FIXCS: a Fuzzy Implementation of XCS

Andrea Bonarini *Member, IEEE* and Matteo Matteucci *Member, IEEE*

***Abstract*— We present FIXCS (Fuzzy Implementation of XCS), a learning classifier system that extend the accuracy-based eXtended Classifier System (XCS) by allowing to match real-valued input by fuzzy sets, and to produce a fuzzy output, then translated into real values. This work gives XCS the ability to face real-valued problems with a fuzzy model that approximates a real valued function better than the original, interval-based model. First results show that, as expected, the learning time is longer, but the obtained fuzzy system is more robust than the interval-based one.**

I. Introduction

In the last years, learning classifier systems, as originally proposed by Holland [1], have received a new attention, mainly due to the introduction of a new fitness evaluation method, based on accuracy, firstly implemented by Wilson in XCS [2]. Traditionally, LCS (and XCS) learn classifiers (i.e., classification rules) represented by binary strings. Each portion of such binary string corresponds to a symbol, specific for the problem, representing the actual input or output. Most of the work in LCS and XCS has been done on discrete problems; when real-valued problems are faced, symbols used in the classifiers denote ad-hoc discretization intervals.

In the last years, we have worked on extending the interval-based model for LCS to the more general fuzzy model [3], [4], [5], where sub-strings denote labels for fuzzy sets. This has been shown to produce a more accurate approximation of the real-valued function relating input and output, also resulting in a more robust model, able to successfully face real-world problems affected by noise.

Other researchers have tried to integrate fuzzy logic and LCS, in particular to extend XCS [6], [7]. In this last paper a solution is proposed to reduce the noise effect (called "cluster aliasing" in [3]) due to the triggering of many fuzzy rules on the same state, to which is associated a reward, while the same rules triggers together with others in different states. In [7] only one rule is selected to trigger, thus losing the beneficial effect w.r.t. noise in the application space (related to "perceptual aliasing") due to the interaction of fuzzy rules. In our approach we take a different way, and aim at the generation of a standard fuzzy controller by adopting a standard XCS modified to consider fuzzy rules. As shown also by the results reported here, this makes learning slower, but preserves the mentioned desired characteristic of a fuzzy system.

In this paper, we present how we apply our fuzzyfication approach to XCS, and we discuss the preliminary results obtained on the classical, non-linear cart-pole problem. We first recall some basic notions about Learning Fuzzy Classifier Systems (LFCS) and XCS, then introduce FIXCS, and finally discuss the results of some experiments.

Andrea Bonarini and Matteo Matteucci are with the AI and Robotics Lab, Department of Electronics and Information, Politecnico di Milano, Piazza Leonardo da Vinci, 32, 20133 Milan, Italy (phone: +39 23993525; email: {bonarini, matteucc}@elet.polimi.it).

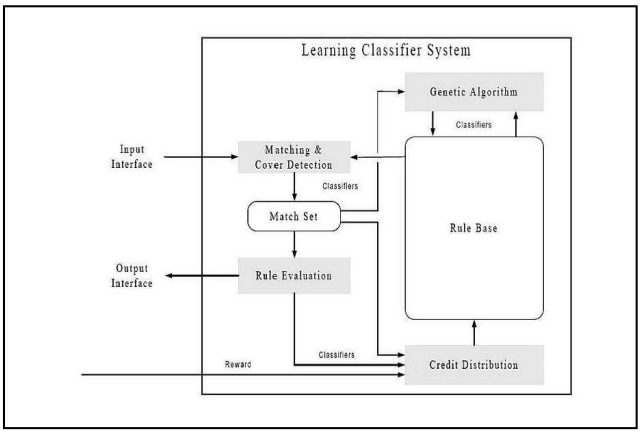

Fig. 1. Block Diagram of a generic Learning Fuzzy Classifier System (LFCS)

II. Background

In this section, we present an architectural description of a Learning Fuzzy Classifier Systems (LFCS) and XCS.

