Learning Fuzzy Classifier Systems: Architecture and Exploration Issues

Matteo Matteucci

May 18, 2000

Abstract

Introducing fuzzy logic in knowledge representation is a general technique to improve flexibility and performances of knowledge based and control software. Many researchers propose to introduce fuzzy logic representation in learning algorithms, discussing some implications of this tecnique. Interesting features arise when fuzzy sets substitute the interval-based classification of input in a learning system; some of them imply an improvement in performance others an increased structural complexity in the architecture of the system and in the learning process. Focusing on Learning Classifier Systems, the introduction of fuzzy logic produces some new interesting features in this learning algorithm from many points of view: a new approach to classifier competition, the birth of competition vs. cooperation dilemma and the introduction of an appropriate fuzzy interface with external world. In this paper, we discuss a fuzzyfication of the classical architecture of a learning classifier system (Holland's approach) and the improvements deriving from the use of fuzzy logic. In this work we especially discuss the competition vs. cooperation dilemma, analyzing the influence of exploration policy on the performance of crisp and fuzzy versions of learning classifier systems. We mainly focus on the use of fuzzy classifier systems to implement behaviors for reactive autonomous agents in the mobile robotics domain.

1 Introduction

The growing interest of part of Artificial Intelligence (AI) community in Fuzzy Logic (FL) and Learning Classifier Systems (LCS) has generated some hybrid systems to join the advantages of both in a new extension of the reinforcement learning (RL) algorithm. Many fuzzy implementations of classic RL algorithms, reflecting this new approach, are presented in literature [14] [6] in this paper, we focus on Learning Fuzzy Classifier Systems (LFCS) analyzing the features of LCS crisp and fuzzy architecture and the issues in the learning process arising in using fuzzy logic in these learning systems. LFCS [23] [9] have been one of the first applications of Evolutionary Learning (EL) to learn Fuzzy Logic Controllers (FLC). FL has been introduced in both the streamlines along with LCS where developing: the Pittsburgh [19] [13] and the Michigan [1] [12] approaches. In the Pittsburgh approach the EL algorithm works on a population of rule bases (FLCs) evolving and evaluating them in parallel; in the Michigan approach, it works on a population of rules in the rule base applying the learning algorithm to sub-population them.

In this paper we describe a complete LCS architecture following the Michigan approach and we propose a modular schema in both crisp and fuzzy versions. We choose to implement LFCS with the Michigan approach to deeply investigate how the features of fuzzy logic face the problem of generating a complex non-linear model smoothly joining some local models implemented by sub-population (i.e. this is the origin of competition vs.cooperation dilemma).

The rest of this section provides a basic background on FL and LCS to introduce the concepts used in the definition of LFCS. In next sections we discuss some motivations for introducing fuzzy logic in knowledge representation and the model of classical LCS architecture discussing its fuzzification. The new features arising from our approach are further discussed with an experimental example in the mobile robotics domain. In this section we mainly focus on the relationship between the well-known exploration vs. exploitation dilemma in RL algorithm and the new competition vs. cooperation dilemma arising in LFCS.

1.1 Fuzzy logic

Since their introduction in the late sixties [27], fuzzy sets have been adopted to map real numbers to symbolic labels. Elements of a universe of discourse belong to a fuzzy set to a certain extent, according to the so-called membership function that defines it. Let us consider an universe of discourse X (say, an interval of real numbers), and a fuzzy set, associated to the label *close*, defined by a membership function $\mu_{close}(x)$ that ranges on elements of the universe of discourse and maps them to the real interval $[0 \quad 1]$ (figure 1).

For instance, using fuzzy sets to classify a real-valued measure for a distance given by a sonar sensor, we can say that an element x of the universe of discourse (the distance measured in mm) belongs to the fuzzy set *close* to the extent $\mu_{close}(x)$. For the fuzzy set *close* in figure 1, $\mu_{close}(690) = 0.7$. Giving a semantic value to the fuzzy set, we can say that the value $x = 690$ can be classified as *close* with a degree of 0.7.

Fuzzy sets are often used to classify real values in categories, thus making it possible symbolic reasoning about the phenomena under the incoming numbers. Membership of a number (e.g. x) to a fuzzy set (e.g. *close*) includes two kinds of information: the name of the fuzzy set, that brings information about the category to which the number is classified, and the degree of membership (e.g., $\mu_{close}(x)$) that comes from the definition of the fuzzy set. The membership value can be considered as an alternative representation of x, although it is unique only for the class of monotonic membership functions. In general, the relationship between a real value x and its degree of membership to a fuzzy set l (i.e., $\mu_l(x)$) is not bijective. This potential problem is solved by defining

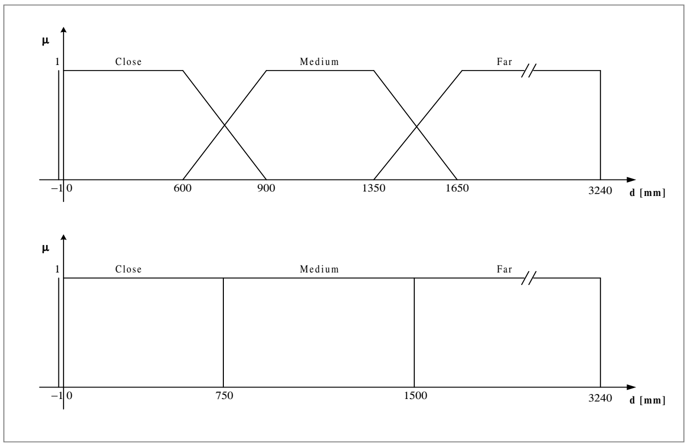

Figure 1: The fuzzy membership function (above) and the intervals (below) to interpret the distance measured by a sonar sensor.

partially overlapping fuzzy sets, and considering as representative of x the set of all the values $\mu_l(x)\ \forall\ l$.

