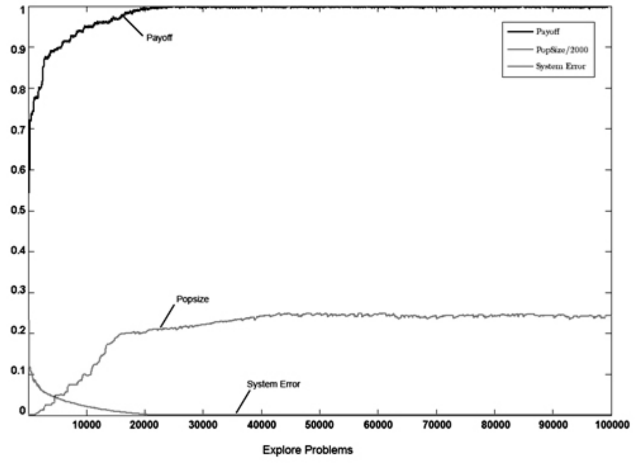

Figure 6: Results of *frog*2 average over ten runs

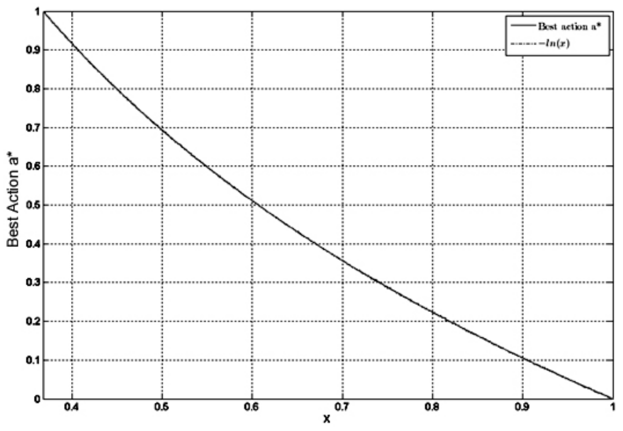

Figure 7: Best action $a*$ of *frog*2 is plotted by scanning the values of x from 0 to 1, increased by 0.001 averaged over ten runs. Best action $a*$ is slightly borken at some input x

Fig. 8 shows two classifiers with similar condition part where the first one calculate actions using linear function like XCSFCA and the second one calculate actions using polynomial equation. The condition part of both classifiers are *Cl.c = [0.503, 0.812]*, but the action weight vector of first classifier is$Cl.\zeta$=(ζ_0, ζ_1) = (-1.5210, 1.4251) and action computation coefficient of second one isCl. υ = (υ_0, υ_1, υ_2) = (1.90, -2.99, 1.11). Fig. 8 clearly shows that the second classifier activity area is larger than first; consequently, in this type of action computation, the number of classifiers for handle a wide area of problem is smaller than linear case and the population tends to be compacted.

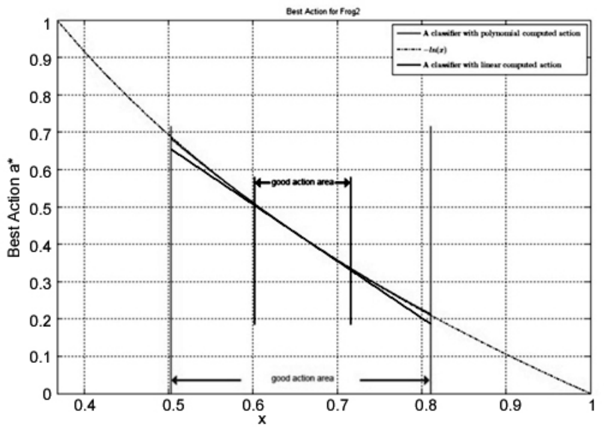

Figure 8: Compare two classifier activites;first classifier with polynomial computed action and second classifier with linear computed action

Table2, shows 35 more activated classifiers from one run of *frog2*, which are selected from the finalized population. All columns of Table 2 are same as Table 1. For example, in the first row of this table, the classifier condition starts from 0.398 and spreads to 0.706,so from this interval x = ***0.4*** and *Cl.a*is calculated by Eq.(13) as follows: $Cl.a = \upsilon_2 * 0.16 + \upsilon_1 * 0.4 + \upsilon_3 = 1.6209 * 0.16 + -3.6205 * 0.4 + 0.1.6209 =$ **0.9137**while –*ln(0.4)=0.09163.*

6.3. Frog3`s results.

In Section 5.1, we described that both *frog1* and *frog2* are uniformly descending but *frog3* is not uniformly descending and has two direction changes. Calculating action by linear function is effective for *frog1* and *frog2* and similar problems because each classifier calculates the actions as a line, but for *frog3* linear function is not effective and need more classifier to cover all points of problem landscape.