A. LFCS

The generic architecture shown in Figure 1 is general enough to hold both for LCS and LFCS, only with some modifications in how the different modules are implemented [5]. The input interface will read the state from the environment and adapt it to the classifier system, by matching it to the internal representation. In case of interval models, this gives a single state match; in case of a fuzzy representation it will most likely result in several different fuzzy states matching the real-valued sensorial input with a different matching degree each. This is a first difference among interval-based and fuzzy models: the first match the real-valued input with only one internal state, the latter with many.

According to this, the fuzzy matching module will have to create a match set $[M]$ with different subpopulations of rules, each subpopulation matching one of the fuzzy states perceived. Each fuzzy state is composed by a set of fuzzy sets matching the inputs, together with their matching degree. Cover detection, i.e., the generation of rules to cover situations not yet faced, should be applied to each subpopulation needing it. On the other side, when the matching module is crisp, the obtained match set $[M]$ would only have one subpopulation matching the only perceived state.

1-4244-1210-2/07/$25.00 ©2007 IEEE.

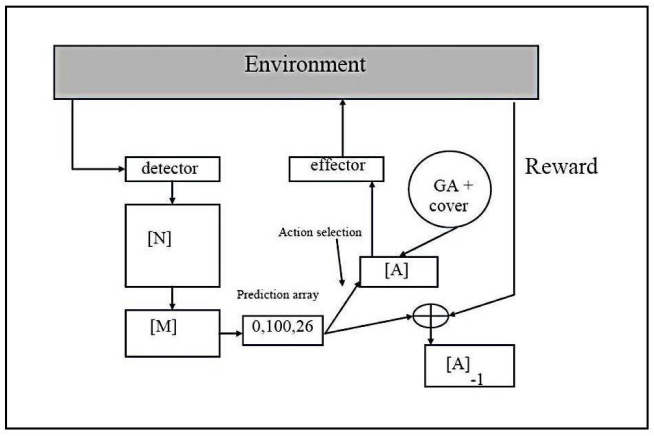

Fig. 2. Block Diagram of a simple XCS system from [10].

Then, one rule is selected in each subpopulation of classifiers in the match set, according to exploitation/exploration criteria. The actions proposed by the selected rules are then composed by the standard fuzzy operators to obtain a resulting action to be executed in the environment.

The reward received is distributed to the rules that collaborateg to produce the final action, according to their matching degree. The genetic algorithm does not differ between fuzzy and crisp implementations since it works on symbols in both cases.

B. XCS

In 1995, Wilson introduced XCS [2], a significant improvement to traditional learning classifier systems. XCS retains essential aspects of Holland's model, while adding some new features. First, the classifier fitness, used to select rules on which applying gentics, is based on the *accuracy* of a classifier's payoff prediction instead than on the prediction itself. Moreover, the genetic algorithm takes place in the action set $[A]$ instead than in the whole rule base and deletion occurs panmicticly, i.e., it uses niche genetics instead of population wide. Moreover, as it has no message list, XCS is, in principle, suitable for learning only in Markov environments, although some memory mechanisms for XCS have been introduced [8], [9] and are still under analysis.

The architecture is relatively similar to the one of LCS. There is a match set $[M]$, created out of the input state, and for every action in $[M]$ a prediction array value is calculated by averaging each rule's fitness and prediction. Using these values from the prediction array, one action is selected and an action set $[A]$ formed. Again, as in classical LCS, it has to be a trade off between exploitation of best rules and exploration of new rules.

Using the reward obtained from the system, reinforcement is distributed among the rules in the action set $[A]$. First each rule's experience exp, prediction p, error ϵ, average size as, and accuracy k are updated, and then the fitness F.

Finally, the genetic algorithm is applied on the action sets (*niches*), conditioned on the average last time the rules in the set had genetics applied. Crossover and mutation happen with a certain probability on the offspring generated out of two classifiers selected by roulette wheel selection; doing in this way, each rule will have a probability of being chosen proportional to their fitness. Rule replacement is global and based on the estimated size of each action set each rule has participated in, with the aim of balancing resources across niches.