Considering a crisp model and a fuzzy one with the same granularity (i.e. number of intervals or fuzzy sets), the information content of a fuzzy representation is very close to that of a real-valued representation, and considerably higher than that of an interval-based. Moreover, the selection of certain well-known configurations of fuzzy intervals guarantees high robustness to noise and certain design errors in control sofware implemented by FLC [17].

1.2 Learning Classifier Systems

We can consider LCS [15] as RL algorithms [16] working on a model of the system's behavior in the environment, without any explicit model of the environment itself. For a LCS the rule base, containing the classifiers, can be considered as a knowledge base (figure 2) updated by the RL algorithm during system interaction with the environment.

A classifier can be seen as a rule with a condition/message structure. Condition codifies the perception of the state of the system, represented by sensorial perception or an internal message, needed to activate the rule. In the message part, it is possible to code an action to submit, through the output interface, into the environment or an internal message to be used by the classifier system. Such implementation of LCS can learn reactive rule bases and systems with an internal state, represented by the internal message list to face with non-Markovian environments [18]. In this paper we consider a simplified type of LCS [4] using the Michigan approach to implement purely reactive behaviour.

Each classifier is a rule whose antecedents corresponds to a value for a *linguistic variable* associated to a real-valued variable. The information needed by

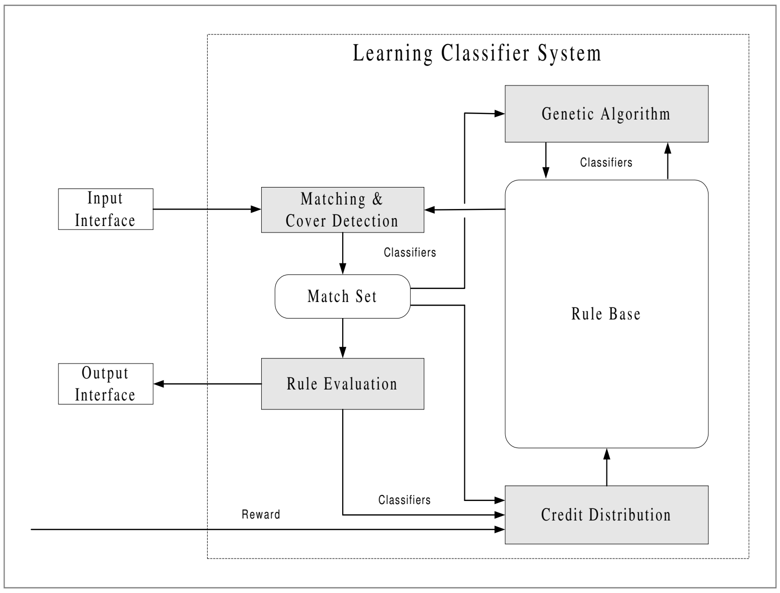

Figure 2: A simplified LCS architecture.

a LCS is usually discretized in order to have classifiers working on a finite set of input and output values. Using an input interface module it is possible to classify the real input values into classes corresponding to either crisp intervals or fuzzy sets. In crisp LCS, we consider that each value is one of the n symbols representing intervals of values of the linguistic variable; in LFCS each value is a symbol representing a fuzzy subset [17] of the set of values the real-valued variable can take. A *don't care* symbol may replace any of the mentioned symbols, meaning that the value of the corresponding variable is not relevant, that is any real value matches the antecedent. The consequents are symbols corresponding to values for each of the consequent variable. A classifier of this kind as a conceptual shape like this:

```
IF (FrontDistance is Low) AND
   (LeftDistance is High) AND
   (RightDistance is Don_T_Care)
THEN (TurnDirection is Left)
```

At each control step, one classifier is selected among the ones whose antecedent matches the current perception of the environment. The consequent value of the selected classifier is sent to the output interface and, by means of an evaluation signal from the environment (the reward), selected rules are rewarded in the Rule Base. In some extent, it is possible to partition our LCS implementation in four principal submodules (figure 2):

- *Matching & Cover detection*: chooses classifiers matching the actual perception from Input Interface to form the Match Set and eventually generates random ones

- *Rule evaluation module*: chooses classifiers in the Match Set evaluating their characteristics and their message part is sent to the Output Interface
- *Credit distribution*: distributes external reward to classifiers which proposed the last action sent to the environment
- *Genetic Algorithm*: genetic operators (crossover and mutation) operate on the Match Set to generate new classifiers in the Rule Base

2 Motivation for LFCS

The main motivation to integrate LCS and fuzzy sets is to combine the advantages of both in a unique RL algorithm. LCS enables a system to learn a knowledge base and optimize it, by interacting with the environment, even when limited information is available. Fuzzy sets, allowing partial overlapping of their membership functions, give the possibility of a graceful sliding classification of continuous values to contiguous classes. Moreover, it can be demonstrated that if the sum of the membership values equals 1 for any value in the range of the variable, the robustness of the system with respect to noise is optimal [17].

Another important benefit, arising when modeling the application domain and the control system by fuzzy sets, is that we have both a relatively small model and the potential to exploit all the precision available from real-valued data. An interval-based model maps a whole range of real values onto a unique interval, denoted by a label. In other terms, the control system considers in the same way all the values belonging to the interval. A fuzzy set model maps a set of real values onto a potentially infinite number of pairs $< label, \mu_{label} >$ that bring information about both the class to which belongs the real value and the degree of this classification. This is much more informative than a simple interval mapping.

