An Artificial Life Classifier System for Real-Valued Inputs

Julian Bishop
Artificial Intelligence Center
University of Georgia
Athens, GA 30602-7415
+1 706 254 0546
julian1@uga.edu

ABSTRACT
Classification problems with real-valued inputs still represent a challenge for Learning Classifier Systems (LCS), particularly when the training data is distributed non-uniformly or different classes cover different proportions of the input space. This motivates the design of a new LCS following an Artificial Life model. Initial results are encouraging and future work is outlined.

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
ALCSR, A-Life, Learning Classifier System, Endogenous fitness, Reinforcement Learning (RL), Genetic Based Machine Learning.

1. INTRODUCTION
This paper presents ongoing work towards the development of a Michigan-style LCS for handling real-valued inputs and discrete actions. This work grew from a project to implement Wilson's XCSR [3][6][7] which provides many ideas that are important here. However, this work is guided by its own artificial life model, and the resulting LCS, named ALCSR, is not an XCS variant in my opinion. ALCSR employs an endogenous fitness scheme inspired by Holland's ECHO model [4]. A similar endogenous fitness scheme has already been shown to work in an LCS by Booker [1][2], although Booker's system is quite different overall handling binary inputs (0,1,#) and employing partial matching which is not used here.

Section 2 sets out the goals of this work. Section 3 describes ALCSR, guided by an abstract model of how a population of artificial organisms might evolve usefully in the RL setting. Initial experiments are presented in Section 4. Section 5 discusses the results and Section 6 concludes and outlines ongoing work.

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

2. SYSTEM DEVELOPMENT GOALS
(1) **Problem**: The first version of ALCSR (presented here) should handle noise-free single-step classification problems with discrete fixed reward levels P over the space of continuous input values X and discrete actions A.

(2) **Extensions**: The system should be extensible to handle noisy problems and also multi-step problems with delayed rewards.

(3) **Model**: ALCSR should learn a complete map ($X \times A \Rightarrow P$) of the reward levels that is independent of any exploitation policy (like an accuracy-based system such as XCS).

(4) **Performance**: In the absence of noise, ALCSR should achieve stable perfect performance using a simple exploitation policy (at least in single-step classification problems).

(5) **Niching**: All niches in the input space should be discovered and populated regardless of their relative sizes, or the relative frequencies with which representative inputs occur. In other words, if a niche exists, ALCSR should learn accurate rules to cover it even if it is small and surrounded by much larger niches of different types.

(6) **Optimality**: The population should continually evolve toward the minimal set of maximally general rules that could solve the problem without becoming stalled on a sub-optimal population.

(7) **Currency**: Each rule in the population should have its own reservoir of currency (like a strength-based system) so that ALCSR could be extended to investigate ways in which transactions between rules might improve the system's capabilities and performance.

(8) **EA Scope**: ALCSR is intended to serve as a platform for broadening the scope of the evolutionary algorithm within an LCS. It would be interesting to extend the rule's genetic representation to facilitate the evolution of new features. This agenda will include the evolution of condition morphologies and cooperative interactions between rules.

3. SYSTEM DESCRIPTION
3.1 Natural Problem Topography
A-Life Model: Each data-point/training example experienced by ALCSR is a vector X of n real values, and is taken to represent a single unit of food welling up from the ground. The n real values locate the unit of food on an n-dimensional landscape in the space of possible system inputs. If the system has c potential actions, then c completely separate landscapes are considered to exist, one for each action. The type of food is not determined until ALCSR

has executed an action and a non-zero reward has been returned to it. The reward level identifies the type of food; the quantity is always 1 unit.

Each landscape is to be colonized by a population of organisms that cannot move around once established. Organisms on different landscapes are considered to be completely different major species and are never permitted to interact in any way. Within each landscape, organisms may be adapted to eating just one type of food. Organisms that are adapted to eating different types of food are considered to be different sub-species. In order to survive, each organism must maintain positive strength by feeding on the appropriate food-type for its sub-species.

Each organism has a body which covers a region of the landscape within which it can feed. The size of an organism is defined as the n-dimensional hyper-volume it covers on the landscape. Multiple organisms can cover the same region of landscape.