Due to these changes in the fitness approach and GA niche application, XCS tends to evolve classifiers that are both accurate in their prediction of payoff (resulting in higher performance) and maximally general, as a predictive model, in their niche, i.e., they implement the most effective model with the highest number of "don't care" symbols instead of specific symbols in the rules.

Since the first basic XCS proposed by Wilson [2] many changes have improved the original system. For example, the accuracy measure k has evolved from the original exponential function to a power law function [11]. A specific order of update of parameters and the MAM (*"moyenne adaptive modifiée"*) technique are used in the fitness update to increase robustness against inaccurate classifiers. A detailed and complete review of how XCS works can be found in [12].

III. The Modular Architecture

Through the last decade, a quite a large amount of literature has been written on the topic of merging LCS and fuzzy logic [5], [6]. It is important to make clear how the LCS conversion into LFCS will be a capital reference to our proposal of FIXCS. We identify four different modules to consider in the architecture of a LCS: *Matching & Cover Detection*, *Rule Activation*, *Credit Distribution* and *Genetic Algorithm*. *Input/Output Interfaces* also play an important role in this fuzzyfication process being the modules performing data fuzzyfication.

1) Input and Output Interfaces: Manage the interaction with the environment, adapting the data to be processed by the LFCS. In the case of the Input Interface this is done by classifying a real-valued input into a fuzzy perception of the actual state. This is achieved by using membership functions and fuzzy operators (T-norms, basically) to calculate the correspondent matching degrees. On the other side, the output interface combines the proposed actions into a real-valued output.

2) Matching & Cover Detection: Matching selects and groups the classifiers that match the real-valued states sensed by the Input Interface. As fuzzy logic allows partial overlapping of the memberships functions, there are many possible fuzzy sets matching the same input value. This results in several subpopulations inside the same Match Set $[M]$, each with a different matching degree. As classifiers have to cooperate in the same set and compete to survive appears the *cooperation vs. competition dilemma* [5]. The Cover Detection module checks whether there is at least one classifier in the Match Set or a minimum number of different actions, and eventually creates enough classifiers up to that minimum.

3) Rule Activation: This module decides which action has to be passed to the output interface by evaluating the fitness of the classifiers in the Match Set $[M]$. This is a classic problem of LCS, and Reinforcement Learning in general; it is called the *exploitation vs. exploration dilemma*. The action selection policy has to balance between choosing the strongest classifiers and giving opportunities to the more recent ones with no such a good fitness. When applied to LFCS, an action is selected for each subpopulation in $[M]$. Out of all the classifiers that propose that action in their subpopulation, the Action Set $[A]$ is created. Then the Action Set $[A]$ will be formed by different subpopulations.

4) Credit Distribution: Credit Distribution updates the value of the fitness for all the classifiers in the Action Set $[A]$. To do so, the system receives a reward from the environment after the action is executed. In the case of multiple subpopulations in $[A]$, reinforcement should be distributed proportionally to the degree of matching of the antecedents. To make that contribution adapting to the learning process evolution, we can introduce a factor ξ (see section IV-A), as is proposed in [5].

5) Genetic Algorithm: To guarantee the evolution of the population, this module will apply genetic operators (selection, mutation, crossover) on part of the population. This part usually is either the whole population, the Match Set $[M]$ or the Action Set $[A]$. There is no conceptual difference between applying it in the crisp or in the fuzzy model.

IV. FIXCS: XCS Fuzzyfication

We are now almost ready to introduce FIXCS. Let us first point out how the differences between LCS and XCS can drive the implementation of FIXCS, using as reference the algorithmic description of XCS proposed by Butz and Wilson [12].

Each classifier in XCS has a condition-action-prediction structure and holds several parameters:.