For instance, referring to figure 1, we can observe that with the fuzzy classification the value 950 mm is considered as completely *medium*, $\mu_{medium}(950) = 1$; the nearby value 750 mm is still considered as *medium* with a degree $\mu_{medium}(750) = 0.5$, but also considered as *close* to some extent, $\mu_{close}(750) = 0.5$. With an interval-based classification such as that shown in figure 1 (below), which is commonly adopted in LCS, the value 751 mm is considered *medium* and the close value 749 mm is considered *close*. This is in contrast to the common-sense interpretation of these classes and may result in abrupt changes of the action taken by a control system based on this classification. In fact, because more than one fuzzy state may be visited at the same time, we have a smooth transition between a state and its neighbors and, consequently, smooth changes in the actions. This is desired in many behaviors for autonomous agents [1] [2] as well as in control application [20].

When the fuzzy set approach is used to model a control system [10], usually we have a set of fuzzy rules mapping fuzzy values of input variables to fuzzy values of output variables. The resulting FLC has many interesting properties, such as robustness, smoothness of action, and wide range of applicability [17]. Adopting a fuzzy representation, it is also possible to integrate expert knowledge into the control system. Moreover, prior knowledge derived from another LFCS learning session or an human expert, can be used to initialize the RL algorithm starting the learning process.

It is common to operate with RL in simulated environments, modeled on simple assumptions about the sensorial input; fuzzy controllers have proved to be robust enough to compensate differences between simulation and reality once applied on real agents interacting with a real environment [14].

3 Classic architecture

In this section, we briefly discuss our LCS architecture and introduce a modular approach to understand the real interplay between the learning components. Using this approach we underline factors that interact in forming the knowledge base containing the rules governing the agent's behavior. In figure 2 modules belonging to the learning algorithm are represented with rectangles, modules containing classifier sets with rounded rectangles and the information flows with labeled arrows. As previously described, the architecture of the LCS considered in this paper consists of four main modules working on the Rule Base and on the Match Set:

- Matching & Cover Detection
- Rule Evaluation
- Credit Distribution
- Genetic Algorithm

information from the environment is classified by an Input Interface in intervals and the action proposed by the selected classifier is sent to the Output Interface at the end of the inference cycle. In the next paragraphs we describe the implementation of each module operating on the classifier sets and the algorithms cooperating to the RL process in the Credit Distribution module.

3.1 Matching & Cover Detection module

The Matching module selects classifiers in the Rule Base matching the actual perception of environment's state sensed by Input Interface with their condition part. Selected classifiers form the Match Set. When no classifier matches with its condition part the actual state the Cover Detection module generates at least one classifier with a condition part matching the perception and a random

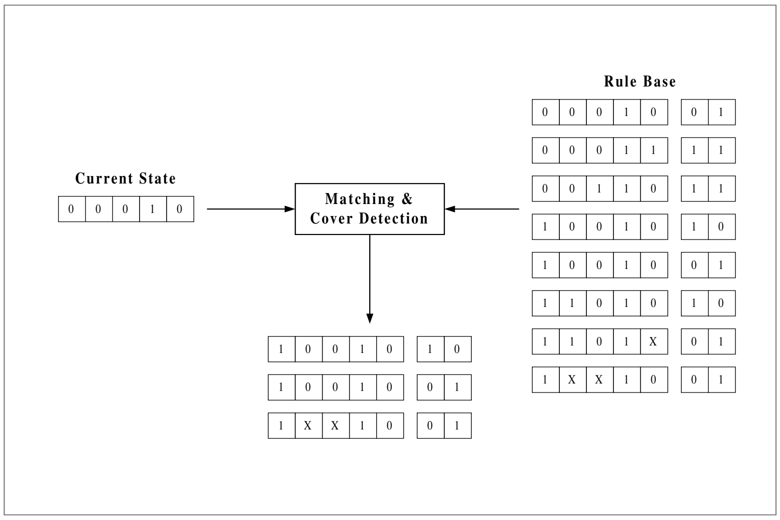

Figure 3: Match Set formation.

message part. Moreover, it is possible to set a minimum number of classifier to be present in the Match Set after Matching & Cover Detection.

Figure 3 reports the formation of a Match Set from the state sensed by the Input Interface and the classifiers in the Rule Base; in this simple example, labels used by the crisp interface are 1 and 0 the label x stands for *don't care*. The *don't care* condition in a classifier allows the matching of corresponding variables independently of their value. Generalized rules, containing *dont' care* symbols in their antecedent, compact the Rule Base and enable the reactive behavior of the agent to face, at least, unknown states of the environment [26].

3.2 Rule Evaluation module

The Rule Evaluation module selects the action to be sent into the environment among the classifiers actually present in the Match Set. This module chooses a classifier by evaluating a particular rule feature: its fitness. This feature, updated by the Credit Distribution module, is a measure of action's quality (i.e., message part of it) proposed for the actual state (i.e., agent perception). Different choices are possible to represent the fitness for a classifier, two of them are:

- *strength*: fitness is an estimation of the future expected reward earned by the classsifer
- *accuracy*: fitness is inversely proportional to the prediction error of the action proposed by the classifier (it estimates the lack of knowledge)

The policy applied in the Rule Evaluation module in choosing rules has to deal with the so-called *exploitation vs. exploration dilemma*. The deterministic choice of strongest classifiers, *exploitation*, does not guarantee a correct learning process, since, in this case, the system continues to evaluate the quality of few rules and nothing is done to consider rules generated by the Genetic Algorithm module and never tested. Vice versa, by always choosing the less tested rules, *exploration*, it is possible to cover widely the search space but without refining the learnt policy exploiting it. These two approaches in choosing rules from the Match Set are the extremes of a continuous set of possibilities; we need a trade-off to obtain satisfying results.