A-Life term	LCS Analogue
Landscape	Map of reward levels over the space of system inputs for a single action.
Organism	Rule / Classifier
Body	Rule / Classifier condition
Size	Rule generality
Food, 1 unit	Data-point / training example
Food-type	Reward level
Major species	Rules predicting for the same action
Population	Set of all rules predicting for the same action
Sub-species	Rules predicting same reward re same action

Rule Implementation: A rule is completely described by {Condition, Action, Prediction, Strength, Experience, Mature, Starved, CanBreed, Damaged}. Condition is the conjunction of n half-open intervals $[l_i, u_i)$, each represented as an ordered pair of bounds, which is satisfied if $l_i \leq x_i < u_i$. The entire condition describes a hyper-rectangle in the space of possible inputs. The generality of a rule is the hyper-volume of its condition. Hyper-volumes are maintained in normalized form such that a volume of 1 covers the entire known landscape. Action is actually implied by a rule's containing population where it is represented as an integer code. Prediction is a real number representing the expected reward level should Action be executed. Strength is a real number representing the rule's state of health. Experience is an integer representing how many times the rule has been in the action set. The remaining four attributes are Boolean flags.

3.2 Performance System

Implementation: For each population, all rules whose condition is satisfied by the input vector are added to a potential action set; there is a separate potential action set for each possible action. If any of the potential action sets are empty, one of the corresponding actions is selected at random for exploration. If no potential action set is empty then with probability P(explore) an action is chosen at random for exploration. With probability 1-P(explore) this operating cycle is treated as a test problem and the rules in each potential action set take a weighted vote on what the reward level will be if their corresponding action is executed. The weight for each rule is calculated as its strength per unit hyper-volume of its condition. A rule that is not mature is allowed to vote, but the weight of its vote is reduced by a factor of 10. The action with the highest predicted reward level is selected. This exploitation scheme is based on the prediction array used in XCS but with strength per unit hyper-volume replacing fitness. ALCSR presents its selected action to its environment which returns one of a finite set of real-valued reward levels. All experiments so far have used {+1000, -1000}. In test problems, this is the end of the operating cycle.

3.3 Accuracy Pressure

A-Life Model: Each single unit of food is evenly split m ways between all m organisms covering the location of the food on the landscape corresponding to the action just executed by the system. Whether or not the consuming organisms benefit from this food depends upon how well adapted they are to metabolizing food of that type. The gain in strength to each organism is offset by a fixed cost of metabolism and the potentially fatal loss of strength that occurs if an organism is poisoned by food of a type it is not well adapted for. The less well adapted the more loss occurs. An organism whose strength becomes negative is considered to have died. This is the death pressure against mal-adaptation. The fixed cost of metabolism introduces an additional pressure against overcrowding.

A-Life term	LCS Analogue
Feeding organisms	Rules in the action set
Number of times fed	Action set experience
Degree of adaptation	Accuracy of reward prediction
Poisoned by food	Inaccurate reward prediction
Starved	rl.Strength <= 0
Damaged	rl.Error > 0.01
Organism dies	Rule removed from population

Reinforcement Implementation: The strength of each rule rl in the action set is updated using:

rl.Strength += (1 – rl.Error · ErrorCost) / m – MetabolismCost

rl.Error is the rule's prediction error as a fraction of the reward actually returned; ErrorCost is the damage that would occur for a 100% error; m is the number of rules in the action set; MetabolismCost is a flat cost set to 1/50. If the action set contains more than 49 rules, then they cannot gain in strength even if they all make zero error, and it is possible for the weaker rules to starve to death due to overcrowding. ErrorCost is set to 2000 to provide a very strong death pressure against inaccurate rules; it is insensitive to relative accuracy (differing from XCS).

3.4 Genetic Generalization Pressure

3.4.1 Creating More General Rules

A-Life Model: A breeding pool is formed by an organism and its k_b-nearest neighbors on the landscape any time a sufficient proportion of them satisfy breeding criteria (currently half or more). Breeding criteria are that all organisms that will participate in a breeding pool must be of the same sub-species and must have their strength and experience above fixed thresholds per unit hyper-volume of their bodies. Setting these thresholds per unit hyper-volume is intended to result in all organisms being fairly evaluated for breeding. Furthermore, if food is uniformly distributed over the landscape then the breeding rate should be about the same in all niches regardless of their size. Requiring that a certain proportion of an organism's k_b-nearest neighbors satisfy breeding criteria should focus breeding within niches rather than between them. In our experiments $k_b = 12$.