- The condition C $\in$ {1,0,#} specifying input states for the classifier
- The action A $\in \{a_1, a_2, \ldots, a_n\}$ specifying the action
- The prediction p estimates the payoff expected if the action proposed by this classifier is performed
- The prediction error ϵ estimates the error made in the predictions
- The classifier fitness F
- The experience exp is the number of times that the classifier has belonged to an action set
- The time step ts of the last time the classifier was in an action set where genetics occurred
- The action set size as which holds the average size of the action set where this classifier belonged
- The cardinality n represent the number of classifiers (or micro-classifiers) subsumed by this one (macro-classifier)

Covering is performed anytime the number of actions in the match set $[M]$ is below a given threshold θ_{mna} and then the covering module creates a classifier matching the current state with a probability $P_{\#}$ of having a don't care symbol in any variable position.

Once the match set is formed, XCS makes a "best guess" prediction of the payoff for each possible action and stores it in an array called Prediction Array (PA). This "best guess" is computed as:

$$PA_{action_i} = \frac{\sum_i p_i * F_i}{\sum_i F_i}$$

When an action is not included in a match set, its prediction is set to $null$.

XCS uses a form of Q-Learning for credit distribution. It updates the parameters associated to the classifier in the order exp, ϵ, p, as, and then F, although ϵ could be updated after p. This would lead to faster learning in simpler problems, but misleading in more complex ones.

Finally, the genetic algorithm triggers when a given average time is passed since the members of the action set last participated in a genetic activity. It runs over the action set selecting two rules by roulette wheel and combining them with crossover or mutation and inserting the new classifiers in the population, eventually making room for them; this implements genetic pressure.

A. FIXCS design

As a reference to our implementation of the FIXCS algorithm, we referred to the algorithm description published by Wilson and Butz in 2001 [12]. Also, the implementations of XCS in Java (Butz) and C (Lanzi) have been a relevant source of consultation. These can be downloaded from the IlliGAL (Illinois Genetic Algorithms Laboratory) site at *http://www-illigal.ge.uiuc.edu/index.php3*.

In this section, we will reconsider all the critical points we have presented before, showing what we have done to implement FIXCS.

1) Input and Output Interfaces: The I/O interfaces are the same as in the LFCS model.

2) Matching & Cover Detection: The Matching module is the same as in LFCS, as well. Cover Detection would be called for each of the subpopulations of the Match Set that would have a number of actions less than θ_{mna} (the XCS parameter for the minimal number of actions present in a Match Set). "Don't care" symbols are inserted in place of fuzzy variable values as in XCS, to deal with generalization.

3) Rule Activation: Already having a match set formed by multiple subpopulations, to solve the rule activation issue as much likely to XCS as possible, we have introduced a PA for each of the subpopulations. Everything else would work the same: there would be the same PA formula for each prediction array, and the exploration policy would select an action for each subpopulation, thus forming the action set. General rules eventually covering different subpopulations included in M participate to teh selection mechanisms as any other rule.

We have implemented the exploration policy in doing the classifier selection as suggested in [12]: biased exploration

with a probability p_{expl} of choosing either pure exploration or pure exploitation, at each step.

4) Credit Distribution: XCS uses a type of Q-Learning for updating the classifier parameters. Although the basic pattern is the same as in regular Q-Learning, in XCS classifier predictions are updated using the discounted maximum payoff anticipated from the next PA: $P = \rho + \gamma * PA_{max}$. But as we are considering a fuzzy system, we have different subpopulations and different PAs. We take the same approach proposed for LFCS:

$$m_t(s) = \sum_{k=1,K} \mu_k^*(s) * PA_{max_k},$$

where $\mu_k^*(s)$ represents the degree of matching of subpopulation k in state s and PA_{max_k} is the maximum value of the PA of subpopulation k. This parameter $m_t(s)$ represents a weighted sum of all the maxima of each subpopulation, and will be used instead of PA_{max} to calculate payoff: $P = \rho + \gamma * m_t(s)$.