Fig. 9 shows the system performance (in black), population size (in blue) and system error (in red) of *frog3*where averaged over ten runs. The system performance is greater than 99% and the system error drops to smaller than 1% after 29000 explore problems. The final population size of classifiers N is smaller than

IJCSI
www.IJCSI.org

30% of N. Fig. 10 shows the best action $a*$ of *frog3* which is very similar to curve $\frac{1}{2}(\sin(2\pi x)+1)$.

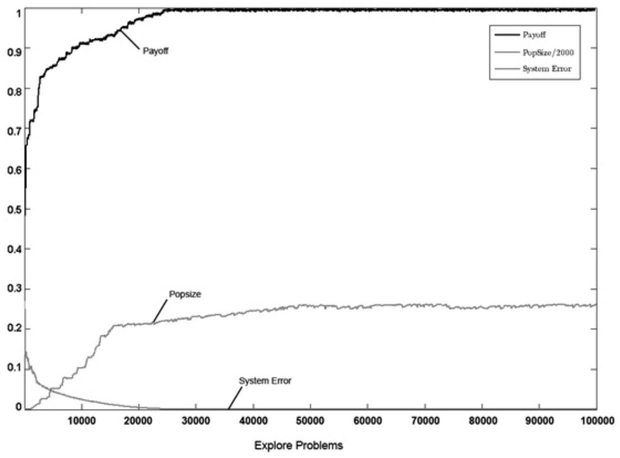

Figure 9: Results of *frog3* average over ten runs

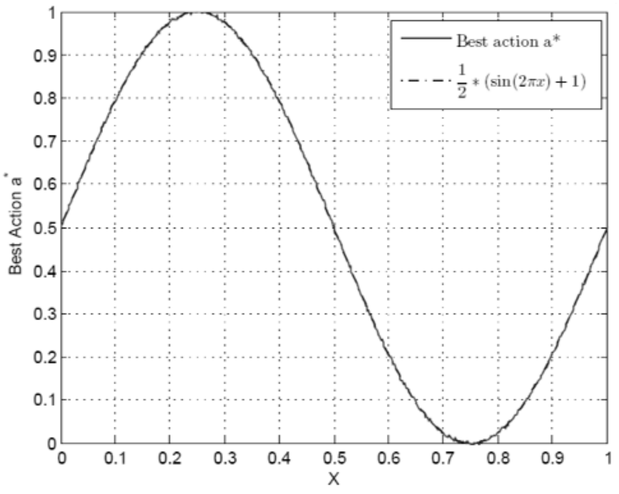

Figure 10: Best action a* of *frog3* is plotted by scanning the values of x from 0 to 1, increased by 0.001 averaged over ten runs. Best action a* is slightly borken at some input x

Fig. 11 shows a classifier activity for *frog3* problem where selected from the finalized population of one run. In Fig. 11, the classifier condition is $Cl.c = [\boldsymbol{l_0},\boldsymbol{u_1}]$ = [0.105782, 0.54728], the classifier actions are calculated by Eq. (13) where x starts from$\boldsymbol{l_0}$to$\boldsymbol{u_0}$ and increases by 0.001 and the action computation coefficients is.$v=(v_0,v_1,v_2)=(0.47778, 4.089679, -8.149012)$. Fig. 11 clearly shows that computing action using polynomial equation makes classifiers that are more powerful and in more complex problems such as *frog3*, XCSFCA could solve problem with smaller population size.

