XCSF with Computed Continuous Action

Trung Hau Tran, Cédric Sanza, Yves Duthen
IRIT
Paul Sabatier University
Toulouse, France
hau@irit.fr

Dinh Thuc Nguyen
Faculty of Information Technology
University of Natural Sciences
HoChiMinh City, Vietnam

ABSTRACT
Wilson introduced XCSF as a successor to XCS. The major development of XCSF is the concept of a computed prediction. The efficiency of XCSF in dealing with numerical input and continuous payoff has been demonstrated. However, the possible actions must always be determined in advance. Yet domains such as robot control require numerical actions, so that neither XCS nor XCSF with their discrete actions can yield high performance. This paper studies computed action in XCSF, where the action is continuous with respect to the input state. In comparison with Wilson's architecture for continuous action, our XCSF version, called XCSFCA, proves to be more efficient.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*.

General Terms
Algorithms.

Keywords
XCSF, continuous actions, evolution strategy.

1. INTRODUCTION
Research on learning classifier systems has focused on the use of accuracy-based fitness after the introduction of XCS [11], in which (1) the classifier fitness is based on the accuracy of the prediction and (2) general and accurate classifiers are maximally obtained. The payoff is a discontinuous function of the state x. The predictions are estimated values representing the expected payoffs if the system matches x and activates an action. The system builds a complete map of predictions *Condition x Action* $\Rightarrow$ *Prediction*. However, when the number of states and actions increases, it is clear that the system cannot construct such map of predictions. This is not only due to the memory allocated by the system, but also the time as well as the examples used for training the system. The system should generalize from a subset of data to approximate a payoff function as the study of the generalization in reinforcement learning [7]. Wilson recently introduced a new learning classifier system XCSF [13][14][15] that differs from XCS in real value inputs for the condition part and the new concept of a computed classifier prediction. XCSF learns how to construct a function approximator which approximates a payoff function. It can be a discontinuous 0/1000 payoff function or a continuous payoff function of real inputs. When XCSF receives the input x, it builds a *system prediction* for each discrete action $\hat{P}(x,a)$ via the function approximator and chooses the action a^* with the highest prediction $\hat{P}(x,a^*)$ to maximize the payoff. Some XCSF versions adapting to different types of payoff functions are presented in [4][13][14][15].

In traditional learning classifier systems, the possible actions must always be determined in advance. An animat in *woods* problems [10][11] can take one of eight actions corresponding to eight directions (*go north, north-east, east, south-east*…). One can use three bits to encode these eight actions in the action part. However, when dealing with problems that require real value outputs, the systems show some limits. In the example of an animat which could move in any direction represented by an angle from $[0, 2\pi]$, the solution is to increase the number of bits encoding actions in order to represent real angles. This one is feasible but we might lose the reactive attribute in the learning classifier systems. Indeed, the population size of classifiers, the time of building a match set and the convergence time of the system increase. Another solution is to use a generalized classifier system (GCS) [16], in which the input x is linked with the action a in the computation of the prediction as well as in the process of building the match set M. Each classifier employs the form of $t(x,a):w \Rightarrow p(x,a)$ where $t(x,a)$ is the condition form allowing the classifier to be or not to be a member of M. The prediction weight vector w extends a new weight associated with the action a and the prediction $p(x,a)$ is computed linearly with x and a. The difficulty of GCS is to evolve a condition form $t(x,a)$ so that the action is continuous. Wilson stated that if $t(x,a)$ employs interval predicates, the best action a^* is discontinuous. An example of GCS with orientable elliptical form $t(x,a)$ is applied to the frog problem.

In this paper, we will investigate a new approach in which the action is computed directly. Thus, the action is continuous with respect to the input state. Basing on XCSF with a computed classifier prediction $\hat{P}(x,a)$, we extend a computed classifier action to XCSF. The classifier action is computed as a linear combination of the input and a vector of action weights. The

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

action weights are themselves adapted to the problem by using an evolution strategy. We apply our XCSF version, called XCSFCA, to the frog problem described in [15][16]. The results show that XCSFCA obtains better performances than GCS. In addition, the population size of classifiers is more compact.

The next section describes briefly XCSF. Section 3 presents XCSFCA, its application in the frog problem and the results. Section 4 concludes this paper.

2. 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 form 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 accurate values of the prediction. Thus, the system performance improves, 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_{win} between possible actions in M according to an action selection strategy, forms an action set A of the classifiers advocating a_{win}, and sends a_{win} 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 [11] or M contains less than ϕ_{mna} discrete actions as in [3]. 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. $l_i \le x_i \le u_i$. Fortunately, the system with the real input was studied in [12] 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 [14] proposed that the value of x_0 should be the same order of the values of the elements of x.

$$\begin{aligned} cl.p(x) &= w \cdot x' \\ w &= (w_0, w_1, ..., w_n) \\ x' &= (x_0, x_1, ..., x_n) \end{aligned} \quad (1)$$

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) permutes 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 intervals out of the condition range. If it happens, l_i and u_i of an interval predicate int_i are brought back to the extremes of the condition range 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* [9]:

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

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

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

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta(|P - cl.p(x)| - cl.\varepsilon) \quad (4)$$

$$cl.\kappa = \begin{cases} 1 & \text{if } cl.\varepsilon < \varepsilon_0 \\ \alpha \left(\frac{cl.\varepsilon}{\varepsilon_0} \right)^{-\nu} & \text{otherwise} \end{cases} \quad (5)$$

$$cl.\kappa' = \frac{cl.\kappa \times cl.num}{\sum_{j \in [A]} cl_j.\kappa \times cl_j.num} \quad (6)$$

$$cl.fit \leftarrow cl.fit + \beta(cl.\kappa' - cl.fit) \quad (7)$$

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, so $cl.\varepsilon$ and $cl.fit$ take into account the change of $cl.w$.

3. EXTENSION TO XCSF

3.1 XCSF with Computed Continuous Action

As introduced in the first section, XCSF with discrete actions performs inefficiently when facing problems requiring continuous real values. We propose an extension to XCSF where actions are directly computed. The actions are then a continuous function of the input. The modified classifier system is called XCSFCA.