Many exploration policies introduce probability $Pr(c^*)$ in doing the classifier selection to accomplish with the exploration vs. exploitation dilemma, some of them are:

- *Random choice*:

$$Pr(c^*) = \begin{cases} \frac{N-1}{N}, & \text{for the strongest classifier} \\ \frac{1}{N}, & \text{for a random classifier (also the strongest)} \end{cases}$$

- *Montecarlo*: given the strength s for a classifier

$$Pr(c^*) = \begin{cases} 0, & \text{if } s = 0 \\ \frac{s(c^*)}{\sum_{c \in MatchSet} s(c)}, & \text{else} \end{cases}$$

- *Boltzman*: given the strength s for a classifier, it uses the temperature T to regulate the exploration level

$$Pr(c^*) = \frac{e^{\frac{s(c^*)}{T}}}{\sum_{c \in MatchSet} e^{\frac{s(c)}{T}}}$$

- *Mixed strength and accuracy random choice*:

$$Pr(c^*) = \begin{cases} \frac{N_{ac}-1}{N_{ac}}, & \text{for the strongest classifier} \\ \frac{1}{N_{ac}}, & \text{for the less accurate classifier} \end{cases}$$

3.3 Credit Distribution module

The Credit Distribution module updates, using the reinforcement signal (reward), the fitness of active classifiers in the Rule Base whose action has been proposed to the environment. The credit apportionment faces a structural and temporal problem: the reward is distributed to active classifiers, either generalized (i.e., with *don't care* symbols in the condition part) or not, proposing the same action (structural distribution) and back-propagated to the previous active classifiers to build temporal chains of rules (temporal distribution).

Bucket Brigade [8] was the algorithm originally proposed for credit assignment in LCS, it uses the concept of *Action Set*. In our implementation, the Action Set consist of the classifiers in the Match Set proposing the same action sent to the environment. At each inference cycle the algorithm executes these steps:

- creates the Action Set A_t with classifiers proposing the chosen action a_t
- reduces the strength s of each classifier c_t belonging to A_t according to:

$$s'(c_t) = (1 - b) \cdot (c_t) \qquad 0 < b \ll 1,$$

- distributes to classifiers in A_{t-1} (i.e., the previous Action Set) the strength subtracted to the classifiers in A_t. To update the active classifiers' strength it applies:

$$s'(c_{t-1}) = s(c_{t-1}) + \frac{1}{n} \sum_{c \in A_t} b \cdot s(c_t) \qquad n = \text{number of classifier in } A_{t-1},$$

- distributes to classifiers in A_t the reward r_t obtained executing a_t

$$s''(c_t) = s'(c_t) + r_t.$$

Bucket Brigade is not a defined standard and there is not a rigorous demonstration of convergence for this algorithm, the implementation presented in this section is only one out of the possible; many versions of this algorithm implement more sophisticated criteria in strength subtraction and redistribution.

Recently Q-Learning [24] has been applied to the credit apportionment in a LCS; the strength of a rule matches the Q-value of the corresponding State-Action pair [25]. To each classifier c_t belonging to the Action Set A_t the Credit Distribution module applies:

$$q'(c_t) = q(c_t) + \alpha(r_t + \gamma \cdot \max_{c_{t+1} \in A_{t+1}} (q(c_{t+1})) - q(c_t))$$

$$0 < \alpha < 1 \wedge 0 \leq \gamma < 1.$$

In [5] and [7] also Credit Distribution algorithm based on TD(λ) [21] [22] for LCS is proposed, but its description is not in the scope of this paper.

3.4 Genetic Algorithm module

The Genetic Algorithm module applies genetic operators to evolve classifiers in the Rule Base substituting the worst ones. Best classifiers, from the fitness point of view, are selected as parents and a new generation of classifiers is created applying crossover and mutation operators: new classifiers will compete with other ones in the rule base to survive. The main features of this implementation of the genetic algorithm are:

- Selection criteria
- Population's choice
- Crossover and mutation operators

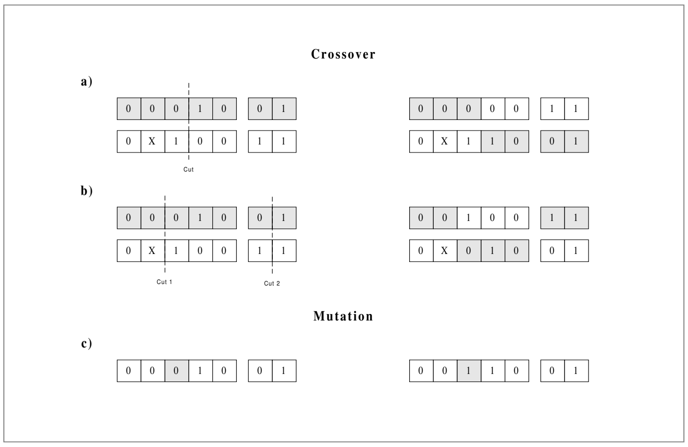

Figure 4: Examples of genetic operations.

In our approach, the probability of selection for a classifier is proportional to the normalized strength: the ratio between the classifier strength and the average strength of classifiers in the selected sub-population (classifiers with the same condition part). Accuracy is another possible choice as fitness value [25], it can be useful to avoid the growth of too many imprecise, generalized classifiers.

To implement a genetic algorithm we have to select a population; we present here two different possibilities: selecting all the classifiers in the Rule Base or the classifiers in the Match Set. The first approach uses genetic algorithm to optimize globally the Rule Base, but does not consider that classifiers refers to different parts and context of the search space, so classifiers with low fitness may represent the best knowledge for a certain part of the environment. The second approach uses niche genetic [8], where the population selected for genetic manipulation is only the Match Set, to avoid the above mentioned problem.

Once selected n classifiers to *"reproduce"*, the genetic algorithm applies genetic operators to their copies executing *crossover* with probability P_{cross} on every $n/2$ pairs of them (figure 4a, 4b) and *mutation* with probability P_{mut} considering that every *allele* (i.e., a cell in the figure) has probability P_{allele} of being mutated (figure 4c).

4 Fuzzy architecture

The architecture of LFCS presents some differences with respect to LCS especially in the inference cycle; these differences introduces the necessity of a deeper analysis comparing fuzzy architecture to the classical one.