Table 3, shows 45 more activated classifiers from one run of *frog3*, that are selected from the finalized population. All columns of this table are same as Table 1.Fig. 12 shows the activity of classifiers showed in Table 3. Fig. 12 demonstrates that all points of curve $\frac{1}{2}(\sin(2\pi x)+1)$can covered by the classifiers with continuous action using polynomial equation and we can conclude that cover this curve by classifiers that use linear function to compute actions, is more difficult and consequently needs larger population.

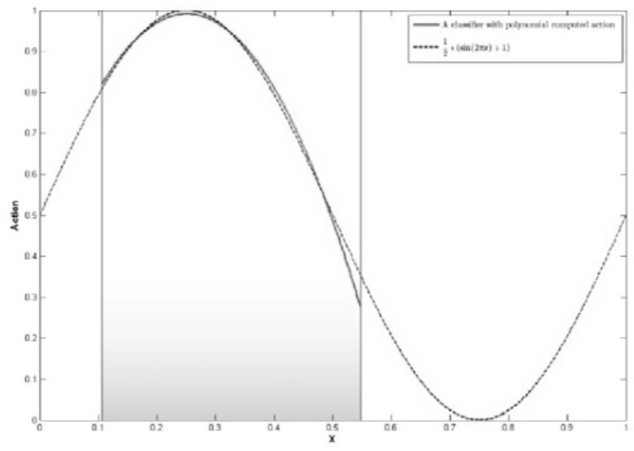

Figure 11: A classifier with polynomial computed action for *frog3* problem

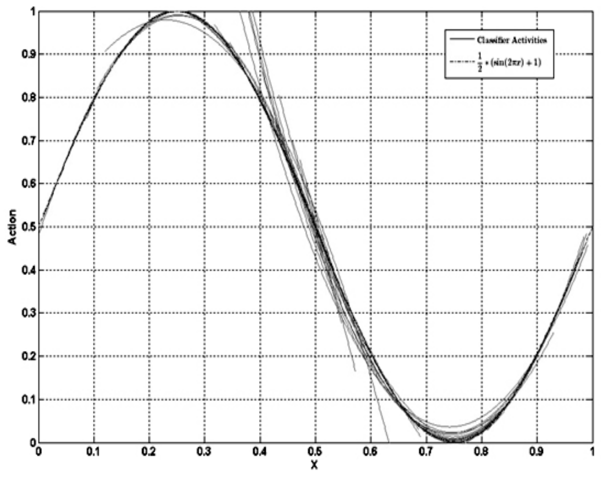

Figure 12: Classifier activities for *frog3*takes from table3

7. Conclusion

In this paper we presented a new method for calculating continuous actions in which there would be directly computed as a polynomial equation of the input state a and a vector of polynomial coefficient $Cl.v=(v_0,v_1,v_2)$. We have shown that computing action using polynomial equation not only can solve the previous problems but it can solve more difficult problems with smaller population size. In our proposed method the classifiers is more general because the mapping from input states to actions is none linear. Therefore, the action curve effectively can align to desired action curve. For better examination, we introduced a new *frog* problem called *frog3* where this is more difficult in compare to *forg1* and *frog2*.