A vector of action weights $\zeta=(\zeta_0,\zeta_1,...,\zeta_n)$ is associated to each classifier action. The classifier action $cl.a$ is computed as a linear combination of ζ and the current input $x=(x_1,...,x_n)$ concatenated with a constant x_0.

$$cl.a(x,\zeta) = \zeta \cdot x'$$
$$x' = (x_0, x_1, ..., x_n) \quad (8)$$

A separate process called an evolution strategy (ES) evolves the action weights to make the computed actions more accurate. The ES was developed by Schwefel [6] for optimizing real-vectors. Each individual in the ES consists of a pair of two real-vectors (ζ,σ) having the same length n_σ: a vector of action weights ζ and a vector of standard deviations σ used by mutation. The ES, which we use to evolve the action weights, is (1+1)-ES, in which one parent produces one offspring 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 best individual becomes the parent of the next generation. Mutation on ζ is performed by adding a random value $N(0,\sigma_i)$ from a normal distribution with mean zero and standard deviation σ_i. The standard deviation vector is also itself adapted.

$$\sigma_i' = \sigma_i exp(\tau' N(0,1) + \tau N_i(0,1)) \quad (9)$$
$$\zeta_i' = \zeta_i + N(0, \sigma_i') \quad (10)$$
$$\tau' = \left(\sqrt{2n_\sigma}\right)^{-1} \quad (11)$$
$$\tau = \left(\sqrt{2\sqrt{n_\sigma}}\right)^{-1} \quad (12)$$

Where (ζ',σ') is an individual mutant, n_σ is the length of the vector σ, and the factors τ' and τ are set by default. The normal distribution with mean zero and standard deviation one $N_i(0,1)$ is called anew for each value of i=0,…,(n_σ-1). Each element ζ_i is initialized to a value uniform randomly from [-1,1] and each element σ_i is initialized to a constant value $\sigma^{(0)}$.

XCSFCA works like XCSF with a computed classifier prediction $\hat{P}(x,a)$, except for the following cases: (1) the process of building the match set *M*, covering operator, action selection in exploration and crossover operator are modified, and (2) a new component updating action weights is introduced.

Match set *M*. A classifier will become a member of *M* if its condition part matches the input as usual and its computed action *cl.a* belongs to the range of the action a_{range}. The ranges of action weights are not given. The action weights are initially generated randomly and then updated by mutation. So, a classifier can compute *bad* actions out of a_{range} in response to the input. We modified the process of building *M* so that *M* consists of matching classifiers advocating *good* actions, which belong to a_{range}, and thus they will have a chance to correct their action weights.

Covering operator. Given the input *x*, a vector of action weights *cl.*ζ is repeatedly generated until an action *cl.a* computed by Eq.(8) belongs to the action range allowed. This ensures that the match set *M* has at least one *good* action in response to the input. Interval predicates $cl.int_i$ are created as in XCSF.

Action selection in exploration. A pure random exploration is usually used for exploring actions of the environment. We propose to replace it by a deterministic strategy, i.e. the action with the highest prediction is selected. The classifiers advocating the best action will survive while an ES will allow the system to explore actions.

Crossover operator. The recombination of alleles of two parents does not generate new values of alleles. The system cannot explore values near the parents' values, and thus appropriate conditions are not generated. To solve this problem, we use a *mix* crossover [5]. The allele value v_i^o of an offspring is the result of a *mix* function of two allele values v_i^1 and v_i^2 of parents at the same position *i*.

$$v_i^o = (1-\gamma_{GA})v_i^1 + \gamma_{GA} v_i^2 \quad (13)$$
$$\gamma_{GA} = (1+2\alpha_{GA})u - \alpha_{GA} \quad (14)$$

Where α_{GA} is a real constant from [0,1] and *u* is a random value from [0,1]. When α_{GA}=0, the *mix* crossover is a kind of crossover that interpolates two parents' values. Notice that XCSFCA still has the XCSF mutation operator.

Action weights update component. An (1+1)-ES is applied on each member of action set *A* to evolve actions during explore problems. Indeed, it will mutate the current action of each classifier of *A* to produce one mutant action and evaluate it. Accurate actions will be kept to the next time-step. Consider a classifier *cl* in *A* with its action weights *cl.*ζ, its standard deviations *cl.*σ, its error *cl.*ε, its experience *cl.exp*, its numerosity *cl.num* and its fitness *cl.fit*. The mutant classifier *cl'* is a copy of the parent classifier *cl*. It inherits the parameters as *cl.*ε, *cl.fit*, *cl.exp* and *cl.num* of the parent classifier *cl* except for its action represented by $(cl'.\zeta, cl'.\sigma)$ generated by mutation on $(cl.\zeta, cl.\sigma)$. This corresponds to step 1 (Algorithm 1). The evaluation of the mutant classifier's fitness works as follows. The mutant action represented by $(cl'.\zeta, cl'.\sigma)$ is calculated. The mutant classifier *cl'* computes its prediction and is tested in the context of the current payoff *P*. Then its prediction error *cl'.*ε is estimated and its accuracy value *cl'.*κ is calculated. Finally, its fitness *cl'.fit* is updated by its relative accuracy value. These correspond to step 2 (Algorithm 1). If the mutant is accurate and better than the parent in the fitness sense, it will replace its parent (step 3 in Algorithm 1). If replacement doesn't occur, the mutant is discarded and a new mutant will be generated in next problems. This update process (Algorithm 1) is called after the reinforcement component.

Algorithm 1. Update action weights by (1+1)-ES.

1. Mutate one parent action represented by $(cl.\zeta, cl.\sigma)$ to produce one mutant action $(cl'.\zeta, cl'.\sigma)$ by Eq.(9) and Eq.(10)
2. Evaluate the mutant action $(cl'.\zeta, cl'.\sigma)$

$cl'.a(x, cl'.\zeta)$ *// computed action of the mutant cl'*

$$good = \begin{cases} 1 & \text{if } cl'.a \in \textit{action range allowed} \\ 0 & \text{otherwise} \end{cases}$$

// the mutant cl' computes its prediction cl'.p
// and updates its error cl'.ε *and its fitness cl'.fit*

$$cl'.\varepsilon \leftarrow cl'.\varepsilon + \begin{cases} \dfrac{\left(\left|P - cl'.p(x, cl'.a)\right| - cl'.\varepsilon\right)}{cl'.exp} & \text{if } cl'.exp < \dfrac{1}{\beta} \\ \beta\left(\left|P - cl'.p(x, cl'.a)\right| - cl'.\varepsilon\right) & \text{otherwise} \end{cases}$$

$$cl'.\kappa = \begin{cases} 1 & \text{if } cl'.\varepsilon < \varepsilon_0 \\ \alpha\left(\dfrac{cl'.\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{otherwise} \end{cases} \quad // \textit{ accuracy}$$

$$cl'.fit = \delta_{ES}\, cl.fit \quad // \textit{ reduce inherited fitness by } \delta_{ES}$$

$$cl'.fit \leftarrow cl'.fit + \beta\left(\frac{cl'.\kappa \times cl'.num}{cl.\kappa \times cl.num + cl'.\kappa \times cl'.num} - cl'.fit\right)$$

3. Replace the parent classifier if the mutant classifier is accurate and dominates the parent classifier

if ($good = 1$ and $cl'.\kappa = 1$ and $cl'.fit > cl.fit$)
replace cl by cl′

Where β, ε_0, α, ν are parameters used in XCS and the factor δ_{ES} is used to reduce the mutant's fitness.