Once a breeding pool is formed, breeding events occur in it until the breeding criteria are no longer satisfied by enough of its members. A breeding event consists of two distinct parents randomly selected (but not removed) from the breeding pool producing a single offspring of the same sub-species by crossover, mutation and strength transfer. Crossover provides the genetic pressure toward the production of increasingly large organisms, i.e. offspring tend to be larger than their parents. If the resulting offspring's body precisely duplicates that of another organism then small mutations are applied until the offspring is unique.

A-Life term	LCS Analogue
Distance between organisms	Euclidean distance between geometric centers of rule conditions
Mature	rl.Experience > MatureLevel · Generality
CanBreed	rl.Strength >= BreedLevel · Generality

Implementation: MatureLevel = 500. Crossover: the condition of the offspring is formed such that with probability 2/3 in each dimension, it will be a generalization of its parents. With probability 1/3 in each dimension, the offspring's condition will be a copy of one of its parents, picked randomly as in uniform crossover. Mutation: with probability 1/12 each condition bound is independently mutated by a random amount, at most 1% of the corresponding input's value range. Strength Transfer: the strength each parent transfers to the offspring is calculated as

P_1.Transfer = stf · BreedLevel · Min(P_1.Gen, O.Gen · P_1.RelSize)
P_2.Transfer = stf · BreedLevel · Min(P_2.Gen, O.Gen · P_2.RelSize)

P_1 and P_2 denote the parents; O denotes the offspring; Gen denotes generality which is a normalized hyper-volume; stf denotes StrengthTransferFraction = 0.43; BreedLevel = 376; and

P_x.RelSize = P_x.Gen / (P_1.Gen + P_2.Gen)

The offspring's experience is set to zero and all its flags are set to False.

3.4.2 Removing Less General Rules

A-Life Model: Organisms of the same sub-species that cover the same region of the landscape are in competition for the same units of food, and so the larger organisms constantly initiate combat with the smaller ones, attempting to kill and eat them. However, such attacks are only made by mature organisms against other mature organisms, and are only successful if the aggressor has greater strength than its intended victim. Conveniently, the cost to the aggressor of killing another organism is equal to that organism's strength and therefore precisely replenished by eating its dead body. A further constraint on successful attacks is that the aggressor must fully engulf (logically subsume) its victim. (This could be relaxed to investigate fuzzy subsumption schemes and a stochastic combat model.) In the present model, all constraints are enforced and all possible fights are always played out among the feeding organisms on each system cycle. This is the death pressure against needlessly small organisms (over specific rules).

3.5 Covering Pressure for Specificity

A-Life Model: When an organism dies due to eating too much of the wrong type of food its corpse remains inactive on the landscape, and those parts of its body that caused it to die are wizened and infertile. However, any part of the organism's body that had allowed it to gain strength from feeding is considered to have the potential to sprout new organisms of the same sub-species. Such sprouting only occurs in the absence of other rules of that sub-species on that part of the landscape. Sprouting is intended to produce new organisms with at least one surface of their body likely to be in the region of the boundary between different food-types on the landscape. This is the pressure toward the production of organisms that are small enough to eat only a single type of food (rules that are specific enough to make no prediction errors). Additionally, sprouting should increase the chance of introducing useful alleles into the sub-species' gene pool as new organisms are created progressively closer to the food-type boundaries the sub-species cannot cross.

When ALCSR is initiated there are no organisms and the landscapes are completely undiscovered so a seeding process is required to get the populations started. Seeding is stopped when there are enough corpses for sprouting to take over.

A-Life term	LCS Analogue
Seeding	Traditional random covering
Sprouting	ALCSR Adaptive covering
Food-type boundary	Concept decision surface

Implementation: This step is only performed if the system has just explored the action corresponding to an empty potential action set. A new rule is created with zero strength, zero experience and all flags set to False. Its prediction is set to the reward level just returned. Defining the new rule's condition is more complicated. Each population has associated with it a set, termed mausoleum, for storing rules that have died due to making inaccurate predictions. The new rule's condition bounds are randomly generated as

$l_i = x_i$ – Rnd[0, LowerCoverFraction$_i$·r_i) and
$u_i = x_i$ + Rnd[0, UpperCoverFraction$_i$·r_i) where r_i is input i's range

If there are fewer than 20 rules in the relevant mausoleum, then all CoverFractions are set to 0.5. If there are 20 or more rules in the relevant mausoleum then the CoverFractions are set by adaptive covering. This is the most challenging part of the artificial life model to implement because it requires the rules in the mausoleum to know in which regions of their conditions their predictions were accurate or inaccurate, which must be estimated.