Appendix

Table 1: Activated classifiers from one run of *frog1*

l_0	l_1	w_0	w_1	w_2	v_0	v_1	v_2	σ_0	σ_1	σ_2	ε	*fit*	*num*
0.564	0.895	551.34	447.53	448.30	0.9435	-0.8359	-0.1133	0.0003	0.0002	0.0002	0.00003	0.932543	17
0.422	0.453	552.12	446.15	447.03	0.7595	0.1012	-1.2599	0.0001	0.0002	0.0001	0.00001	0.85019	34
0.512	0.696	551.15	446.17	446.39	0.8351	-0.4288	-0.4888	0.0002	0.0001	0.0002	0.00000	0.931921	26
0.405	0.724	550.76	449.89	449.99	1.0099	-1.0494	0.0562	0.0001	0.0000	0.0001	0.00002	0.936204	2
0.435	0.670	551.67	447.81	447.47	1.0783	-1.3053	0.2960	0.0001	0.0003	0.0002	0.00000	0.900447	39
0.309	0.817	551.34	446.71	445.91	0.9762	-0.8981	-0.1010	0.0003	0.0002	0.0001	0.00003	0.914114	20
0.351	0.834	550.17	448.96	448.56	0.9965	-0.9972	0.0016	0.0001	0.0003	0.0000	0.00003	0.912424	35
0.119	0.519	552.13	448.01	448.70	1.0094	-1.0726	0.1227	0.0001	0.0002	0.0002	0.00000	0.886267	32
0.053	0.506	550.81	448.93	448.68	1.0059	-1.0317	0.0339	0.0002	0.0000	0.0003	0.00001	0.933522	23
0.036	0.610	550.19	449.24	450.03	0.9957	-1.0011	0.0201	0.0003	0.0002	0.0003	0.00002	0.910299	18
0.014	0.751	550.37	449.38	448.45	0.9948	-0.9458	-0.0745	0.0001	0.0002	0.0002	0.00001	0.981172	14
0.406	0.847	550.46	449.51	448.80	1.0601	-1.2017	0.1624	0.0001	0.0002	0.0001	0.00000	0.886937	12
0.297	0.787	550.76	449.07	449.85	1.0262	-1.0952	0.0829	0.0000	0.0002	0.0003	0.00003	0.935824	7
0.591	0.965	550.24	448.99	449.85	0.9449	-0.8878	-0.0478	0.0002	0.0000	0.0001	0.00002	0.98569	36
0.125	0.782	550.18	448.87	449.22	1.0112	-1.0508	0.0465	0.0002	0.0002	0.0001	0.00000	0.859041	33
0.458	0.942	551.19	448.65	448.77	0.9300	-0.7961	-0.1428	0.0002	0.0000	0.0001	0.00002	0.869776	30
0.102	0.926	550.71	449.03	449.00	0.9942	-0.9912	-0.0026	0.0001	0.0000	0.0002	0.00001	0.923078	17
0.212	0.972	550.32	449.59	449.64	0.9925	-0.9505	-0.0465	0.0001	0.0000	0.0001	0.00001	0.918337	18
0.224	0.867	550.83	448.89	449.78	0.9887	-0.9548	-0.0390	0.0001	0.0003	0.0001	0.00001	0.863084	15
0.041	0.935	550.11	449.82	449.30	0.9995	-1.0023	-0.0021	0.0001	0.0001	0.0002	0.00000	0.934733	27

