Fuzzy and Crisp Representations of Real-valued Input for Learning Classifier Systems

Andrea Bonarini, Claudio Bonacina, and Matteo Matteucci

Politecnico di Milano AI and Robotics Project,
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 , 20133 Milano, Italy
bonarini@elet.polimi.it,
WWW home page: http://www.elet.polimi.it/people/bonarini

Abstract. We discuss some issues concerning the application of learning classifier systems to real-valued applications. In particular, we focus on the possibility of classifying data by crisp and fuzzy intervals, showing the effect of their granularity on the learning performance. We introduce the concept of *sensorial cluster* and we discuss the difference between *cluster aliasing* and *perceptual aliasing*. We show the impact of different choices on the performance of both crisp and fuzzy learning classifier systems applied to a mobile, autonomous, robotic agent.

1 Introduction

There is an increasing interest in the Learning Classifier Systems (LCS) community about issues related to the application of LCS to real-valued environments. Most of the LCS applications proposed up to now work on gridworlds [14] or an interval-based representation of the input [7]. When interfacing to the real world by real-valued sensors, the selection of the interval granularity becomes a relevant issue. A fine-grained classification translates in a large search space, and a coarse-grained classification tends to induce *perceptual aliasing* [13]. We discuss some criteria to face this trade-off in section 2, where we also discuss the relationships between the interval-based representation partitioning the sensor data space, and the traditional gridworld, where the partition is done on spatial dimensions, strictly related to the *configuration space* (*c-space*) [9]. We remind the reader here that the c-space is the space of the variables needed to completely describe the relevant aspects of a system, in general, and of a robot, in our case. In section 3, we propose to adopt fuzzy intervals as a representation of the classifier input [2] [3]. Considering an interval-based model and a fuzzy one with the same number of intervals, 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 (discussed in Section 3) guarantees high robustness to noise and certain design errors [8]. However, the introduction of partially overlapping fuzzy intervals raises some learning problems that can be reduced by a careful redesign [4] of the classical reinforcement distribution algorithms. In section 4, we present the simplified learning classifier system that we have devised to test our proposal. In Section 5, we show the results we have obtained on a simulated mobile robot, where we consider most of the real world features, including stochastic and systematic noise, approximation, and uncertainty in sensors and actuators. The aim of this paper is to discuss the issues concerning learning crisp and fuzzy interval representations when operating with real-valued input/output; in particular we do not introduce new RL algorithms; they are presented and compared with other proposals elsewhere [6], [1], [2], [3], [4], nor we discuss about genetics, generalization, or the reinforcement function in our proposal. The considerations we present in this paper are general enough to be relevant for any RL algorithm operating on real values.

2 Grids, granularity, and real values

The information needed by a LCS is usually discretized, in order to have classifiers working on a finite set of input and output values. Most of the work done in the past in this field is based on gridworlds, where an *animat* [14] moves from one cell to another connected to the first, and perceives the presence of elements fully occupying cells. The kind of discretization we are proposing in this paper is different, since it is done on the information coming from sensors. We call the set of real values, one for each input variable, a *real-valued state*. We classify the real input values into classes corresponding to either crisp or fuzzy intervals. In figure 1 you may see such a classification for the distance measured by a sonar sensor. On the ordinates the degree of membership μ of the distance reported on the abscissas in each interval (below in the figure) or fuzzy set (above).

We call the set of crisp intervals, one for each input variable, matching a real-valued state, a *crisp state*. We call the set of fuzzy sets, one for each input variable, matching a real-valued state, a *fuzzy state*. In the following, we will refer to both these interpretations of a real-valued state with the common term *state*. The main point is that this discretization is done on the animat's perception, whereas the discretization in gridworlds is usually done on the environment where the animat operates, which corresponds, in general, to a, eventually partial, view of the c-space.

The other main difference concerns the type of movement the animat can do. In gridworlds, it can move from one cell to another (taking one of at most 8 possible actions from one position), in a real-valued world it can change its position in the c-space (with a real-valued move), but this may, or may not, bring the animat to a different state. The shift from a state to another is not linearly related to the movement. Depending on the sensors, their distribution, the classification model (and on the position in the environment, of course), the same movement may either bring the animat in a different state or not. As we show in section 5, the fact that an action brings it quite often to a different state heavily influences the learning performance. The relative frequency of state