The fuzzy inference cycle starts by classifying the input, usually real values, computing the degree of matching of each variable to corresponding the fuzzy sets. Then, fuzzy operators [11] are used to combine the matching degrees of the different variables obtaining a matching degree for the state described by all the input variables. At this point, all the matching rules are triggered; each one proposes an action with a weight that depends on the degree of matching of the rule. The output comes from the aggregation [11] of all proposed weighted outputs and the global output is defuzzyfied to a real value.

The keys of this process are input/output fuzzyfication/defuzzyfication, obtained through the Input and Output Interface modules, and the fuzzy matching executed in the Matching & Cover Detection modules. In the next parts of this section, we describe the fuzzy implementations for the crisp modules previously described, we do not present Genetic Algorithm module since, working on labels, it does not differ from crisp one.

4.1 Input and Output Interface modules

The Input Interface module classifies the real-valued sensorial input in a fuzzy perception of the environment. This perception is computed considering each input variable configuration and using fuzzy operators to combine matching degrees. This interface combines the input fuzzy values (pairs of $< l, \mu_l(x) >$) in a fuzzy vector characterized by a global matching degree. Common operators used to calculate the global matching degree are *minimum* (that makes a conservative choice) and *product* (that considers all the components), being both *T-norms* [11] implementing a fuzzy logic conjunction.

The membership functions used in our implementation for input classification are triangular or trapezoidal, overlapping in pairs, as shown in figure 1. Using labels in definition of sets for the condition part, the main difference between crisp and fuzzy classifiers is the matching degree used by the fuzzy version to compute rule pertinence in a particular configuration.

Menbership function are also used by the Output Interface module to combine proposed actions in a unique output message. In the Output Interface module the actions proposed by the selected classifiers are composed in a real-valued output using methodology derived from fuzzy control. Classifiers in the Match Set proposing the same action are aggregated by a fuzzy operator (*T-Conorm* [11]) obtaining pairs $< y, \mu(y) >$ whose elements are the output fuzzy set and the corresponding matching degree. To defuzzify the proposed output it is possible to implement the center of gravity (COG) method obtaining the action A:

$$A = \frac{\int y \cdot \mu(y) \cdot dy}{\int \mu(y) \cdot dy} \qquad \text{for the continuous case}$$

$$A = \frac{\sum_{\forall i} \mu(y_i) \cdot y_i}{\sum_{\forall i} \mu(y_i)} \qquad \text{for the discrete case}$$

where in the discrete case the output fuzzy sets are represented by singletons (a common choice in real implementations).

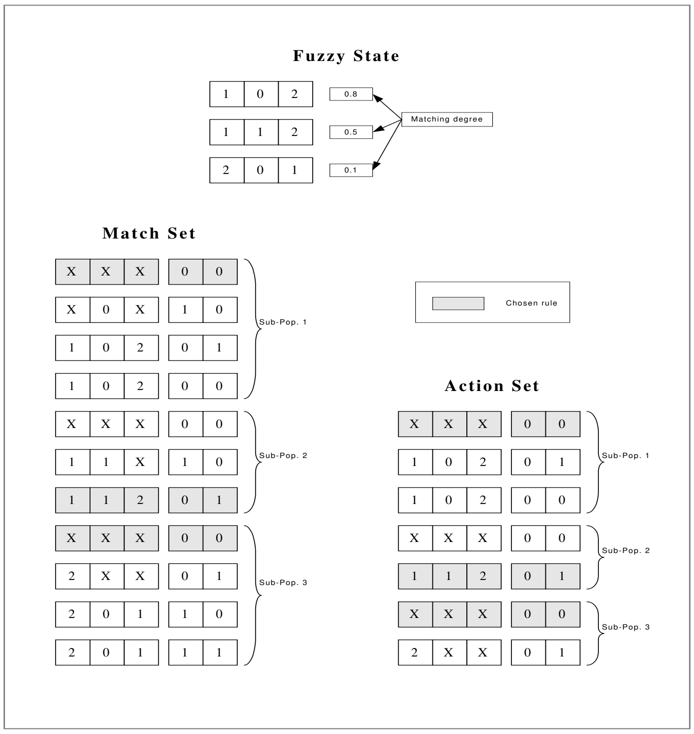

Figure 5: Match Sets, Action Sets and selected classifiers for different sub-populations.

4.2 Matching & Cover detection module

The Matching module selects classifiers in the Rule Base that match, by the fuzzy labels, the real-valued states sensed by the Input Interface. We notice how LFCS operate on sub-population of classifiers that match a different perception generated by the fuzzy interface. In fact, the real value of an input variable can be classified as belonging to at most two different fuzzy sets, so a configuration of the environment can be classified by almost 2^n state vectors, were n is the number of variables forming that vector.

Classifiers selected by the the Matching module, form the Match Set, they can be associated to sub-population in the Rule Base. Generalized classifiers may belong to two or more sub-population in the Match Set (figure 5). Sub-populations play a role in the output determination proportional to the matching degree of the corresponding fuzzy state, in fact the matching degree of each selected classifier is used by the output interface to combine the proposed actions in a real valued output.

The main difference between LFCS and LCS implementation of Matching module, is the presence of many sub-populations in the Match Set at the same time that have to cooperate in forming a unique output action. This presence of sub-population is the cause of the so-called *cooperation vs. competition dilemma* [3]. In fact, classifiers belonging to the same sub-population of the Rule Base have to compete to survive, but, at the same time they have to cooperate with classifiers of other sub-population in forming a correct output. In this paper we mainly focus on this problem and its correlation to the exploration vs. exploitation dilemma proposing an analysis of their relationship in the experimental section. The Cover Detection module acts exactly the same for both LCS and FLCS.

4.3 Rule Evaluation module

Also in FLCS the policy applied to select rules has to deal with the exploitation vs. exploration dilemma. As for LCS, it is possible to implement different selection policies applying different criteria on the fitness value.