Computed classifier prediction. Notice that because the classifier prediction *cl.p* is computed linearly with *x* and *a*, a weight associated with the action *a* is added to the prediction weight vector *cl.w*.

$$\begin{aligned} cl.p(x,a) &= w \cdot x'' \\ w &= (w_0, w_1, ..., w_n, w_{n+1}) \\ x'' &= (x_0, x_1, ..., x_n, a) \end{aligned} \tag{15}$$

In summary, XCSFCA has a couple of computations: one for the action based on the input $a(x)$, the other one for the prediction based on the input and the result of the first computation $p(x, a(x))$.

3.2 Frog Problem

In this section, we summarize the frog problem introduced in [15]. A frog-like system learns to jump to catch a fly that is at a distance *d* far from the frog. The frog receives the signal input *x* related to *d* ($0 \le d \le 1$) and jumps a certain distance. A jump can be over or under the fly. Then the frog receives a payoff related to the remaining distance. Consider a jump as an action *a* that the system should choose. The frog problem is how to choose the best action *a** to maximize the payoff. Mathematically, the payoff function is given as follows:

$$P(x,a) = \begin{cases} x+a & \text{if } x+a \le 1 \\ 2-(x+a) & \text{otherwise} \end{cases} \tag{16}$$

$$x(d) = 1-d \tag{17}$$

The payoff function is continuous and nonlinear composed of two linear planes. To facilitate the reading, we called the frog problem with the payoff function in Eq.(16) *frog1*. *frog2* is a modification of *frog1*. The payoff function is still continuous and nonlinear but composed of two nonlinear forms (Eq.(18)).

$$P(x,a) = \begin{cases} xe^{a} & \text{if } a \le -\ln x \\ x^{-1}e^{-a} & \text{otherwise} \end{cases} \tag{18}$$

$$x(d) = e^{-d} \tag{19}$$

In both of *frog1* and *frog2* problems, the best value of *a** should be continuous with respect to *x* in order to be able to maximize the payoff.

3.3 Simulation Settings

In order to investigate the impact of computed action and facilitate the comparison of XCSFCA with XCSF in [16], the minimal change to parameter settings is made to the latter and the same measures measuring the system performance are used. The parameter settings used in our results were: N=2000, β=0.2, η=0.2, θ_{GA}=48, χ=0.8, μ=0.04, r_0=0.1, m_0=0.1, x_0=1, ε_0=0.01, α=0.1, ν=5, θ_{del}=50, δ=0.1, δ_{ES}=0.1, α_{GA}=0, and initial standard deviation $\sigma^{(0)}$=3. $\sigma^{(0)}$=3 is chosen as the default value proposed by [2]. GA-subsumption and action set subsumption were not activated. Since the input *x* is one-dimensional, the vector length of action weights and standard deviations are n_σ=2. Explore and exploit problems are alternated with probability 0.5. One run is stopped after 100,000 explore problems. The payoff received after jumping is plotted. The system error measures the difference between the prediction of the expected payoff and the payoff received. The payoff and system error curves are plotted by using a 50-point running average from exploit problems, averaged over ten runs. In each problem, the fly was randomly placed at a distance *d* ($0 \le d \le 1$). The action range allowed is [0,1]. At the end of one run, *x* was scanned from 0 to 1 and from e^{-1} to 1 in *frog1* and *frog2* respectively, increased by 0.001, the best action *a** is plotted. The best action curve is plotted, averaged over ten runs. This curve exhibits the continuous actions related to the input.

3.4 Results

Since *d* is generated randomly from [0,1], the sensory input *x* received by Eq.(17) gives the value from [0,1]. However, values of *x* in *frog2* received by Eq.(19) have the range $[e^{-1},1]$, they are scaled into [0,1] when computing the prediction and the action so that the input and the action use the same range. In covering, given *x*, $cl.\zeta$ of a covered classifier is repeatedly generated until an action *cl.a* calculated by Eq.(8) belongs to the action range allowed [0,1].

***frog1*'s results.**

In Fig. 1, the system performance is greater than 99% after an averaged number of 30,000 explore problems. The system error drops to smaller than 1%. The population size of classifiers *N* is to about 37% of *N*. Fig. 2 shows the best action curve which is nearly coincident with the diagonal 1-*x*. The system receives the input *x* (not the distance *d*). If the frog-like system wants to catch the fly at *d*, it must jump a distance *a*=*d*. Since *x* is given by Eq.(17), the best action, which the system needs to obtain, is the diagonal *a*=1-*x*. The averaged number of classifiers building the best action curve of Fig. 2a is 60.8 classifiers. Fig. 3 shows the error of *best action a** of Fig. 2b, which measures the difference between *a** and the diagonal 1-*x*. In Fig. 5, the vectors of action weights (ζ_0,ζ_1) are close to (-1,1), which implies that the best action curve of Fig. 2b is nearly coincident with the diagonal. The standard deviations σ_0 and σ_1 approach zero values and thus the values of ζ_0 and ζ_1 will be little updated by Eq.(10). For example, in the last classifier in Fig. 5, given x = 0.999, its action *cl.a* and its prediction *cl.p* are computed as follows: $cl.a = \zeta_0 x_0 + \zeta_1 x$ = 1.0006 – 1.0006*0.999 = 0.001 and $cl.p = w_0 x_0 + w_1 x + w_2 a$ =

670.17 + 329.84*0.999 + 329.38*0.001 = 1000.0. The results showed here are better than the ones of GCS [16] (Fig. 4 and Fig. 6).