Considering that we are building a fuzzy system, the reinforcement distribution has to be changed accordingly; we consider also the factor ξ, which controls the contribution of each classifier $\mu(s_t)$ (degree of matching of a fuzzy classifier to a state s in time t) adapting to the learning process evolution by weighting it by the sum of the contribution until now up to a minimum threshold T (to help the young classifiers with few history).

$$\xi = \frac{\mu(s_t)}{\min\left(T, \sum_{k=1,t} \mu(s_k)\right)}$$

Now we are ready to see how each classifier parameters can be updated given a learning rate β and parameters α, ϵ_0 and ν . The updating of exp, ϵ, p and F needs to follow an order as they are related, whereas the as can be executed at any point in time. We will follow here the same order proposed in [12], which can be seen in figure 3.

There is only one more thing to mention, although it works the same for fuzzy or crisp XCS. It is also frequent to use MAM (*Moyenne Adaptive Modifiée* [13]), a two phase technique that makes the system less sensitive to initial, possibly arbitrary, settings of the parameters. According to the (MAM), as long as a classifier is less experienced than $\frac{1}{\beta}$, the previous values are just averaged instead of using the above reported equations. Basically, this means that whenever a classifier verifies $\frac{1}{\beta} > exp$, the learning rate β shall be replaced by $\frac{1}{exp}$.

V. Experimental Results

We decided to perform first tests about the performance of FIXCS on the well-known cart-pole problem. This is a classic example of inherently unstable system. It has often been used to test new approaches to learning control and has a nice fuzzy representation.

Rules are represented as strings of literals, each representing a fuzzy value for a variable. In this example, we have

1) `Experience:` $exp = exp + 1$.
2) `Prediction:` $p_j = p_j + \xi_j * \beta * (P - p_j)$
3) `Prediction Error:` $\epsilon_j = \epsilon_j + \xi_j * \beta * (|P - p_j| - \epsilon_j)$
4) `Action set size:`

$$as = as + \xi_j * \beta * \Big(\sum_{classifier \in [A]} cardinality_{classifier} - as \Big)$$

5) `Fitness:`
   a) `Accuracy of the classifier:`

$$k_j = \begin{cases} \alpha * \left(\frac{\epsilon_0}{\epsilon}\right)^{-\nu} & \text{if } \epsilon \geq \epsilon_0 \\ 1 & \text{if } \epsilon < \epsilon_0 \end{cases}$$

   b) `Relative accuracy to all the classifiers in same the action set:`

$$k'_j = \frac{k_j * n}{\sum_i k_i}$$

   c) `The actual Fitness of the classifier:`

$$F_j = F_j + \xi_j * \beta * (k'_j - F_j)$$

Fig. 3. Reinforcement update for XCS Q-Learning, used also in FIXCS.

five literals: two for cart position and velocity, other two for pole angle and its angular velocity, and the last one for the force applied to the cart (the output of this problem). Each of the input variables can take one of the values corresponding to one of the fuzzy sets defined on the corresponding base variable, respectively 5 for the cart position, 3 for velocity, 7 for the pole angle, 3 for the angular velocity. The output can take one out of two values, corresponding to singletons for positive and negative force. The crossover probability is 0.8, the mutation 0.04.

We have performed 8 experiments. Each *experiment* consists of 200 *trials*. Each trial starts with the pole in a random position within a given degree range (± 5 deg), and the cart in a random position on the trail, within the admissibility range. The trial ends either when the pole reaches the end of the trail or a critical angle, or when a given number of control steps are performed with the pole still in equilibrium.

The evaluations that we present in this paper are aimed at comparing FIXCS with its predecessors: XCS, LFCS and a regular, crisp LCS. We are interested in the evaluation of two aspects: the quality of the rule base at the end of each trial, and the quality of learning.