To evaluate classifiers, we consider rules proposing the same action as belonging to an Action Set like in the crisp version. The presence of many sub-population in the Match Set generates different subpopulation in the Action Set; the Rule Evaluation module selects a rule from each subpopulation proposing it to the Output Interface module to be defuzzyfied.

As previously discussed the introduction of sub-population in the Rule Base generates some problems to the learning process due to the cooperation vs. competition dilemma. The learning algorithm as to search a correct combination among local model of desired behaviour implemented by sub-population. The need of learning also this coordination, implies to explore also this dimension of the problem. We deeply discuss this difference and how exploration level influences LCS and LFCS in section 5 with experimental results.

4.4 Credit Distribution module

As previously highlighted, the fuzzy version of Credit Distribution module has to deal with the presence of sub-population in th Rule Base, distributing reinforcement to the rules proportionally to their contribution to the performed action. Using the strength of a classifier to represent its fitness it is possible to update it using the formula:

$$s'(c_t) = s(c_{t-1}) + (r_t - s(c_{t-1})) \cdot \frac{currc_c}{pastc_c}.$$

This formula increments classifer's fitness by a quantity proportional to the difference between the present reinforcement and the past fitness, multiplied by the contribution of the classifier to the actions just performed $currc_c$ and weighted by the sum of past contributions $pastc_c$ [1] [3]. This process is needed to take into account the nature of the fuzzy inferential algorithm, in which a rule contributes to the global action proportionally to its degree of matching with the current state. The current contribution $currc_c$ is a measure of how much the action proposed by this rule has contributed to the action actually done. It is computed by the following equation, in this formula c is the classifier under examination, and C is the set of triggering rules

$$currc_c = \frac{\mu(c)}{\sum_{c \in C} \mu(c)}$$

At each rule activation, $pastc_c$ is updated adding the current contribution $currc_c$ to the old value of $pastc_c$, until a given threshold is reached and the classifier has been tested enough. It is possible to apply the previously discussed formulas to the crisp algorithm Bucket Brigade modifying its four steps as follows:

- create the Action Set A_t^i (i stands for the sub-population) with classifiers proposing the chosen action a_t^i
- compute *bid* value considering classifiers belonging to each sub-population in the Action Set:

$$bid = \sum_{\forall i \wedge \forall c \in A_t^i} b \cdot currc_c(c_t) \cdot s(c_t) \qquad 0 < b \ll 1,$$

- reduce the strength s of each classifier c_t belonging to A_t^i according to

$$s'(c_t) = (1 - b \cdot currc_c(c_t)) \cdot s(c_t) \qquad 0 < b \ll 1,$$

- distribute to classifiers in A_{t-1}^i (i.e., previous sub-populations of the Action Set) the strength subtracted to the classifiers in A_t^i and the reward. To update the active classifiers' strength it applies:

$$s'(c_{t-1}) = s(c_{t-1}) + \frac{currc_c}{pastc_c} \cdot (bid + r_t)$$

It is also possible to implement a fuzzy version for the Q-value updating formula:

$$q'(c_t) = q(c_t) + \frac{currc_c}{pastc_c} \cdot \alpha(r_t + \gamma \cdot max_{c_{t+1} \in A_{t+1}}(q(c_{t+1})) - q(c_t))$$

$$0 < \alpha < 1 \wedge 0 \leq \gamma < 1.$$

5 Experimental results

The application we select to analyze FLCS consist in learning a reactive behaviour for a mobile robot moving across a corridor. The learning activity is done on a simulated model of the actual mobile robot CAT [3], which has a car-like kinematics, is 700 mm long, 600 mm wide, and can run at a maximum speed of 300 mm/sec. The maximum steering degree is 30° on each side, and this is an important limitation in the maneuvering capability to be learnt. In the

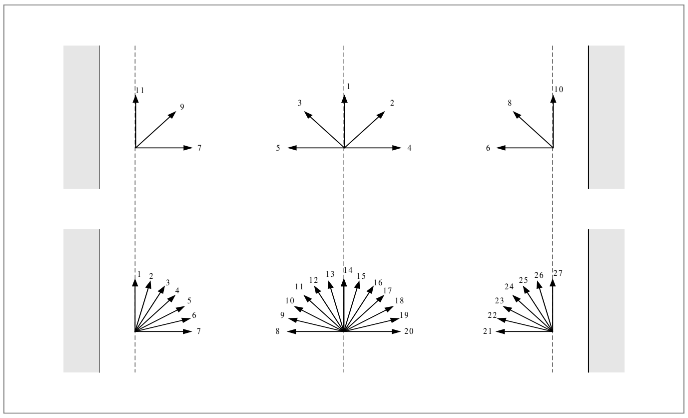

Figure 6: Starting positions during learning (above) and test (below) experiments.

configuration we adopt for these experiments, CAT has 8 bumpers (i.e., on/off contact sensors) all around its body, and 6 sonar sensors, covering the frontal 210°. Each sonar produces an ultrasonic wave and the time between emission and detection of a reflected wave (i.e., Time of Flight - ToF) is measured. This is proportional to the distance from the closer surface orthogonal to one of the rays of a 60° cone originating from the sonar. The range is between 200 and $3,240$ mm. The distance obtained by each sonar is affected by noise, uniformly distributed in the interval $\pm 5\%$ of the maximum range. This model is rough, but realistic, and corresponds to a couple of two sensors available on the market, each having a detection cone of about 30°. Data from sonar is interpreted either in terms of fuzzy sets, or in terms of crisp intervals, as previously described in figure 1. Notice the special singleton value used to represent a characteristic feature of sonars: when no echo returns within the maximum ToF, the sensor gives a specific out of range value. This happens either when no object is within the range, or when all the objects within the range deflect the ultrasonic wave away from the sensor, due to the relative direction of their surfaces.