The convergence speed is not fast because action weights are updated by mutation by small changes from one generation to another in the evolution strategy. In contrast to the *delta rule* method, the weights are updated by adding small changes in the direction that would reduce the error. Because of only one payoff used for evaluating one individual mutant action, the system should have strict replacement conditions to ensure that replacement will occur if the mutant is significantly better than its parent. This means a poor mutant could replace the effective classifier by mistake and then cause the system performance to be sometimes volatile. So, a good insurance is to make a reduction of the mutant's fitness by the factor δ_{ES} before updating it and use the accurate mutant in the replacement.

XCSFCA evolves thanks to the loop "reinforcement – ES – GA". This means a GA generates offspring that inherit the parameters of the parents, a reinforcement procedure corrects their parameters in the context of the current payoff, and their vectors of action weights will be updated through an ES procedure in order to produce actions more accurate. The GA will be triggered to generate new offspring. The loop of "reinforcement – ES – GA" continues.

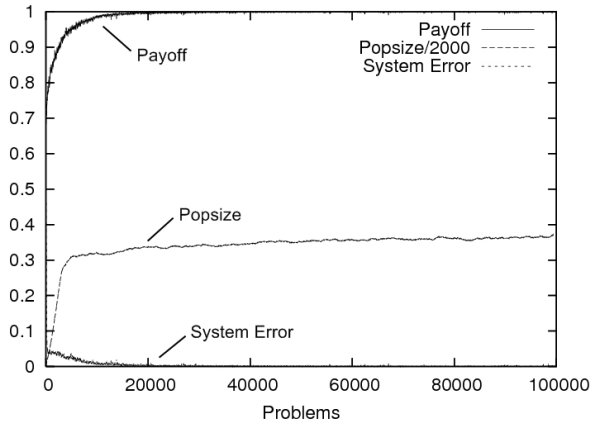
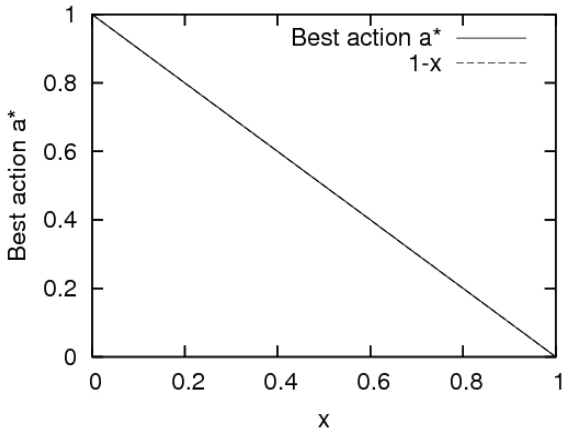
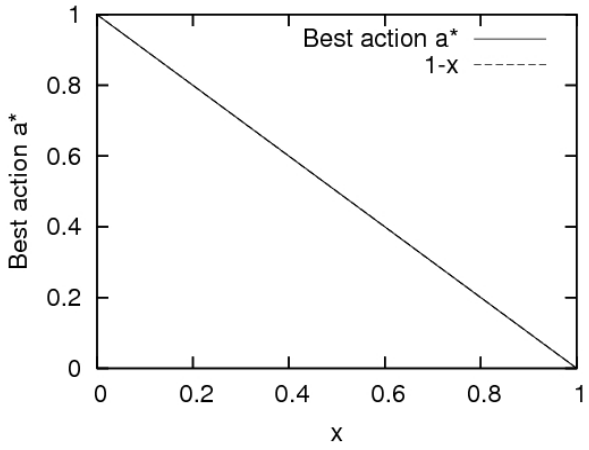


Figure 1: Results of *frog1* averaged over ten runs.



(2a) averaged over ten runs



(2b) from one run

Figure 2: *Best action a of *frog1* is plotted by scanning the values of *x* from 0 to 1, increased by 0.001. The diagonal 1-*x* and *best action a** are nearly coincident. *Best action a** is slightly broken at some inputs *x*.**

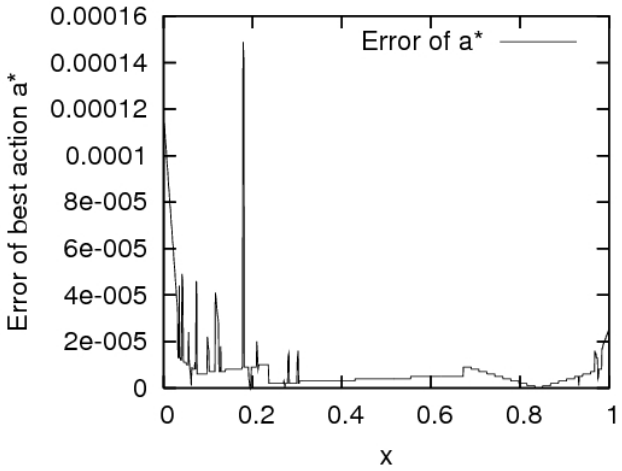


Figure 3: Error of *best action a of Fig. 2b. The vertical axis is scaled to [0..0.00016] to make visible.**

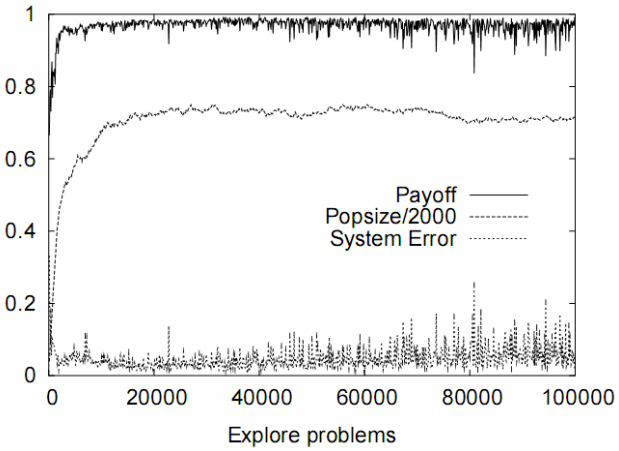


Figure 4: Results of *frog1* from one run in [16].