Table 2: Activated classifiers from one run of *frog2*

l_0	l_1	w_0	w_1	w_2	v_0	v_1	v_2	σ_0	σ_1	σ_2	ε	*fit*	*num*
0.398	0.706	604.256	396.477	396.526	2.1026	-3.6205	1.6209	0.0003	0.0003	0.0000	0.0000	0.9537	9
0.391	0.629	585.297	414.412	413.953	2.1997	-4.0158	1.9946	0.0003	0.0000	0.0002	0.00002	0.8696	37
0.399	0.643	561.773	438.986	439.848	2.2198	-4.0811	2.0620	0.0001	0.0000	0.0001	0.00002	0.9758	28
0.405	0.500	590.845	409.476	410.112	2.3525	-4.6685	2.7014	0.0003	0.0000	0.0002	0.00002	0.8987	25
0.405	0.516	576.622	422.729	421.877	2.2265	-4.1553	2.1497	0.0001	0.0003	0.0003	0.00002	0.8611	9
0.420	0.536	612.416	387.540	388.513	2.3077	-4.4925	2.5071	0.0001	0.0001	0.0003	0.00003	0.9282	27
0.420	0.629	579.191	420.497	419.617	2.1635	-3.8855	1.8803	0.0002	0.0001	0.0001	0.00003	0.9713	50
0.446	0.742	639.625	360.915	360.770	1.9903	-3.2264	1.2854	0.0003	0.0002	0.0002	0.00001	0.9022	43
0.479	0.791	588.738	411.101	410.245	1.9896	-3.2382	1.2924	0.0001	0.0003	0.0001	0.00002	0.8521	2
0.480	0.799	600.556	399.426	398.663	1.9768	-3.1913	1.2548	0.0002	0.0001	0.0002	0.00000	0.9100	16
0.490	0.608	572.706	427.506	428.083	2.8821	-6.5533	4.3381	0.0003	0.0001	0.0001	0.00002	0.9063	28
0.503	0.812	598.578	401.457	401.738	1.9052	-2.9885	1.1108	0.0002	0.0001	0.0003	0.00001	0.9657	48
0.530	0.759	625.036	374.677	374.576	1.9234	-3.0317	1.1385	0.0001	0.0000	0.0002	0.00001	0.9430	49
0.530	0.783	598.628	402.074	402.496	2.0107	-3.3034	1.3426	0.0003	0.0000	0.0002	0.00003	0.9363	4
0.531	0.838	630.043	369.081	369.800	1.9341	-3.0842	1.1761	0.0001	0.0000	0.0001	0.00003	0.9007	9
0.541	0.849	596.694	403.953	404.266	1.9566	-3.1262	1.2020	0.0000	0.0003	0.0001	0.00002	0.9191	31
0.546	0.740	613.995	386.354	385.797	1.8822	-2.9157	1.0560	0.0001	0.0002	0.0002	0.00001	0.8973	34
0.548	0.663	628.909	370.247	371.084	1.8607	-2.8304	0.9648	0.0003	0.0000	0.0003	0.00002	0.9085	37
0.566	0.693	648.515	350.579	349.597	1.9129	-3.0237	1.1376	0.0003	0.0001	0.0001	0.00001	0.9135	13
0.568	0.993	602.391	397.975	397.244	1.8327	-2.7698	0.9414	0.0003	0.0001	0.0001	0.00001	0.9651	13
0.574	0.980	611.149	389.761	389.652	1.8068	-2.7000	0.9014	0.0001	0.0001	0.0002	0.00002	0.9418	69
0.579	0.939	627.030	373.264	373.685	1.8066	-2.7032	0.9022	0.0003	0.0001	0.0003	0.00001	0.9967	6
0.584	0.763	585.835	413.546	412.997	2.0794	-3.4752	1.4639	0.0000	0.0000	0.0002	0.00002	0.9611	45
0.602	0.748	655.315	345.240	346.014	1.9172	-3.0460	1.1553	0.0001	0.0002	0.0001	0.00001	0.9282	33
0.608	0.987	569.095	431.738	431.644	1.7512	-2.5447	0.8046	0.0003	0.0002	0.0002	0.00002	0.9981	5
0.613	0.757	573.704	427.036	427.983	1.8813	-2.9274	1.0793	0.0002	0.0002	0.0002	0.00002	0.9036	8
0.644	0.979	578.538	421.785	421.784	1.6683	-2.3524	0.6925	0.0000	0.0000	0.0002	0.00001	0.9886	10
0.657	0.863	621.729	377.725	378.440	1.8960	-2.9551	1.0741	0.0001	0.0001	0.0001	0.00001	0.9741	19
0.668	0.942	594.465	404.623	404.831	1.6236	-2.2387	0.6194	0.0002	0.0003	0.0003	0.00001	0.9227	26
0.670	0.845	615.929	383.544	384.027	1.6384	-2.2888	0.6479	0.0002	0.0002	0.0002	0.00000	0.9118	53
0.681	0.991	587.441	411.934	412.173	1.6209	-2.2347	0.6040	0.0000	0.0000	0.0001	0.00001	0.9442	21
0.683	0.871	644.612	354.853	354.850	1.6956	-2.4515	0.7647	0.0002	0.0001	0.0001	0.00001	0.8855	25
0.738	0.831	630.146	370.818	371.600	0.5328	0.5937	-1.2293	0.0002	0.0002	0.0002	0.00001	0.9312	17
0.838	0.992	601.617	398.968	399.775	-0.5003	2.4174	-1.9341	0.0002	0.0002	0.0002	0.00003	0.8864	19