To evaluate the quality of the rule base through the trials, at the end of each learning trial an exploitation trial is performed with the best rules so far obtained, and the learning mechanism deactivated. This kind of evaluation allows us to measure the actual quality of the learnt rulebase, better than the usual accumulated reward averaged over a window of trials, which is biased by the exploration strategy. For instance, if we had pure random exploration, the second

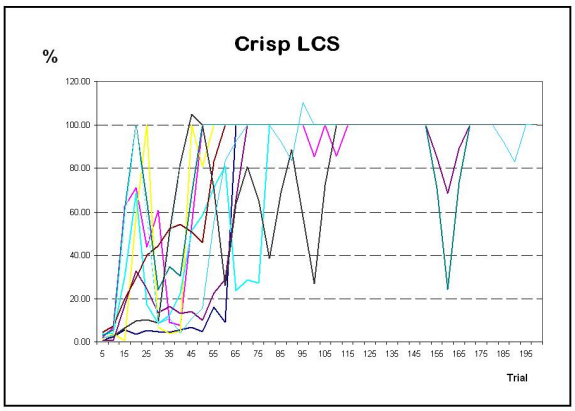

Fig. 4. Performance evaluation - Crisp LCS.

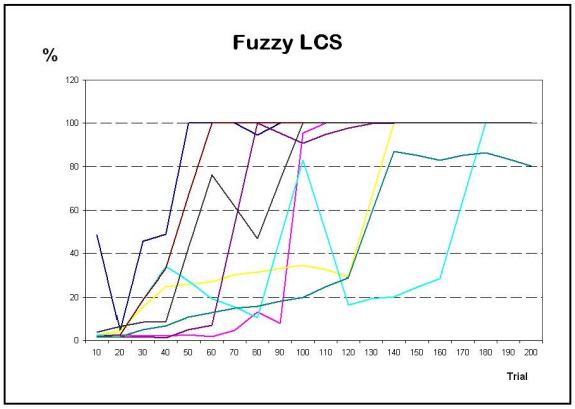

Fig. 5. Performance evaluation - Fuzzy LCS.

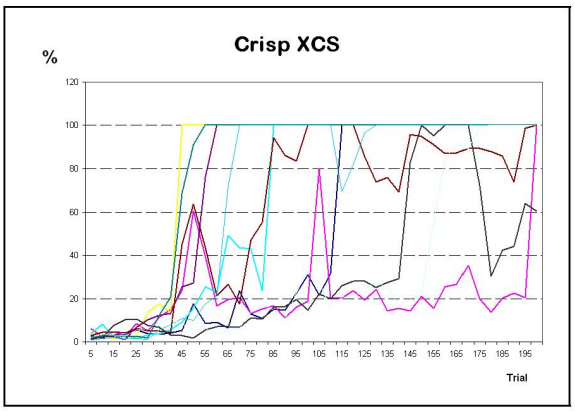

Fig. 6. Performance evaluation - Crisp XCS.

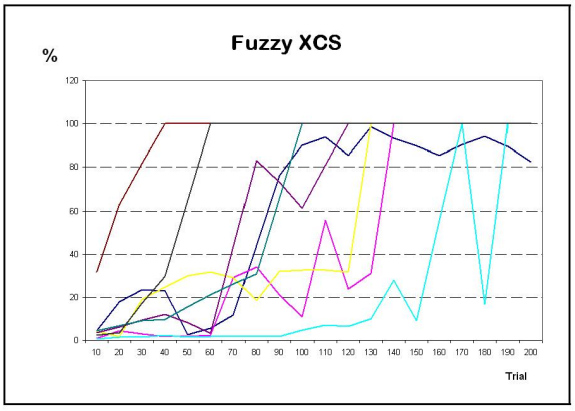

Fig. 7. Performance evaluation - Fuzzy XCS.

measure does not converge, while the policy learnt by the agent is expected to converge to the optimum. Measuring the real behavior becomes the only way to understand whether the algorithm converged to the optimal policy, i.e., the pole stands up.

The results are reported in Figures 4, 5, 6, 7. Each figure concerns the results respectively obtained by the crisp LCS, the fuzzy LCS, the crisp XCS and FIXCS. In each figure, we reported the plots corresponding to the 8 experiments. On the abscissas the number of trial, on the ordinates a percentage of the fulfillment of the task of keeping the pole within a given degree range ($\pm 12\,\mathrm{deg}$) for at least 200 control steps. It can be noticed that all the systems have been able to evolve a stable controller at the end of 200 trials.