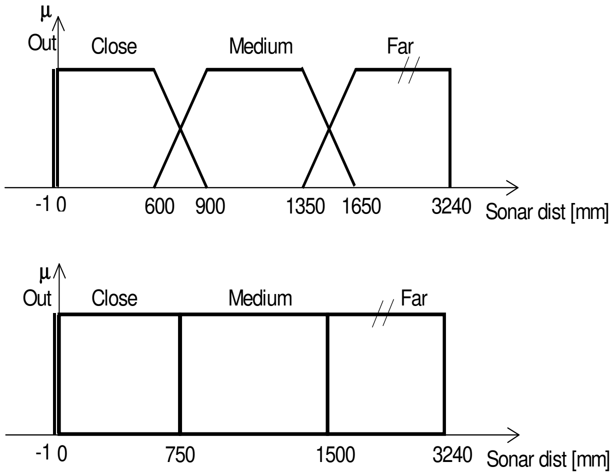

Fig. 1. The fuzzy membership functions (above) and the intervals (below) to interpret the distance measured by a sonar sensor.

change depends both on the granularity of the classification of the input space, and on the duration of the perceive-act (or *control*) step. This is the second type of discretization we need to do. In gridworlds, the duration of the control step is just one tick of an ideal clock; things may change at each tick, and, usually, the animat can move, in a tick, only from one cell to a connected one: its speed is one cell per tick. In the real world, we have real time, and real-valued output (in our case, speed and steering). The control step duration has a lower bound in the physical time needed to perceive (i.e., acquire and elaborate data from sensors), select the action, and send it to the actuators. The actual duration of the control step may be selected as a trade-off between the need for fast reaction, to select at each moment the best action, and the desire of changing state at each step; this improves the effectiveness of the learning process since reduces cluster aliasing (see section 3) and enhances the effect of the exploration policy. The results we present in section 5 show the impact of the length of the control step on the learning activity. This becomes even more important when the LCS receives so-called *continuous reinforcement*, that is when it is reinforced at each control step. If we call the time interval between two reinforcements an *evaluation step*, we may say that, with continuous reinforcement, the control step is identical to the evaluation step. A possibility to keep a fast reaction, but also a good chance of changing state between two subsequent evaluation steps, is to select an evaluation step larger than the control step. A first intuition about the importance of this problem was already present in [6] [2], where the term *episode* was used to name a sequence of control steps at the end of which a reinforcement is given. An even more radical approach is taken by many researchers, who evaluate the performance only when the system reaches a different interval (or fuzzy) state. The difference with respect to the episode approach is that, during an episode, the system may visit different states, while here the performance is evaluated every time the system changes state. In both cases, the action selected for a state is kept the same until the performance is evaluated, at the end of an episode, or when the system enters a new state.

Another important problem related to discretization concerns the *aliasing* phenomenon. Let us call the set of real values belonging to the same (crisp or fuzzy) interval state a *sensorial cluster*. All the real-valued states belonging to a sensorial cluster are classified in the same state, and correspond to the same subpopulation of classifiers. From each real-valued state it is possible to select one of the competing classifiers in the subpopulation matching the state, thus obtaining, in general, different reinforcement. Notice that the same classifier may be triggered by any of the real-valued states belonging to the sensorial cluster, and that, in general, the reinforcement function varies inside a sensorial cluster. Let us call the situation where it is possible to do an action that does not bring us out of the sensorial cluster *cluster aliasing*. This may cause problems to the learning algorithm since the same classifiers may have different reinforcements. The larger the cluster, the higher the possibility to obtain different reinforcement for the classifiers belonging to the corresponding subpopulation, and the lower is the accuracy of the LCS to learn the correct action to take from a sensorial cluster. This is another motivation to keep the cluster (and the corresponding state) as small as possible. Let us notice that cluster aliasing is slightly different from *perceptual aliasing* [13], defined as having the same internal representation for different states. Perceptual aliasing concerns the representation of the percepts, cluster aliasing concerns the fact that actions done in a state do not bring us out from it. In perceptual aliasing the problem is that there is the possibility to take the same reinforcement in different situations, in cluster aliasing is the opposite: taking different reinforcements for the same classifier causes problems to the learning activity.

3 Motivations for fuzzy intervals

Since their introduction in the late sixties [15], *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 a universe of discourse X (say, an interval of real numbers), and a fuzzy set, associated to the label *close*, defined by a membership function $\mu_{close}(.)$ that ranges on elements of the universe of discourse and maps them to the real interval $[0 \quad 1]$ (see figure 1). We can say that an element $\bar{x}$ of the universe of discourse (in our case a distance in millimeters) belongs to the fuzzy set *close* to the extent $\mu_{close}(\bar{x})$. 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 $\bar{x} = 690$ can be classified as *close* with a degree of 0.7.