Each dimension i of a rule's condition has four attributes: StrengthCenter$_i$, WeaknessCenter$_i$, LowerCoverFraction$_i$ and UpperCoverFraction$_i$. StrengthCenter$_i$ and WeaknessCenter$_i$ are maintained as the average ordinate value in each dimension i of all correct and incorrect predictions respectively. When a rule dies with Damaged set to True, it is added to the appropriate mausoleum and its CoverFractions are set to represent its size in each dimension as fractions of each full attribute range. The dimension in which StrengthCenter and WeaknessCenter are furthest apart is guessed as being most likely to span a decision surface, and one of the rule's CoverFractions in that dimension is reduced accordingly by a factor of 2. If the StrengthCenter has a lower value than the WeaknessCenter then the UpperCoverFraction is chosen as the one to be reduced, otherwise the LowerCoverFraction is reduced. The CoverFractions of the dead rule's k_c-nearest neighbors in the mausoleum are updated to reflect this new information using a Widrow-Hoff update in which the learning rate is inversely proportional to the distance of the neighbor.

When adaptive covering creates a new rule, the k_c corpses in the mausoleum nearest to the input vector X take an inverse-distance weighted vote on what values should be assigned to the new rule's CoverFractions, which are then used to generate the condition bounds. If a condition lower or upper bound is generated that lies outside the range of values that ALCSR has experienced in that dimension, the bound is set to the minimum or maximum value experienced respectively. Note that this increases the probability of generating a bound equal to the range limit as discussed in [5]. In our experiments $k_c = 5$.

4. INITIAL EXPERIMENTS

In [7], Wilson presented the results of two experiments in which XCSR learned a real-valued version of the Boolean 6-multiplexer function. Instead of 6 binary values, the input vector consists of 6 real values $0.0 \leq x_i < 1.0$, generated randomly with uniform distribution. In each dimension i a decision threshold is used to interpret the component value as 0 or 1 so that the multiplexer can be applied. In order to present our initial results in an established context, Wilson's experiments have been repeated using the implementation of ALCSR described in Section 3, run for 300,000 system cycles with P(Explore) = 0.9. As in [7], each experiment was run five times. Results with ALCSR were found to vary very little between runs, so the figures below present results from single representative runs.

In the following results percentage performance is the classification rate over the previous 50 test problems. Where 50 test problems do not provide sufficient precision to report results, other measures are stated specifically. System error is calculated as a moving average, also over the previous 50 test problems, and is presented as a fraction of the reward range (2000) in these experiments.

Experiment 1: All decision thresholds are set to 0.5. In [7] Wilson reports that XCSR achieves a maximum performance of approximately 98% after about 15,000 explore problems. Figure 1 presents typical performance and system error for ALCSR. In 15,000 explore problems ALCSR achieves a lower performance of approximately 95%, but continues to improve reaching 98% after about 35,000 explore problems. For the last 10,000 test problems in a run (after about 180,000 explore problems), ALCSR typically provides the correct classification for more than 9,970.