Although the quality of the rulebases seems to reach a maximum quite late in the evolution, we can observe the quality of learning in figures 8, 9, 10, 11. In these plots, we show the moving average of rewards obtained in the last 25 learning steps. We recall here that the reinforcement policy for the cart-pole problem distributes -1 to any falling situation and 0 to the others. We can notice that the fuzzy version of both learning classifiers takes longer w.r.t. the crisp one to reach the maximum. However the reward is less noisy, especially in fuzzy lcs, as expected by fuzzy representation.

VI. Conclusions

In this paper we have introduced FIXCS, a fuzzy implementation of the accuracy-based learning classifier system XCS. We firstly remark that using the tools that have been developed through the years about both LFCS and XCS, we were able to achieve a fuzzy implementation of XCS (FIXCS) that respects the essence of both accuracy-based XCS and the cooperation mechanism among rules from LFCS. The first results presented in this paper show that on a standard problem such as cart-pole, fuzzy XCS is able to reach the optimal solution, like XCS and many other systems, but taking more episodes than XCS. This is expected, since the ambiguity and the perceptual aliasing phenomenon are stronger for Fuzzy XCS than for XCS. In experimentes not reported here, as already obtained by LFCS, put in evidence that a slower learning time is counterbalanced by a more robust control system obtained w.r.t. noise and error in sensors and actuation.

We have already implemented a version of LFCS that also learns the positions and shapes of the fuzzy sets, and we are integrating this feature in FIXCS so to have a system able to adapt interpretations of input data in order to optimize its performance.

Another important issue concerns generalization. A discussion about this issue, which, with a fuzzy adaptive model,

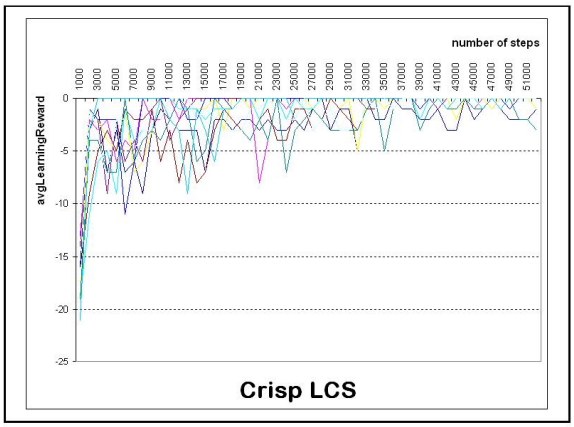

Fig. 8. Learning evaluation - Crisp LCS.

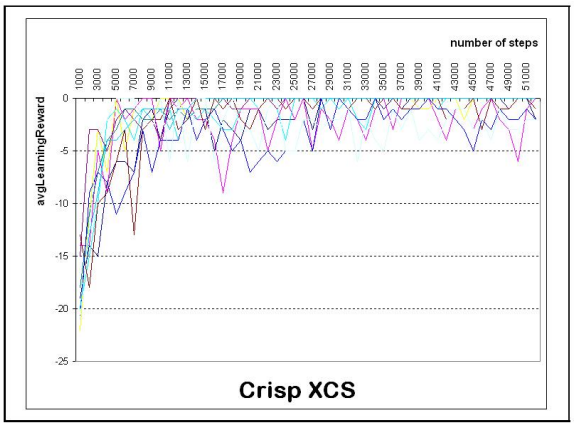

Fig. 10. Learning evaluation - Crisp XCS.

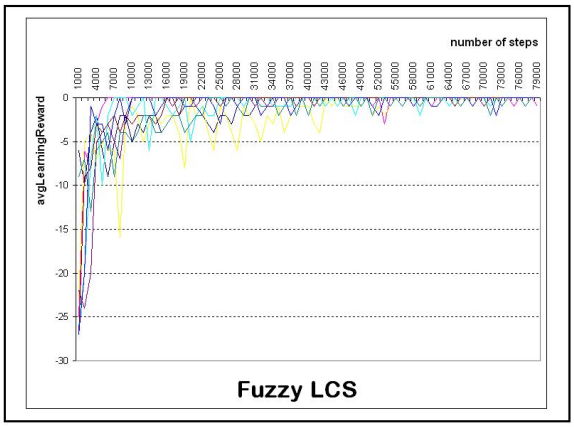

Fig. 9. Learning evaluation - Fuzzy LCS.