Fuzzy sets are often used to classify real values in categories, thus making possible symbolic reasoning about the phenomena that underlie the incoming numbers. Membership of a number (e.g., $\bar{x}$) 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., $\mu_{close}(\bar{x})$) that comes from the definition of the fuzzy set. The membership value can be considered as an alternative representation of $\bar{x}$, although it is unique only for the class of monotonic membership functions. In general, the relationship between a real value $\bar{x}$ and its degree of membership to a fuzzy set $\bar{l}$, $\mu_{\bar{l}}(\bar{x})$, is not bijective. This potential problem is solved by defining partially overlapping fuzzy sets, and considering as representative of $\bar{x}$ the set of all the values $\mu_l(\bar{x})$, $\forall\ l$.

Fuzzy rules represent relationships among fuzzy sets. Since the mapping between a number and its classification is crisp and well-defined, we can say that a set of fuzzy rules is a compact representation of a relationship between the real values that can be classified by the fuzzy sets in their antecedents and consequents. A set of fuzzy rules implement a mapping among real values that combines the mappings implemented by each rule. So, a fuzzy LCS is a way to consider different local models (fuzzy rules) to obtain a complex, in general non linear, mapping [1] among real values. In particular, it is common to consider partially overlapping fuzzy sets, as the ones represented in figure 1. In this case, any set of real values for the input variables is matched to some extent by at least two rules. In particular, 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 [8]. To summarize: fuzzy sets can provide an alternative representation of real numbers, a fuzzy rule implements a model that maps real values to real values in given intervals, a set of fuzzy rules implement a complex mapping on the full range of the involved variables. These are the main reasons why fuzzy LCS are an interesting approach to learn relationships among real values.

With respect to a real-valued representation, fuzzy LCS introduce an abstraction level that reduces the search space and makes it possible to reason and learn on symbolic models. The accent on local models (shared with the interval representation) implies the possibility to learn by focusing at each step on small parts of the search space only. An interesting aspect of the fuzzy interpretation, when compared with the interval-based, is the possibility to have a smooth transition between close models: a real value in input is interpreted in a way similar to the close values in any region of the input range, while an even small difference in values across the crisp interval boundaries gives rise to radically different interpretations. The smooth transition among different models implemented by fuzzy rules implies robustness with respect to data noise [8].

This aspect has some impact also on the learning process. The interaction among local models, due to the intersection of neighbor fuzzy sets guarantees that local learning (in the *niche* corresponding to a fuzzy state) reflects on global performance [1], since the classifiers are judged for their performance not only inside the niche, but also on the boundaries shared with neighbor niches. In other terms, fuzzy classifiers are in competition with each other inside their niche, but have to cooperate with classifiers belonging to neighbor niches. It is not so for traditional crisp LCS, where competition is local, and cooperation on the output generation is not present, since the selected action is proposed by a single classifier.

Another interesting implication for learning is the increased granularity that the fuzzy representation induces. Let us consider figure 2, where we represent the interval (above) and the fuzzy interpretations (below) for data coming from a sonar sensor, mapped on the c-space of our robot. The c-space is defined by the two variables: d, the distance from the left wall of the corridor, and θ, the heading of the robot.

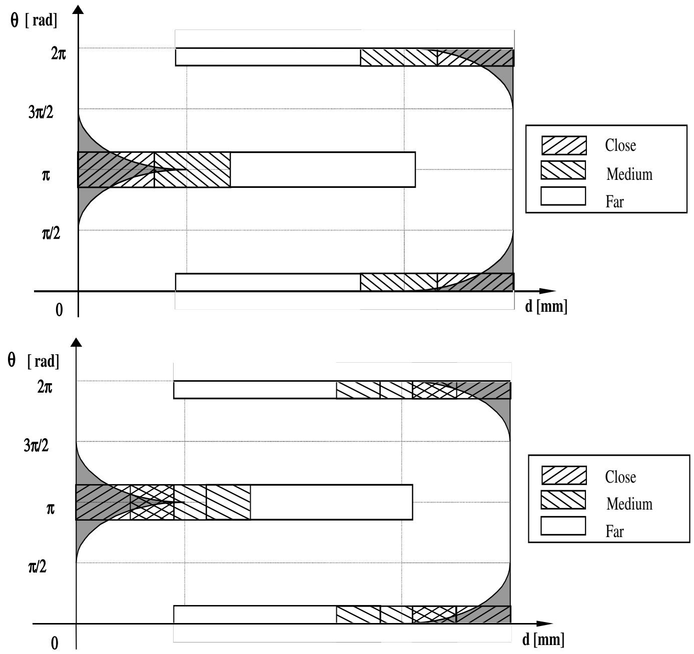

Fig. 2. The crisp (above) and fuzzy (below) interpretations of data from one sonar reported on the c-space.