l_0	u_0	w_0	w_1	w_2	a	ζ_0	ζ_1	σ_0	σ_1	p	ε	fit	num
0.000	0.893	669.33	327.70	330.69	0.108	1.0000	-0.9995	0.0002	0.0009	997.5060	0.0000	1.0000	10.00
0.098	0.654	669.47	327.70	330.83	0.346	1.0002	-1.0022	0.0000	0.0001	997.8604	0.0000	0.9966	2.00
0.278	0.951	669.83	327.77	331.11	0.048	1.0005	-1.0018	0.0000	0.0000	997.3538	0.0000	0.5847	2.00
0.235	0.991	670.31	329.68	329.69	0.009	1.0000	-1.0000	0.0001	0.0016	999.9920	0.0000	1.0000	50.00
0.209	0.577	669.46	327.04	331.48	0.422	1.0009	-1.0040	0.0000	0.0000	997.8859	0.0000	0.9972	9.00
0.070	0.582	669.42	327.70	330.78	0.419	1.0001	-1.0010	0.0000	0.0009	998.2709	0.0000	0.4624	2.00
0.232	0.933	670.18	329.85	329.38	0.068	1.0005	-1.0006	0.0000	0.0000	999.9981	0.0000	0.3147	6.00
0.189	0.650	669.64	327.73	330.97	0.350	1.0003	-1.0014	0.0000	0.0000	998.2962	0.0000	0.6973	5.00
0.056	0.566	669.41	327.70	330.77	0.434	1.0001	-1.0017	0.0000	0.0003	998.1680	0.0000	0.9975	3.00
0.056	0.618	669.41	327.70	330.77	0.381	1.0001	-1.0021	0.0001	0.0004	997.8939	0.0000	0.9892	21.00
0.070	0.586	669.43	327.70	330.79	0.413	1.0001	-1.0015	0.0000	0.0001	998.1676	0.0000	0.7966	8.00
0.115	0.738	669.51	327.71	330.86	0.262	1.0003	-1.0020	0.0000	0.0001	997.6579	0.0000	0.5435	4.00
0.270	0.762	669.81	327.76	331.10	0.237	1.0005	-1.0017	0.0000	0.0001	998.0886	0.0000	0.7509	9.00
0.299	0.970	669.87	327.79	331.14	0.030	1.0005	-1.0017	0.0000	0.0000	997.3851	0.0000	0.4274	3.00
0.035	0.952	669.36	327.69	330.72	0.045	1.0001	-1.0030	0.0001	0.0006	996.3081	0.0000	0.2863	2.00
0.056	0.582	669.41	327.70	330.77	0.418	1.0001	-1.0015	0.0000	0.0001	998.1341	0.0000	0.5594	4.00
0.127	0.705	669.53	327.71	330.88	0.295	1.0002	-1.0021	0.0000	0.0001	997.7748	0.0000	0.8111	8.00
0.191	0.552	669.65	327.73	330.98	0.448	1.0003	-1.0014	0.0000	0.0000	998.6774	0.0000	0.7074	10.00
0.151	0.991	670.31	329.68	329.68	0.009	1.0000	-1.0000	0.0001	0.0006	999.9912	0.0000	0.4019	17.00
0.072	0.991	670.31	329.67	329.69	0.009	1.0000	-1.0000	0.0001	0.0006	999.9897	0.0000	0.7817	27.00
0.229	0.976	670.17	329.85	329.38	0.024	1.0006	-1.0006	0.0000	0.0000	999.9995	0.0000	0.3594	4.00
0.040	0.539	669.37	327.70	330.73	0.461	1.0001	-1.0015	0.0000	0.0002	998.2476	0.0000	0.8092	9.00
0.178	0.573	669.35	327.02	331.39	0.427	1.0009	-1.0041	0.0000	0.0000	997.7532	0.0001	0.6971	6.00
0.308	0.994	670.17	329.85	329.38	0.007	1.0006	-1.0006	0.0000	0.0000	1000.0013	0.0000	0.5493	5.00
0.032	0.991	670.30	329.67	329.68	0.009	1.0000	-1.0002	0.0000	0.0005	999.9342	0.0000	0.8500	12.00
0.229	0.975	670.17	329.84	329.38	0.026	1.0006	-1.0006	0.0000	0.0000	1000.0002	0.0000	0.2828	1.00
0.235	0.988	670.31	329.68	329.69	0.012	1.0000	-0.9999	0.0001	0.0004	999.9987	0.0000	0.6848	9.00
0.000	0.943	669.34	327.69	330.70	0.055	1.0001	-1.0027	0.0001	0.0015	996.4049	0.0001	0.2902	3.00
0.343	1.000	670.17	329.84	329.38	0.001	1.0006	-1.0006	0.0000	0.0000	1000.0005	0.0000	0.0501	1.00

Figure 5: Activated classifiers from one run of *frog1* of Fig. 2b. Values of the action and prediction columns are the values computed from the last input. $int_0=(l_0,u_0)$ is the interval predicate; w_0, w_1 and w_2 are the prediction weights for the constant x_0, the input x and the action a, respectively; ζ_0 and ζ_1 are the action weights for x_0 and x, respectively; σ_0 and σ_1 are the standard deviations used by mutation on ζ_0 and ζ_1, respectively; a, p, ε, *fit* and *num* indicate action, prediction, prediction error, fitness and numerosity, respectively.

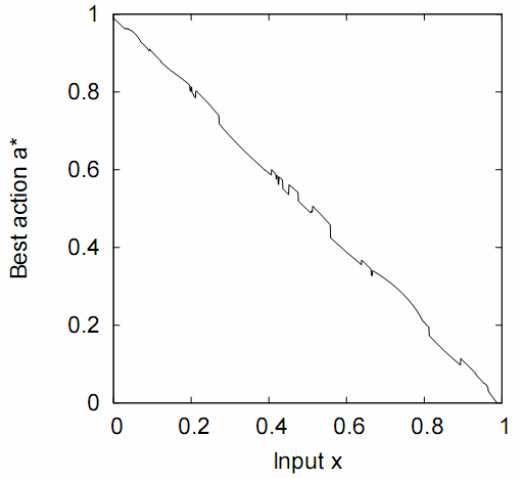


Figure 6: *Best action a of *frog1* from one run in [16].**

***frog2*'s results.**

frog2 is more difficult than *frog1* because the form of its payoff function is composed of two nonlinear tent-like forms on each side. But the system performance in Fig. 7 is still greater than 99%. The system error drops to about 1%. The population size is to about 37% of *N*. The best jump (action), which the system needs to obtain, is *a=-lnx*. The averaged number of classifiers building the best action curve of Fig. 8a is 73.6 classifiers. Fig. 9 shows the error of *best action a** of Fig. 8b. Fig. 10 shows activated classifiers with their main parameters that participated in the creation of *best action a** of Fig. 8b.