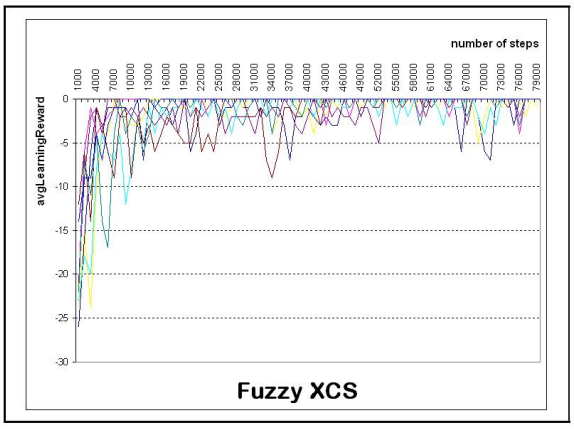

Fig. 11. Learning evaluation - Fuzzy XCS.

becomes more general than usually treated in the XCS community, will be reported in a next paper.

ACKNOWLEDGMENTS

The authors would like to thank Jorge Munoz Vazquez for implementing and testing FIXCS as his ERASMUS thesis.

REFERENCES

[1] J. H. Holland, "Adaptation", *Progress in Theoretical Biology*, vol. IV, pp. 263–293, 1976.

[2] S. W. Wilson, "Classifier fitness based on accuracy", *Evolutionary Computation*, vol. 3, pp. 149–175, 1995.

[3] A. Bonarini, C. Bonacina, and M. Matteucci, "Fuzzy and crisp representations of real valued input for learning classifier systems", in *Learning Classifier Systems, From Foundations to Applications*, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds., London, UK, 2000, pp. 107–124, Springer Verlag.

[4] A. Bonarini, "Reinforcement distribution for fuzzy classifiers: A methodology to extend crisp algorithms", *IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 51–5, 1998.

[5] A. Bonarini, "An introduction to learning fuzzy classifier systems", in *Learning Classifier Systems, From Foundations to Applications*, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds., London, UK, 2000, pp. 83–106, Springer Verlag.

[6] B. Carse and A. G. Pipe, "X-fcs: A fuzzy classifier system using accuracy based fitness. first results", in *Proceedings of the International Conference on Fuzzy Logic and Technology, EUSFLAT*, Intelligent Autonomous Systems Laboratory. University of the West of England, Bristol, 2001, pp. 195–198.

[7] J. Casillas, B. Carse, and L. Bull, "Fuzzy xcs: An accuracy based fuzzy classifier system", in *Proceedings of the XII Congreso Espanol sobre Tecnologia y Logica Fuzzy*, 2004.

[8] P.L. Lanzi, "An Analysis of the Memory Mechanism of XCSM", *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 643–651, 1998.

[9] PL Lanzi and D.E. e Inf, "Adding memory to XCS", *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pp. 609–614, 1998.

[10] L. Bull, "Learning classifier systems: A brief introduction", in *Applications of Learning Classifier Systems*, L Bull, Ed., pp. 3–14. Springer Verlag, Berlin, D, 2004.

[11] S. W. Wilson, "Get real! xcs with continuous valued inputs", in *Learning Classifier Systems, From Foundations to Applications*, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds., London, UK, 2000, number 1813, pp. 209–222, Springer Verlag.

[12] M. Butz and S. W. Wilson, "An algorithmic description of xcs", in *Advances in Learning Classifier Systems: Proceedings of the Third International Conference IWLCS2000*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., Berlin, D, 2000, Lecture Notes in Computer Science, pp. 253–272, Springer Verlag.

[13] G. Venturini, *Apprentissage Adaptatif et Apprentissage Supervise'par Algorithme Genetique.*, PhD thesis, Universite de Paris Sud, 1994.