Table 3:Activated classifiers from one run of *frog3*

l_0	u_0	w_0	w_1	w_2	v_0	v_1	v_2	σ_0	σ_1	σ_2	ε	fit	num

0.236866	0.378151	557.448	442.355	442.105	0.462892	4.367334	-8.863423	0.0001	0.0000	0.0003	0.00001	0.9021	23
0.396261	0.504303	610.746	389.721	390.664	2.585839	-5.64389	2.983449	0.0001	0.0001	0.0001	0.00000	0.9256	6
0.675294	0.813556	565.184	435.705	435.391	5.199552	-13.8721	9.253198	0.0003	0.0002	0.0002	0.00003	0.8860	28
0.144715	0.270772	610.681	390.154	389.984	0.400498	4.772644	-9.499598	0.0001	0.0000	0.0000	0.00001	0.9055	6
0.753308	0.887874	616.630	384.009	384.940	4.652703	-12.5434	8.450933	0.0001	0.0001	0.0002	0.00002	0.9040	32
0.908448	0.957071	595.380	404.479	405.128	-4.055715	6.486184	-1.952416	0.0002	0.0001	0.0002	0.00001	0.9255	21
0.220638	0.529526	637.686	363.275	362.487	0.685409	2.952123	-6.676628	0.0001	0.0000	0.0001	0.00002	0.9854	32
0.590939	0.615211	597.901	402.838	402.149	-0.875622	6.102896	-7.165173	0.0000	0.0002	0.0001	0.00002	0.9498	6
0.476973	0.494184	546.631	452.399	451.762	1.321259	-0.04879	-3.190553	0.0001	0.0002	0.0001	0.00003	0.9644	13
0.429482	0.689566	519.450	480.683	481.642	2.907427	-6.48469	3.3197	0.0001	0.0002	0.0003	0.00001	0.9254	26
0.001792	0.285244	502.017	497.264	497.112	0.479286	3.928016	-7.350406	0.0001	0.0000	0.0002	0.00002	0.8668	16
0.434314	0.989304	614.745	384.941	384.918	4.518964	-12.053	8.060292	0.0001	0.0001	0.0002	0.00003	0.9397	18
0.810792	0.923059	511.993	488.662	488.867	2.549828	-7.67451	5.631641	0.0000	0.0001	0.0002	0.00000	0.9816	7
0.386302	0.975958	556.146	443.034	442.205	4.26313	-11.4037	7.658942	0.0002	0.0002	0.0002	0.00001	0.8891	4
0.105782	0.54728	599.462	401.290	401.297	0.47778	4.089679	-8.149012	0.0002	0.0000	0.0002	0.00003	0.9141	34
0.038944	0.572351	547.421	451.933	451.841	0.46726	4.11836	-8.122994	0.0002	0.0001	0.0003	0.00000	0.9989	12
0.242885	0.454341	625.044	374.427	375.312	0.548812	3.80329	-7.960753	0.0000	0.0001	0.0002	0.00001	0.9415	9
0.884585	0.896512	581.338	417.682	416.684	-2.485004	3.577624	-0.653736	0.0002	0.0002	0.0002	0.00001	0.9676	2
0.008574	0.107252	539.416	461.281	461.744	0.49494	3.389111	-3.951048	0.0001	0.0001	0.0002	0.00001	0.9781	16
0.59412	0.841014	620.532	379.419	378.934	5.044566	-13.4136	8.920132	0.0002	0.0001	0.0002	0.00003	0.9960	18
0.482966	0.912977	616.104	384.263	385.009	4.786066	-12.7864	8.550762	0.0001	0.0001	0.0000	0.00001	0.9445	33
0.21136	0.553078	588.525	411.761	412.337	0.717369	2.735222	-6.337924	0.0001	0.0003	0.0002	0.00002	0.9211	8