4. CONCLUSION

This paper presented an extension to XCSF, in which the action is computed directly as a linear combination of the input state and a vector of action weights. Thus, it is continuous when the input changes. The action weights are themselves adapted to the problem by using a separate process called an evolution strategy. The pure random exploration has been replaced by a deterministic strategy while the exploration of actions was guaranteed by the evolution strategy. XCSFCA has a couple of computations: one for the action and the other one for the prediction. We tested the performance of XCSFCA in the frog problems and the results proved to be more efficient than Wilson's architecture for continuous action.

XCSFCA needs more work. We proposed Algorithm 1 based on the evolution strategy to update action weights, although other methods can perform this update more efficiently. Thus, their integration into XCSFCA might improve the system performance further.

We will try to apply this new system to the prey – predator problem [8] to evolve weights controlling motor schemas [1]. This will be a promising solution to improve the learning capability of the learning predator further.

5. ACKNOWLEDGEMENTS

The authors would like to thank Stewart W. Wilson and the other anonymous reviewers for constructive comments on the submitted paper.

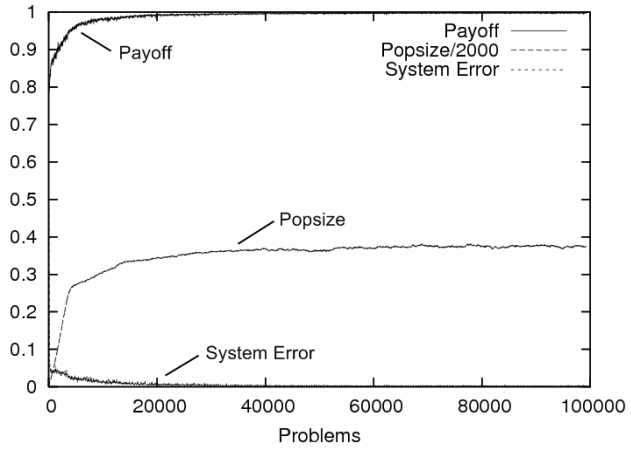


Figure 7: Results of *frog2* averaged over ten runs.

6. REFERENCES

[1] Arkin, R. C., Behavior-Based Robot Navigation for Extended Domains, *Adaptive Behavior*, 1(2):201–225, 1992.

[2] Bäck, T., *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.

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

[4] Lanzi, P. L., Loiacono, D., Wilson, S. W., and Goldberg, D. E., XCS with Computed Prediction for the Learning of Boolean Functions, *Proceedings of the IEEE Congress on Evolutionary Computation Conference (CEC2005)*, 2005.

[5] Sanchez, S., *Mécanismes Evolutionnistes pour la Simulation Comportementale d'Acteurs Virtuels*, PhD thesis, pp. 93-94, University Toulouse I, IRIT (Institut de Recherche en Informatique de Toulouse), France, 2004.

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

[7] Sutton, R. S., Barto, A. G., Reinforcement Learning: An Introduction, MIT Press, 1998.

[8] Tran, T. H., Sanza, C., and Duthen, Y., Evolving the Motor Schema Approach for Learning Cooperation, *CASA'2006 The nineteenth Conference on Computer Animation and Social Agents*, pp. 219-228, Genève, Suisse, 2006.

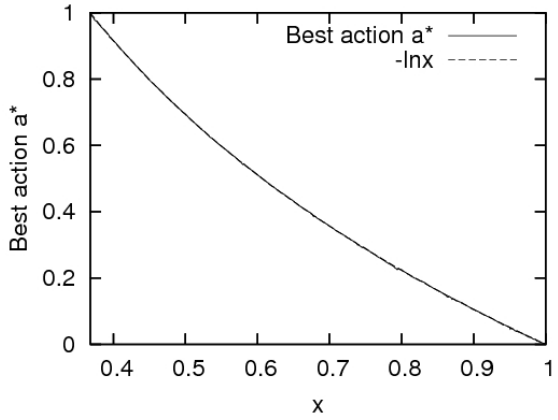
[9] Widrow, B., and Hoff, M. E., Adaptive Switching Circuits, In James A. Anderson and Edward Rosenfeld, editors, *Neurocomputing: Foundations of Research*, pp. 126-134, The MIT Press, Cambridge, MA, 1988.

[10] Wilson, S. W., ZCS: A Zeroth Level Classifier System, *Evolutionary Computation, 2*(1), pp. 1-18, 1994.

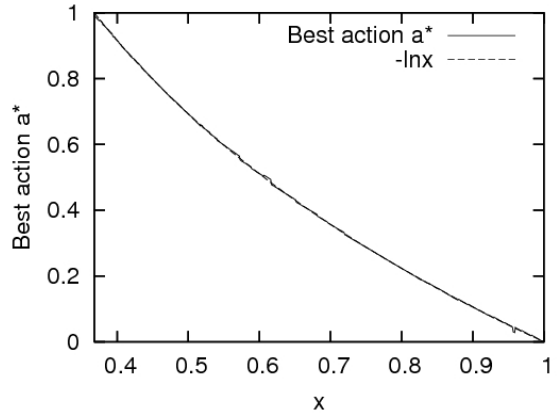
[11] Wilson, S. W., Classifier Fitness Based on Accuracy, *Evolutionary Computation*, 3(2), pp. 149-175, 1995.

[12] Wilson, S. W., Get Real! XCS with Continuous-Valued Inputs, From *Festschrift in Honor of John H. Holland*, L. Booker, S. Forrest, M. Mitchell, and R. Riolo (eds.), pp. 111-121, 1999.

[13] Wilson, S. W., Function Approximation with a Classifier System, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. Spector et al, eds. Morgan Kaufmann, San Francisco, CA, pp. 974-981, 2001.