Note that, since we consider the corridor as infinite, only lateral distances, such as d are relevant. From the dark gray areas of the c-space the robot can only bump on the wall, due to its limited steering ability (see Section 5.1). For each position of the robot in its c-space we have drawn the interpretation of data coming from the selected sonar (hatched rectangles in the figure - in general, the complete state is given by hypercubes). We see that each crisp interpretation is partitioned by the fuzzy one in many zones. This means that there are real-valued states that belong to one crisp state, but, at the same time, to a certain number of fuzzy states (two in the figure) with respective degrees of membership.

In other terms, although the number of classes is the same for both crisp and fuzzy classifications, this last partitions the input space in a larger number of distinct zones (five for each triple of classes in figure 2).

When the agent is in a real-valued state, it matches one crisp state, but more than one fuzzy state. Thus, the reinforcement obtained is distributed to a state (or an action done in a state) in the first case, and to a set of states (actions) in the second case. This means that the information obtained by the environment is distributed at each step to classifiers related to many states in the fuzzy approach. This increases the possibility to evaluate actions for real-valued states which can hardly been reached by the interval representation only. Let us consider, for instance, the *close* interval partially covering the "*no return*" zone of the c-space marked in dark gray in figure 2. It covers also a zone that the animat can leave, but this is very small, and the probability of visiting it is low, although the animat might learn interesting actions by going there. The same area is covered by fuzzy states that extend, with decreasing membership functions, also on the neighborhood. In the zone where membership functions overlap the animat movement comes from the composition of the action proposed by classifiers belonging to different subpopulations. In a sense, this reduces cluster aliasing, since actions done in the overlapping zone are in general different from actions proposed where there is no overlapping. Moreover, they are different with a kind of continuous shift from the action proposed in a state (e.g., *medium* in the figure) to that proposed in the other (e.g., *close*). Learning also takes into account the actions done in the overlapping areas, since reinforcement is also given because of actions done from that area. From figure 2, it is possible to imagine that classifiers operating from the *medium* state may interact positively with those (possibly highly inaccurate) operating from the *close* state in the overlapping area, possibly keeping the agent out from the *"no return"* zone.

Now, let us make some few remarks. Given the above mentioned phenomenon, it is easy to observe that fuzzy LCS converge to behaviors that keep the animat close to the overlapping zones. This means that the learning algorithms we have introduced tend to optimize the interactions among close states. This may be a suggestion for the design of membership functions. A second remark concerns the complexity of the states we have even in a simple application as the one we introduce in section 5. For instance, we have six sonars corresponding to six patterns analogous to those shown in figure 2, covering the c-space with many overlapping zones. We have found that, just by fuzzyfying the intervals as shown in figure 1, the number of the actual states reachable by our animat rises from 84 to 112, on the same c-space. This reduces cluster aliasing, per se, but also gives an idea of the number of the overlapping zones that may correspond to these fuzzy states. As a last remark, let us notice that, on the boundaries between fuzzy states, the contribution of the obtained reinforcement is mediated by the corresponding membership functions, thus propagating the smoothing effect, mentioned above with respect to the actions, also to the learning process.

4 A simple LCS framework

In this paper, we consider a simplified type of LCS [3]. Each classifier is a rule, whose antecedent is the conjunction of symbolic values for input variables. In *crisp LCS*, we consider that each of these symbols denotes an interval of real input values for the corresponding variable. In *fuzzy LCS*, each symbol denotes a fuzzy subset of the range of the real values the variable can take. A *don't care* symbol may replace any of the antecedent values, meaning that the value of the variable is not relevant for the classifier, that is that any real value matches the antecedent. The consequents are symbols corresponding to crisp values for each of the consequent variables. A classifier of this kind has a conceptual shape like this:

```
IF (FrontDistance IS Close)
   AND (LeftDistance IS Far)
   AND (RightDistance IS Don_T_Care)
THEN (TurnDirection IS Left)
```

actually implemented in a more compact string such as "130:3".