0.769896	0.847555	566.054	434.285	434.793	2.344688	-6.86859	4.962867	0.0002	0.0001	0.0002	0.00000	0.9155	3
0.41739	0.500036	577.489	422.148	422.633	4.848799	-15.4056	13.485216	0.0001	0.0000	0.0002	0.00003	0.9217	15
0.295148	0.352	597.111	403.392	404.261	0.387123	4.873423	-9.704807	0.0001	0.0001	0.0000	0.00003	0.8977	23
0.378401	0.93028	535.291	463.994	464.410	4.047738	-10.7379	7.162291	0.0003	0.0001	0.0001	0.00003	0.8694	29
0.316721	0.991723	552.987	446.608	446.387	3.739284	-9.98436	6.73005	0.0001	0.0003	0.0001	0.00001	0.9061	8
0.403481	0.80944	541.112	458.978	459.369	3.566391	-8.98479	5.635987	0.0002	0.0001	0.0001	0.00001	0.9364	31
0.356699	0.986483	542.287	456.866	456.244	4.085368	-10.9451	7.369765	0.0000	0.0003	0.0001	0.00002	0.9090	24
0.120854	0.638617	551.996	448.509	448.036	0.660247	2.776901	-6.040679	0.0002	0.0002	0.0000	0.00001	0.9108	35
0.382687	0.412435	584.311	415.898	415.093	1.397067	-0.47681	-2.57908	0.0000	0.0001	0.0002	0.00001	0.9675	1
0.630326	0.984551	556.056	443.997	444.666	4.810479	-12.8266	8.55884	0.0002	0.0001	0.0001	0.00001	0.9813	13
0.471696	0.9234	582.284	417.544	417.768	4.760066	-12.7141	8.502911	0.0002	0.0001	0.0001	0.00000	0.9128	31
0.140224	0.353918	577.498	423.425	424.000	0.405069	4.754791	-9.51792	0.0000	0.0003	0.0001	0.00002	0.8532	36
0.540241	0.645817	607.457	392.884	391.971	3.705055	-9.16138	5.55151	0.0002	0.0003	0.0002	0.00001	0.9645	24
0.248794	0.332006	611.469	388.959	389.148	0.471754	4.298072	-8.729586	0.0002	0.0002	0.0002	0.00003	0.9836	4
0.086551	0.407214	575.873	424.827	425.413	0.420461	4.625187	-9.249066	0.0001	0.0000	0.0002	0.00001	0.9079	19
0.816528	0.901481	619.306	380.119	379.261	4.270738	-11.6626	7.94208	0.0001	0.0003	0.0003	0.00002	0.9962	29
0.200032	0.314307	591.956	408.027	408.300	0.416822	4.701121	-9.464894	0.0003	0.0003	0.0003	0.00002	0.9158	29
0.345596	0.566242	527.571	472.094	472.440	1.420445	-0.52537	-2.617408	0.0001	0.0002	0.0001	0.00002	0.9697	35
0.347297	0.562507	590.386	408.891	409.880	1.405706	-0.4676	-2.673811	0.0002	0.0001	0.0000	0.00002	0.9226	5
0.31763	0.524651	498.064	502.112	501.684	1.098693	0.95305	-4.299129	0.0001	0.0001	0.0001	0.00002	0.9342	38
0.177281	0.213103	582.993	417.427	416.593	0.675133	1.941115	-2.197091	0.0003	0.0000	0.0000	0.00002	0.9527	33
0.478104	0.686496	568.397	432.095	432.336	3.423344	-8.2471	4.812014	0.0002	0.0001	0.0002	0.00001	0.8610	20
0.45901	0.945764	519.999	479.939	479.635	4.631968	-12.3687	8.273945	0.0000	0.0003	0.0002	0.00000	0.9122	18