(8a) averaged over ten runs



(8b) from one run

Figure 8: *Best action a of *frog2* is plotted by scanning the values of *x* from e^{-1} to 1, increased by 0.001. *Best action a** is close to the curve *–lnx*.**

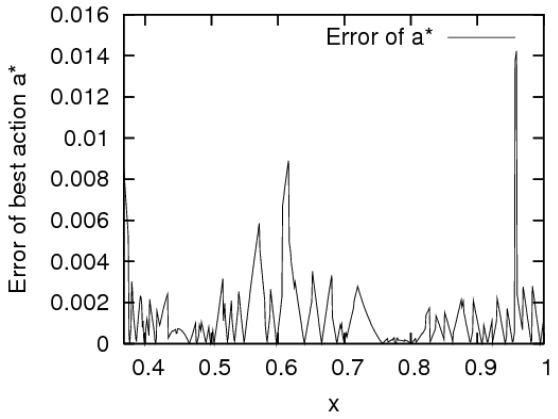


Figure 9: Error of *best action a of Fig. 8b. The vertical axis is scaled to [0..0.0016] to make visible.**

l_0	u_0	w_0	w_1	w_2	a	ζ_0	ζ_1	σ_0	σ_1	p	ε	fit	num
0.588	0.778	658.51	369.18	406.11	0.251	0.7760	-0.8091	0.0000	0.0001	999.9847	0.0000	0.9912	5.00
0.473	0.968	644.66	358.59	402.27	0.033	0.7837	-0.7905	0.0030	0.0016	998.4391	0.0004	0.6286	5.00
0.368	0.876	652.64	365.92	402.95	0.130	0.7819	-0.8107	0.0002	0.0019	999.2504	0.0008	0.7540	3.00
0.548	0.850	654.23	366.60	404.07	0.163	0.7797	-0.8090	0.0001	0.0000	999.5752	0.0001	0.9924	3.00
0.466	1.000	642.42	356.49	402.10	0.001	0.7832	-0.7824	0.0004	0.0015	999.1876	0.0001	0.4417	2.00
0.537	0.822	656.48	367.56	405.66	0.195	0.7766	-0.8092	0.0000	0.0000	999.7370	0.0003	0.7074	1.00
0.681	0.909	650.08	363.60	402.72	0.095	0.7838	-0.8046	0.0007	0.0006	999.6058	0.0002	0.2415	4.00
0.437	0.922	649.33	363.04	402.48	0.080	0.7770	-0.7950	0.0002	0.0003	999.7998	0.0005	0.7984	6.00
0.443	0.862	646.51	372.68	379.04	0.002	0.9821	-1.4079	0.0000	0.0003	906.7233	0.0003	0.7131	2.00
0.502	0.879	652.25	365.59	402.89	0.127	0.7815	-0.8095	0.0004	0.0000	998.9728	0.0006	0.3811	4.00
0.572	0.777	658.63	369.23	406.21	0.254	0.7760	-0.8091	0.0002	0.0000	1000.0260	0.0001	0.9700	3.00
0.393	1.000	637.89	369.15	371.33	0.000	0.9868	-1.3949	0.0004	0.0000	899.0909	0.0005	0.9764	7.00
0.374	0.930	648.71	362.45	402.45	0.073	0.7769	-0.7935	0.0003	0.0015	999.6369	0.0004	0.3324	5.00
0.403	0.956	646.43	360.32	402.29	0.045	0.7835	-0.7943	0.0006	0.0004	999.5351	0.0003	0.9909	12.00
0.453	0.704	660.13	368.53	408.99	0.353	0.7780	-0.8019	0.0011	0.0000	999.8432	0.0002	0.9915	12.00
0.548	0.719	658.09	367.04	408.34	0.333	0.7779	-0.8017	0.0003	0.0001	997.7447	0.0004	0.9589	6.00
0.512	0.828	656.24	367.27	405.74	0.187	0.7760	-0.8091	0.0003	0.0000	999.4477	0.0013	0.7744	2.00
0.498	0.923	642.48	351.75	410.54	0.051	0.8843	-0.9491	0.0046	0.0015	972.2668	0.0023	0.8401	1.00
0.414	0.839	642.20	370.68	375.71	0.001	0.9833	-1.3948	0.0001	0.0000	903.6669	0.0006	0.9771	3.00
0.719	0.837	655.21	368.12	403.36	0.177	0.7778	-0.8092	0.0000	0.0000	999.8699	0.0002	0.2085	1.00
0.521	0.863	653.23	366.48	402.97	0.147	0.7813	-0.8096	0.0001	0.0000	999.6073	0.0002	0.6279	6.00
0.441	0.772	646.47	372.70	378.92	0.085	0.9821	-1.4079	0.0000	0.0001	916.1105	0.0008	0.9903	4.00
0.398	1.000	639.24	369.67	372.59	0.002	0.9875	-1.3942	0.0001	0.0005	901.2440	0.0001	0.9953	12.00
0.446	0.971	647.52	373.13	379.86	0.002	0.9809	-1.3998	0.0004	0.0002	909.2193	0.0004	0.4053	1.00
0.646	0.775	658.54	369.21	406.12	0.255	0.7763	-0.8091	0.0000	0.0001	999.9623	0.0003	0.8124	6.00
0.646	0.773	658.86	369.45	406.21	0.258	0.7763	-0.8092	0.0000	0.0000	1000.2953	0.0003	0.2946	1.00
0.422	0.999	642.95	356.92	402.22	0.000	0.7858	-0.7871	0.0002	0.0015	999.2970	0.0005	0.7268	3.00
0.485	0.872	652.85	375.41	384.39	0.001	0.9831	-1.3948	0.0001	0.0000	917.5700	0.0007	0.2702	1.00
0.496	0.890	653.99	375.92	385.32	0.001	0.9843	-1.3943	0.0018	0.0003	919.4388	0.0003	0.7769	9.00
0.544	0.982	643.97	357.92	402.25	0.018	0.7860	-0.7906	0.0019	0.0056	998.9276	0.0004	0.7794	6.00
0.549	0.591	652.36	358.58	412.07	0.527	0.8379	-0.8800	0.0005	0.0001	996.2214	0.0014	0.7070	3.00
0.502	0.851	654.70	376.31	385.73	0.000	0.9850	-1.3924	0.0000	0.0002	920.9502	0.0001	0.9999	18.00
0.521	0.754	643.97	349.67	415.82	0.294	0.8856	-0.9692	0.0001	0.0000	979.6375	0.0004	0.9953	12.00
0.619	0.651	656.94	361.32	414.41	0.432	0.8184	-0.8632	0.0002	0.0000	997.7046	0.0006	0.8054	3.00
0.368	0.697	659.72	368.24	408.84	0.364	0.7797	-0.8020	0.0003	0.0001	999.4255	0.0004	0.2854	3.00
0.541	0.709	646.42	350.01	418.83	0.355	0.8762	-0.9654	0.0001	0.0000	984.1087	0.0004	0.9187	5.00
0.581	0.596	654.08	359.61	412.97	0.521	0.8389	-0.8802	0.0002	0.0002	999.1343	0.0005	0.7891	7.00
0.646	0.808	657.56	368.42	405.91	0.214	0.7763	-0.8092	0.0000	0.0001	1000.4215	0.0003	0.2391	2.00