In our experiments, the perception of the environment state is thus represented by six variables, each corresponding to a distance measured by one sonar sensor, and one *boolean* variable, which becomes true when any of the bumpers is activated. The only output variable is steering, represented by singletons for seven possible choices: $-10°, -20°, -30°, 0°, +10°, +20°, +30°$. and velocity is fixed to 300 mm/s forward. In the experiments presented here, we consider the Q-Learning version of Credit Distribution module. To test relationship between exploration vs. exploitation dilemma and learning problems due to sub-population in the Match Set we use Random choice as exploration strategy, analyzing the influence of different values of parameter N. In the Genetic Module we have excluded generalization, and, with it, the need for crossover: the classifiers are generated only by cover detection and mutation is limited to

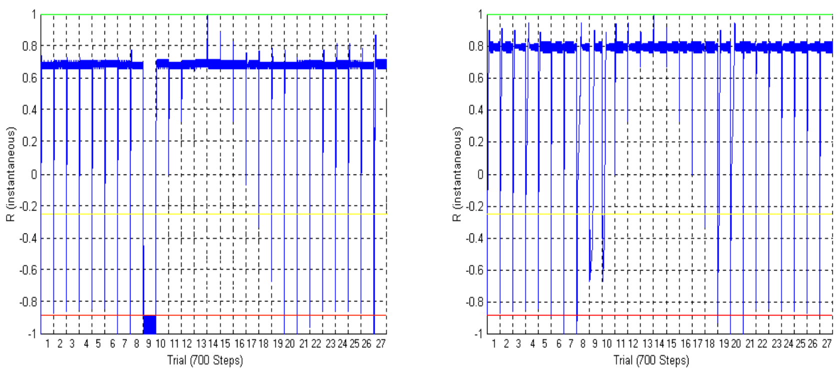

Figure 7: Crisp (left) and Fuzzy (right) performances in 27 test trials.

the consequent part. The number of classifiers in a sub-population matching a state is inversely proportional to the top performance of the sub-population: the worst classifiers matching the current state are deleted when the sub-population performance increases.

The reinforcement function r for the proposed task, is the same in all the experiments, and it is linear with orientation and distance from the center of corridor; a punishment is given when robot is bumping on the walls. Each learning trial lasts 500 control steps, and sets of trials to be compared with each other have the same randomization seed.

The first experiment consists of trials sequentially starting from 11 different positions; each sequence is repeated 10 times for a total of 110 trials and 55,000 control steps (figure 6(a)). From the set of classifiers we have at the end of the learning process, we consider only the best classifiers for each sub-population that have been tested enough. We evaluate the performance of the controller they implement in 27 test trials, lasting 700 control steps each, starting from the 11 positions of learning session and from 16 intermediate ones (figure 6(b)), in order to test the generality of the learnt controller [7]. To have a measure of the learnt behaviour's performance we evaluate it with the same reinforcement function used during the learning experiment.

In figure 7 we show the instantaneous performance, over the above mentioned 27 trials, of a learnt controller whose control step lasts 600 ms. The behavior is almost satisfactory for both LCS and LFCS, although the latter shows a better behavior, since its performance over the 27 trials is higher, and it does not bring the robot against the walls, as it happens with LCS when starting from position 9.

A second experiment tests the influence of using fuzzy logic in knowledge representation from exploration vs. exploitation point of view. We analyze the same experimental setting were the learning trials start every time from the

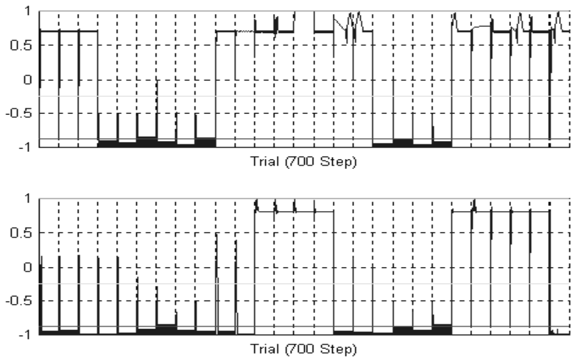

Figure 8: Crisp (above) and Fuzzy (below) performance with 20% probability of random choice ($N = 5$).

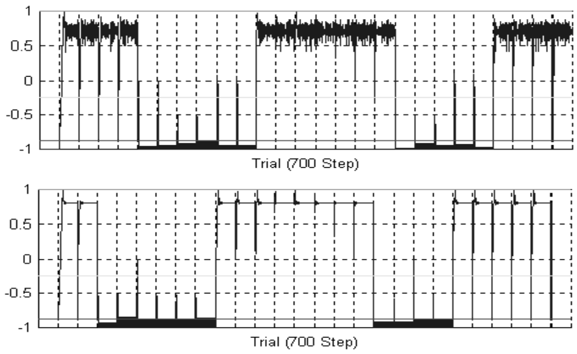

Figure 9: Crisp (above) and Fuzzy (below) performance with 50% probability of random choice ($N = 2$).

central position (the 1^th). We expect the agent to learn a general behaviour by exploring autonomously the unknown environment by a random choice of classifier in the Match Set. Our purpose is to understand limitation introduced in exploration and generalization by the cooperation vs. competition dilemma.

Observing results of this second experiment, we can highlight how FLCS needs more exploration to deal with the complexity of the learning process due to presence of cooperation vs. competition problems. In figure 8 are represented the performances of LCS and LFCS using the same exploration level of first experiment: $N = 5$ in random choice of strongest classifier in the Match Set. The performance with respect to previous experiment reflect a lack of generalization due to the fixed starting position. Moreover, it is possible to notice, in figure 8, how crisp LCS obtains better results with this level of exploration than FCLS.