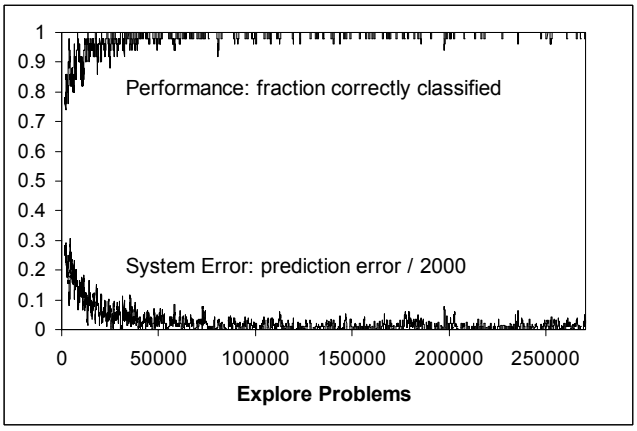

Figure 1. ALCSR Performance and System Error in Experiment 1

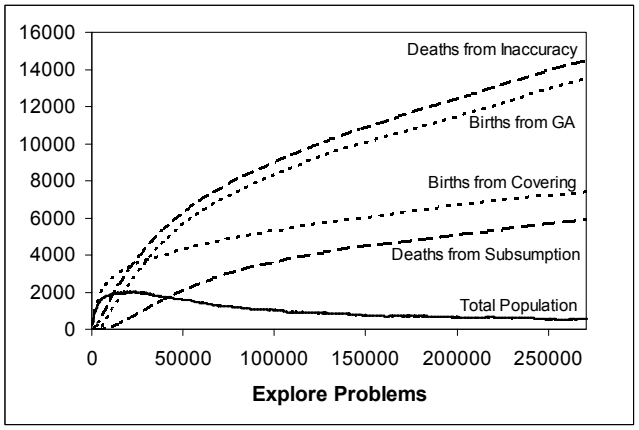

Figure 2. ALCSR Population Dynamics in Experiment 1

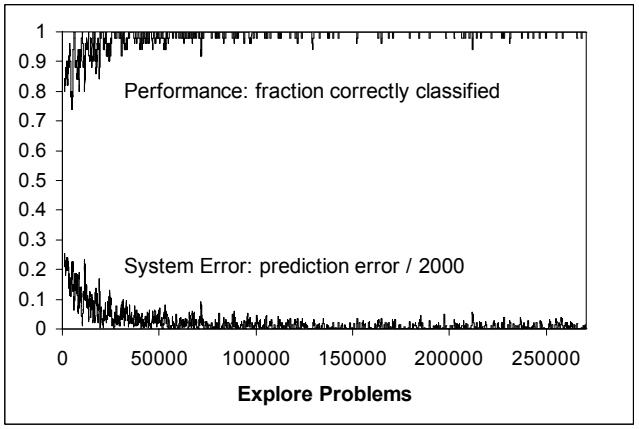

Figure 3. ALCSR Performance and System Error in Experiment 2

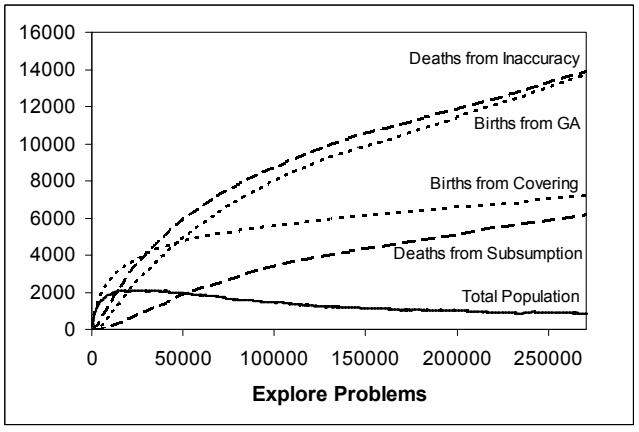

Figure 4. ALCSR Population Dynamics in Experiment 2

Figure 2 shows how the population size varies during a typical run, and also the number of births and deaths. The total rule count peaks around 2,000 after about 20,000 explore problems and then falls to around 535 by the end of the experiment, although this varies more than the performance results: the smallest final population was 452, and the largest was 607.

Experiment 2: The decision thresholds are set at 0.25 and 0.75 in alternate dimensions which skews the probability of generating example points in the different niches. In [7] Wilson reports that XCSR achieves a maximum performance of approximately 93% after about 10,000 explore problems. Figure 3 presents typical performance and system error for ALCSR. In 10,000 explore problems ALCSR also achieves 93% performance, but again, continually improves thereafter, reaching 98% after about 50,000 explore problems. Over the last 10,000 test problems in a run, ALCSR's performance is the same as in Experiment 1.

Figure 4 shows how the population size varies during a typical run, and also the number of births and deaths. The total rule count peaks around 2,100 after about 27,000 explore problems and then falls to around 950 by the end of the experiment. The smallest final population was 859, and the largest was 1112.