0.375	0.841	631.90	364.17	371.53	0.000	0.9956	-1.3919	0.0005	0.0007	892.4365	0.0001	0.6551	6.00
0.545	0.807	646.74	350.13	419.10	0.207	0.8762	-0.9654	0.0001	0.0000	976.2270	0.0009	0.2010	5.00
0.616	0.651	656.73	361.19	414.33	0.432	0.8181	-0.8632	0.0001	0.0000	997.2986	0.0014	0.5429	4.00
0.382	0.999	635.69	367.95	369.99	0.001	0.9918	-1.3924	0.0001	0.0018	897.8411	0.0002	0.7662	6.00
0.701	0.773	658.99	369.55	406.24	0.259	0.7763	-0.8092	0.0000	0.0003	1000.4378	0.0001	0.8978	8.00
0.485	0.715	659.47	368.00	408.83	0.339	0.7775	-0.8014	0.0001	0.0001	999.4167	0.0006	0.9334	14.00
0.589	0.606	655.06	360.25	413.39	0.503	0.8343	-0.8789	0.0004	0.0000	998.7744	0.0006	0.9849	6.00
0.522	0.893	651.32	364.84	402.67	0.113	0.7890	-0.8143	0.0000	0.0000	999.7227	0.0002	0.7928	8.00
0.395	0.974	638.52	369.38	371.92	0.002	0.9893	-1.3963	0.0011	0.0007	900.4267	0.0011	0.0235	2.00
0.646	0.794	657.92	368.71	405.98	0.231	0.7763	-0.8092	0.0000	0.0001	1000.1641	0.0003	0.2323	2.00
0.477	0.863	653.17	366.43	402.96	0.148	0.7815	-0.8095	0.0001	0.0000	999.6073	0.0002	0.5425	7.00
0.555	0.818	656.97	367.98	405.73	0.202	0.7765	-0.8092	0.0000	0.0000	1000.2054	0.0008	0.8824	9.00
0.402	0.775	640.58	370.31	373.61	0.091	0.9850	-1.3920	0.0000	0.0015	912.4028	0.0008	0.2854	1.00
0.385	0.976	636.41	368.23	370.69	0.001	0.9917	-1.3941	0.0002	0.0028	898.5005	0.0002	0.7428	4.00
0.406	0.872	641.55	370.62	374.68	0.001	0.9831	-1.3949	0.0001	0.0000	902.9067	0.0021	0.9893	14.00
0.472	0.781	658.39	369.14	406.03	0.249	0.7763	-0.8092	0.0001	0.0000	1000.0478	0.0004	0.2867	2.00
0.717	0.945	647.34	361.19	402.35	0.057	0.7772	-0.7899	0.0002	0.0013	999.5583	0.0004	0.1072	1.00
0.389	0.694	659.91	368.37	408.91	0.366	0.7800	-0.8019	0.0001	0.0000	999.6985	0.0002	0.7810	9.00
0.589	0.605	655.37	360.44	413.55	0.506	0.8342	-0.8790	0.0007	0.0001	999.2063	0.0004	0.6947	9.00
0.379	0.976	635.71	367.96	370.00	0.002	0.9913	-1.3928	0.0000	0.0005	897.8177	0.0018	0.2927	2.00
0.581	0.640	652.79	358.78	412.38	0.460	0.8389	-0.8802	0.0002	0.0002	996.9392	0.0023	0.4658	3.00
0.625	0.651	657.64	361.77	414.73	0.431	0.8179	-0.8632	0.0004	0.0000	998.5541	0.0005	0.9764	13.00
0.434	0.748	645.70	372.37	378.26	0.138	0.9822	-1.4077	0.0000	0.0000	921.2192	0.0001	0.1068	1.00
0.450	0.681	658.93	367.77	408.45	0.388	0.7836	-0.7995	0.0002	0.0001	999.3950	0.0008	0.4118	3.00
0.476	0.801	651.66	374.79	383.62	0.025	0.9814	-1.3967	0.0001	0.0000	917.8542	0.0001	0.6075	7.00
0.465	0.747	650.77	374.64	382.36	0.140	0.9822	-1.4080	0.0000	0.0000	928.4057	0.0002	0.2865	2.00
0.422	0.872	644.18	371.57	377.31	0.001	0.9830	-1.3948	0.0001	0.0000	906.1741	0.0033	0.9223	12.00
0.418	0.801	644.01	371.51	377.14	0.025	0.9815	-1.3967	0.0001	0.0000	907.8108	0.0044	0.9166	10.00
0.503	0.760	659.30	369.84	406.26	0.275	0.7765	-0.8092	0.0000	0.0000	1000.2538	0.0001	0.7399	6.00
0.530	0.737	645.47	350.33	416.98	0.316	0.8829	-0.9730	0.0001	0.0000	981.3776	0.0006	0.2435	2.00
0.481	0.934	652.20	375.03	384.07	0.000	0.9826	-1.3951	0.0008	0.0001	916.3930	0.0003	0.1073	1.00
0.368	0.976	634.71	367.59	369.02	0.002	0.9914	-1.3936	0.0002	0.0002	896.3896	0.0026	0.6345	7.00
0.737	0.791	658.19	368.92	406.05	0.236	0.7760	-0.8089	0.0000	0.0007	1000.3051	0.0001	0.2865	3.00
0.522	0.959	648.70	362.37	402.55	0.028	0.7890	-0.8142	0.0000	0.0000	998.6697	0.0066	0.4006	3.00
0.449	0.747	649.96	374.27	381.71	0.140	0.9822	-1.4080	0.0000	0.0000	927.2781	0.0017	0.5378	3.00
0.521	0.765	659.30	369.84	406.26	0.268	0.7765	-0.8092	0.0000	0.0000	1000.5648	0.0000	0.0428	1.00

Figure 10: Activated classifiers from one run of *frog2* of Fig. 8b. Values of the action and prediction columns are the values computed from the last input.

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

[15] Wilson, S. W., Classifier Systems for Continuous Payoff Environments, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, K. Deb et al, eds. Springer-Verlag, Berlin, pp. 824-835 in Part II, 2004.

[16] Wilson, S. W., Three Architectures for Continuous Action, *Technical Report No. 2006019*, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 2006.