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.

References

1. L.Bull, A.Sha'Aban,A. Tomlinson,J. Addison, and B.Heydecker,"Towards distributed adaptive control for road traffic junction signals using learning classifier systems", in Applications of LCS,Studies in fuzziness and soft computing,2004, pp. 276–299.
2. M.V.Butz,"An algorithmic description of ACS2", inLanzi PL, Stolzmann W, Wilson SW (eds) advances in learning classifier systems. LNAI,2002, Vol. 2321, pp. 211–229
3. M. V.Butz"Kernel-based, ellipsoidal conditions in the real valuedXCS classifier system", In Beyer HG, O'Reilly UM (eds) Genetic and evolutionary computation conference, GECCO, 2005. pp. 1835–1842.
4. M. V. Butz, Rule-based evolutionary online learning systems, Berlin: Springer,2006.
5. M. V. Butz,P. L.Lanzi, andS. W. Wilson,"Hyper-ellipsoidal conditions in XCS: rotation, linear approximation, and solution structure", inGECCO, 2006, Vol. 8, pp. 1457–1464.
6. M. V. Butz,andS. W. Wilson,"An algorithmic description of XCS", inAdvances in learning classifier systems, LNAI, 2001, Vol. 1996,pp. 253–272.
7. M. Dorigo,"Alecsys and the autonomouse: learning to control a real robot by distributed classifier systems", in Mach Learn,1995, Vol. 19, pp. 209–240.
8. M. Dorigo, andM.Colombetti,"Robot shaping: an experiment in behavior engineering",Massachusetts: MIT Press/Bradford Books, 1998.
9. M.Dorigo, andU.Schnepf,"Genetics-based machine learning and behavior based robotics" in a new synthesis. IEEE Trans Syst Man Cybern, 1993, Vol. 23, pp. 141–154.
10. A. E. Eiben and J. E. Smith, Introduction to Evolutionary Computing, Springer, 2003.
11. P. L.Lanzi,"Learning classifier systems: then and now". Evol. Springer,2008,pp. 63–82.
12. P. L.Lanzi,D.Loiacono,S. W. Wilson,D. E. Goldberg, "Prediction update algorithms for XCSF: Rls, kalman filter, and gain adaptation", in GECCO, Vol. 8, 2006, pp. 1505–1512.
13. P. L.Lanzi,D.Loiacono,S. W. Wilson,D. E. Goldberg,"Generalization in the XCSF classifier system: analysis, improvement, and extension",EvolComput J,Vol. 15, No.2,2007, pp. 133–168.
14. D. Loiacono, andP. L.Lanzi,"XCSF with neural prediction",in IEEE congress on evolutionary computation. CEC 2006, pp. 2270–2276.
15. D.Loiacono,A.Marelli, andP. L.Lanzi, "Support vector regression for classifier prediction",in GECCO,2007, Vol. 2, pp. 1806–1813.
16. W.Stolzmann,and M. V.Butz, "Latent learning and action planning in robots with anticipatory classifier systems", inlearning classifier systems, from foundations to applications, Lecture notes in computer science,2000,Vol. 1813, pp. 301–320.
17. R. S.Sutton, andA.G.Barto, "Reinforcement learningan introduction",Cambridge: MIT Press, 1998.
18. T. H.Tran,C.Sanza,Y.Duthen, andT. D. Nguyen, "XCSF with computed continuous action", inGECCO, 2007, pp. 1861–1869.
19. B.Widrow, and M. E. Hoff."Adaptive Switching Circuits", Chapter Neurocomputing*:* Foundation of Research,Cambridge: The MIT Press, pp. 126-134, 1998.
20. S. W.Wilson, "Classifier fitness based on accuracy",EvolComputVol.3, No.2, pp.149–175,1995.
21. S. W.Wilson, "Get real! XCS with continuous-valued inputs", in Lanzi PL, Stolzmann W, Wilson SW (eds) Learning classifier systems, from foundations to applications, Lecture notes in computer science,2000, Vol. 1813, pp. 209–222.
22. S. W. Wilson,"Function approximation with a classifier system",in GECCO, 2001, pp. 974–981.
23. S. W. Wilson, "Classifiers that approximate functions". J Nat Compute, Vol. 1, No. 2,2002, pp. 211–234.
24. S. W. Wilson,"Classifier Systems for Continuous Payoff Environments",in GECCO, 2004, pp. 824-835 in Part II.
25. S. W. Wilson,"Three architectures for continuous action", in :Kovacs T, Llora` X, Takadama K, Lanzi PL, Stolzmann W, Wilson SW (eds) IWLCS, Lecture notes in computer science,2005, Vol. 4399, pp. 239–257.

S. Goodarzian was born in Shiraz, Iran in 1984. He received his B.Sc. degree in Computer engineering from Shiraz Islamic Azad University in 2008. He is received his M.Sc. degree in Artificial Intelligence at Shiraz University in 2011. His research interests include evolutionary computation and learning classifier systems.

A. Hamzeh received his Ph.D. in artificial intelligence from Iran University of Science and Technology in 2007. Since then, he has been working as assistant professor in CSE and IT Department of Shiraz University. His research interests include evolutionary computation, optimization and learning classifier systems.

S. Hashemi received the PhD degree in Computer Engineering from the Iran University of Science and Technology, in conjunction with Monash University, Australia, in 2008. He is currently a lecturer in the Electrical and Computer Engineering School, Shiraz University, Shiraz, Iran. His research interests include data stream mining, database intrusion detection, dimension reduction, and adversarial learning.

Copyright (c) 2012 International Journal of Computer Science Issues. All Rights Reserved.