At each control step, one crisp classifier is selected among the ones whose antecedent is the crisp state matching the current real-valued state. The consequent values of the selected classifier are sent to the actuators of the robot.

When operating with fuzzy LCS, a real-valued state matches different fuzzy states, i.e., different fuzzy classifiers. We consider subpopulations of classifiers having the same antecedent (eventually including don't cares) and different consequents. At each control step, we select one classifier for each matching subpopulation (i.e., one for each fuzzy state). Each of the matching classifiers proposes an action with a weight proportional to its degree of matching. We combine the proposed actions with a fuzzy aggregation operator [8] (in the experiments here reported, *max*), we defuzzyfy the result [8] (here, by a *weighted sum*), and we send the obtained real value to the actuators. Notice that the output of crisp classifiers is selected in a small range of values; for instance, if we have only one output variable, the output is one of the k values defined for that variable. The output of fuzzy classifiers has a far larger range, since it comes from the weighted combination of the same number of output values (figure 3): the defuzzyfication process makes it possible to exploit the information about membership of the real-valued state to the fuzzy antecedent, thus producing a fine-grained output. Notice that this is not a real-valued output, only because the number of output values depends on the precision considered for the membership values.

We have implemented a tool [4] to study the impact on the learning process of different credit assignment algorithms, exploration strategies, and evolution algorithms. We have done many experiments exploiting the different choices offered by our tool, and the results of this extensive study will be presented in other papers. Here, we present results concerning the paper theme, and obtained by our versions of *Q-Learning* [12] and $TD(\lambda)$ [11], both for crisp and fuzzy LCS [4].

4.1 Reinforcement distribution for interval models

Q-Learning At time t the system is in the real-valued state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the interval-valued antecedent $\hat{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$ is updated by:

$$Q_t(\hat{s}_{t-1}, a_{t-1}) = Q_{t-1}(\hat{s}_{t-1}, a_{t-1}) + \alpha\,(r_t + \gamma\, max_a(Q_t(\hat{s}_t, a)) - Q_t(\hat{s}_{t-1}, a_{t-1}))$$

with the learning rate $0 < \alpha < 1$ and the discount factor $0 < \gamma \leq 1$. The strength of the classifier c_{t-1} selected by the exploration policy at time t is $Q(\hat{s}_{t-1}, a_{t-1})$.

TD(λ) At time t the reinforcement distribution algorithm performs these steps.

1. Receive the eventual reinforcement r_{t-1} due to the action a_{t-1} selected by the policy in state s_{t-1}.

2. Compute the estimation error of the evaluation function $V(\cdot)$ in s_{t-1}, equivalent to the immediate prediction error ΔE of the corresponding classifier c_{t-1} by:

$$\Delta E(c_{t-1}) = r_t + \gamma\, V_t(s_t) - V_{t-1}(s_{t-1})$$

with the discount factor $0 \leq \gamma \leq 1$.

3. For any state $\hat{s}$:

3.1. Update the *eligibility* $e(\hat{s})$ by the formula [10]:

$$e_t(\hat{s}) = \begin{cases} 1 & if\, \hat{s} = \hat{s}_{t-1} \\ \gamma\lambda e_{t-1}(\hat{s}) & otherwise \end{cases}$$

with $0 \leq \lambda \leq 1$.

3.2. If the eligibility of the state $\hat{s}$ is greater than a threshold value ϵ, with $0 \leq \epsilon \leq 1$, update the value function $V(\cdot)$ for any $\hat{s}$ according to:

$$V_t(\hat{s}) = V_{t-1}(\hat{s}) + \alpha \Delta E(c_{t-1}) e_t(\hat{s})$$

with $0 < \alpha < 1$.

4. Update the value of the policy for the pair $\langle \hat{s}_{t-1}, a_{t-1} \rangle$, that is the strength w of the classifier c_{t-1}, by the formula:

$$w_t(c_{t-1}) = w_{t-1}(c_{t-1}) + \beta\,(\Delta E(c_{t-1}))$$

with $\beta > 0$.

4.2 Reinforcement distribution for fuzzy models

We have extended the above presented algorithms to learn fuzzy models, following the considerations discussed here below.

A *real-valued state* is a set of real values for the input variables. We say that a real-valued state s matches a fuzzy state $\tilde{s}^i$ to some extent $\mu_{\tilde{s}^i}(s)$, when each real value s^k belonging to s belongs to the corresponding fuzzy set belonging to $\tilde{s}^i$, and $\mu_{\tilde{s}^i}(s)$ comes from the conjunction of the values $\mu_{\tilde{s}^i}(s^k)$, computed by the selected conjunction operator. Such operators belong to the class known as *T-norms* [8], among which the most common are *min* and *product*.