To test how exploration level influence the learning process in LFCS we repeated the same experiment using $N = 2$ in the random choice policy. As plots in figure 9 reflect, in this case LCS and FLCS results do not differ substantially. Giving more exploration to crisp LCS causes worst performances according to the exploration vs. exploitation dilemma. Moreover, comparing figure 8 above (LCS with $N = 5$ in random choice) and figure 9 below (LFCS with $N = 2$ in random choice) it is possible to notice that crisp LCS and FLCS obtain qualitatively the same generalization performances when applying more exploration to FLCS.

6 Conclusion

In this paper we presented an approach to define Learning Fuzzy Classifier Systems (LFCS), based on the extension of architecture and algorithms defined for crisp Learning Classifier Systems (LCS). We discussed some of the properties of this new class of learning systems, the problems that a fuzzy knowledge representation introduces in learning, and the performance quality that can be obtained. We have used a tool implementing the described architecture to learn reactive behavior for an autonomous mobile agent avoiding obstacles and moving across a corridor. The performance analysis of learnt behavior highlights some interesting features concerning complexity of the learning process and LFCS performances.

The need of learning also cooperation among fuzzy classifiers produces a larger search space where the algorithm should find the correct combination of triggering classifiers. To face this problem we suggest to increase the level of exploration in selection of classifiers in the Match Set. We tested learnt controllers in the environment and we noticed that performances deriving from best fuzzy controllers are better than ones deriving from best crisp controllers.

We do not look at the presented architecture as an isolated experiment on FLCS; we consider this paper also as a proposal to implement modular learning systems; modularity makes possible to isolate module features to understand their interplay in LCS learning process. We also suggest trying new implementations of FLCS in a more complex framework: the Genetic Algorithm module could operate considering also the matching degree of classifiers and the Credit Distribution module could use implementations of other RL algorithm as TD(λ) [22].

References

[1] A. Bonarini. ELF: Learning incomplete fuzzy rule sets for an autonomous robot. In Hans-Jürgen Zimmermann, editor, *First European Congress on Fuzzy and Intelligent Technologies – EUFIT'93*, volume 1, pages 69–75, Aachen, September 1993. Verlag der Augustinus Buchhandlung.

[2] A. Bonarini. Delayed reinforcement, fuzzy Q-learning and fuzzy logic controllers. In Physica-Verlag, editor, *Genetic Algorithms and Soft Computing,*

(Studies in Fuzziness, 8), pages 447–466, Berlin, Germany, 1996. F. Herrera, J.L. Verdegay.

[3] A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer Academic Press, Norwell, MA, 1996.

[4] A. Bonarini. Reinforcement distribution to fuzzy classifiers: a methodology to extend crisp algorithms. In *IEEE International Conference on Evolutionary Computation – WCCI-ICEC'98*, volume 1, pages 51–56, Piscataway, NJ, 1998. IEEE Computer Press.

[5] A. Bonarini. Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems. In *Proceedings of GECCO99*, Cambridge, MA, 1999. AAAI Press.

[6] A. Bonarini. Learning fuzzy classifier systems. In P. L. Lanzi, W Stolzmann, and S. W. Wilson, editors, *Learning Classifier System: new directions and concepts*. Springer-Verlag, Berlin, D, in press.

[7] A. Bonarini, C. Bonacina, and M. Matteucci. Fuzzy and crisp representation of real-valued input for learning classifier systems. In P. L. Lanzi, W Stolzmann, and S. W. Wilson, editors, *Learning Classifier System: new directions and concepts*. Springer-Verlag, Berlin, D, in press.

[8] L. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1–3):235–282, 1989.

[9] Brian Carse, Terence C. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *International Journal for Fuzzy Sets and Systems*, 80:273–293, 1996.

[10] O. Cordon, F. Herrera, and M. Lozano. A classified review on the combination fuzzy logic-genetic algorithms bibliography. Technical Report DECSAI-95129, Department of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain, October 1995.

[11] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, London, 1980.

[12] P. Y. Glorennec. Fuzzy Q-learning and evolutionary strategy for adaptive fuzzy control. In Hans Jürgen Zimmermann, editor, *Second European Congress on Intelligent Techniques and Soft Computing - EUFIT'94*, volume 1, pages 35–40, Promenade 9, D-52076 Aachen, September 20-23 1994. Verlag der Augustinus Buchhandlung.

[13] J. J. Grefenstette. Strategy acquisition with genetic algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, pages 186–201, New York, 1991. Van Nostrand Reinhold.

[14] F. Hoffmann and G. Pfister. Learning of a fuzzy control rule base using messy genetic algorithms. In Physica-Verlag, editor, *Genetic Algorithms and Soft Computing, (Studies in Fuzziness, 8)*, pages 279–305, Berlin, Germany, 1996. F. Herrera, J.L. Verdegay.

[15] J. H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, 2 edition, 1992.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[17] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic; Theory and Applications.* Prentice Hall, Upper Saddle River, N. Y., 1995.

[18] P. L. Lanzi and S. W. Wilson. Optimal classifier system performance in non-Markov environments. Technical Report 99.36, Dipartimento di Elettronica e Informazione - Politecnico di Milano, 1999. Also IlliGAL tech. report 99022, University of Illinois.

[19] A. Parodi and P. Bonelli. A new approach to fuzzy classifier systems. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 223–230, San Mateo, CA, USA, July 1993. Morgan Kaufmann.

[20] K. M. Passino and S. Yurkovich. *Fuzzy Control.* Addison Wesley, Menlo Park, CA, 1997.

[21] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[22] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction.* MIT Press, Cambridge, Massachusetts, 1999.

[23] Manuel Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In Lashon B. Belew, Richard K.; Booker, editor, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 346–353, San Diego, CA, July 1991. Morgan Kaufmann.

[24] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[25] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[26] S. W. Wilson. Generalization in the XCS classifier system. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 22-25July 1998. Morgan Kaufmann.

[27] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.