5. DISCUSSION

The version of XCSR used in [7] employed a center-spread representation for its condition bounds, while ALCSR uses ordered bounds. In [5], Stone and Bull investigate these representations for XCSR and also an unordered bounds representation, providing many useful insights. They found system performance for the real-valued 6-multiplexor consistent with Wilson's results in [7] and affected very little by the choice of representation (although other measures were affected as was performance on other problems).

The fact that ALCSR's best reached classification performance in Experiment 2 is as good as Experiment 1 is encouraging with respect to the Niching goal (5). However, Experiment 2 does cause ALCSR to learn at a slower rate than in Experiment 1, and produce slightly larger populations.

Compared to XCSR, ALCSR learns much more slowly but eventually improves beyond XCSR's maximum performance achieving 99.7% measured over 10,000 test problems, which is encouraging with respect to the Performance goal (4). An analysis of why ALCSR still makes any mistakes at all would be of interest.

Examinations of the final populations from both experiments have identified very good representatives of all niches, even the smallest, which is encouraging with respect to the Model goal (3). However, the majority of the rules in the final populations are very specific rules clustered around the decision surfaces in the input space. The adaptive covering scheme was designed to have this effect so that increasingly useful condition bounds (alleles) would continue to be discovered throughout the run. However, in its current implementation adaptive covering appears to be too aggressive for the system to achieve the Optimality goal (6), unnecessarily bloating the population. About half the rules in each final population are so specific that they very rarely participate in system operation and are mostly a drain on computing resources. If the persistence of these rules could be avoided, the Optimality goal (6) would be within reach.

6. CONCLUSIONS AND ONGOING WORK

ALCSR has a number of features that make it unusual among learning classifier systems. Specifically: ALCSR identifies individuals suitable for breeding by the use of speciation and by exploiting the natural topography of the problem; ALCSR employs an endogenous fitness scheme; GA timing and population size are emergently controlled; ALCSR uses adaptive covering which continues throughout training to provide the GA with increasingly useful alleles so mutation is not relied upon for allele discovery and is just used to avoid duplicate conditions; Since there are no duplicate rules, numerosity is obviated.

Although some of the goals set out in Section 2 have not yet been addressed, inroads have been made towards goals 1, 3, 4, 5, 6 and 7, and Section 2 represents an agenda for ongoing research. Additionally, a minimal set of necessary system parameters for ALCSR needs to be identified, and self-adaptive parameter schemes investigated.

The immediate priority for ALCSR is an analysis of the current implementation's population dynamics in order to understand and correct undesirable characteristics such as slow learning and over specialization at the decision surfaces. Such analysis will probably result in modifications to the implementation and possibly to the artificial life model as well. The modified ALCSR should also be tested against a wider range of problems such as more challenging multiplexers and the Checkerboard problem described in [5].

7. ACKNOWLEDGEMENTS

I am grateful to Alwyn Barry, Stewart Wilson, Khaled Rasheed, Don Potter, Brian Smith and Daniel Tuohy for helpful comments and discussions about this work and/or related issues.

8. REFERENCES

[1] Booker, L.B. (2000). "Do We Really Need to Estimate Rule Utilities in Classifier Systems?" Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Learning Classifier Systems: From Foundations to Applications, pp 125-141

[2] Booker, L.B. (2001). "Classifier systems, endogenous fitness, and delayed rewards: A preliminary investigation" In Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E., eds., Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), pages 921-926, San Francisco, California, USA, 7-11 July 2001.

[3] Butz, V.B., Wilson, S.W. (2001). "An algorithmic Description of XCS" in Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Advances in Learning Classifier Systems, pp 253-272

[4] Holland, J.H. (1995). "Hidden Order: How Adaptation Builds Complexity" Helix Books.

[5] Stone, C., Bull, L. (2003). "For Real! XCS with Continuous-Valued Inputs" Evolutionary Computation 11(3): 299-336

[6] Wilson, S.W. (1995). "Classifier Fitness Based on Accuracy" Evolutionary Computation 3(2): 149-175

[7] Wilson, S.W. (2000). "Get Real! XCS with Continuous-Valued Inputs". Lanzi, P.L., Stolzmann, W., Wilson, S.W. (eds.) Learning Classifier Systems: From Foundations to Applications, pp 209-219