A crisp state usually matches more than one fuzzy state, while in standard LCS, it matches the set of intervals that define only one *interval state*. Let us consider subpopulations of classifiers for each fuzzy (or interval) state. In crisp LCS, classifiers belonging to the only subpopulation that matches the current state put actions in the action set, whereas in fuzzy LCS, we have many subpopulations matching the same crisp state, each proposing an action with some strength, proportional to the degree of matching of the classifier selected within the subpopulation. The strength values of the proposed actions are composed by an *aggregation* operator, usually implemented by a *T-conorm* (such as max), and then defuzzyfied by one of the many methods available [8]. Whatever *T-norm*, *T-conorm*, and defuzzyfication method is selected, the resulting action proposed by the set of selected fuzzy classifiers $\bar{a}$ is a function of the actions a^i proposed by the matching classifiers and of their degrees of matching $\mu_{\tilde{s}^i}$.

Since all the matching classifiers contribute to the performance of a Fuzzy LCS, the reinforcement should be distributed to all of them, proportionally to their contribution. Therefore, the first step to extend the reinforcement distribution algorithms consists of introducing a factor (let us call it ξ_{c^i}) that weights each classifier contribution. For each step, this factor is equal to the current contribution of the classifier (its degree of matching), weighted by the sum of the contribution given till now, saturated to a given value T (in these experiments $T = 2$). This enhances the learning rate of classifiers that do not yet participated much to the performance, yet.

$$\xi_{c^i} = \frac{\mu_{\tilde{s}^i}(s_t)}{min\left(T, \sum_{k=1,t} \mu_{\tilde{s}^i}(s_k)\right)}$$

The other important point concerns the evaluation of the best value in a given state, used in the $Q-Learning$ and $TD(\lambda)$ formulas. In this case, we cannot just take the value corresponding to the best value in the given state, since this state matches many fuzzy states, each contributing with different actions. We present our proposal for Q-Learning, leaving to the reader the analogous passages for $TD(\lambda)$. Let us consider $Q^*(c_t^{*i})$, the highest values of the classifiers c_t^{ji} belonging to each subpopulation corresponding to the matching fuzzy states i at time t.

$$Q^*(c_t^{*i}) = max_j(Q(c_t^{ji}))$$

Let us call $m_t(s)$ the sum of the $Q^*(c_t^{*i})$, each weighted by $\mu^*_{\tilde{s}^i}(s)$, the degree of matching of the corresponding classifier to the current, real-valued state s. This means that we consider as the reference best value in the current state, the combination of the best classifiers that can trigger in this state, one for each matching subpopulation. The corresponding formula is:

$$m_t(s) = \sum_{k=1,K} \mu^*_{\tilde{s}^k}(s)\, Q^*(c_t^{*k})$$

Q-Learning At time t the system is in the state s_t, after having executed action a_{t-1} from a state s_{t-1} matched by the fuzzy antecedent $\tilde{s}_{t-1}$, and receives the reinforcement r_t. The Q-value given to the classifiers c^i_{t-1} selected by the policy for each pair $\langle \tilde{s}^i_{t-1}, a^i_{t-1} \rangle$ is updated by the formula:

$$Q_t(c^i_{t-1}) = Q_{t-1}(c^i_{t-1}) + \alpha\, \xi_{c^i_{t-1}} \, \left(r_t + \gamma\, m_{t-1}(s_{t-1}) - Q_t(c^i_{t-1})\right)$$

TD(λ) The extension of the $TD(\lambda)$ algorithm presented for crisp LCS is straightforward, given the considerations done at the beginning of this section. We do not present again all the steps, but only the key formulas.

The eligibility trace of the fuzzy state $\tilde{s}$ is updated by the formula:

$$e_t(\tilde{s}^i) = \begin{cases} \mu_{\tilde{s}^i}(s) & if\ \tilde{s}^i_t = \tilde{s}^i_{t-1} \\ \gamma \lambda \mu_{\tilde{s}^i}(s) e_{t-1}(\tilde{s}^i) & otherwise \end{cases}$$

The strength of the classifier c_{t-1}, is updated by the formula:

$$w_t(c^i_{t-1}) = w_{t-1}(c^i_{t-1}) + \beta\, \xi_{c^i_{t-1}} (r_t + \gamma \sum_{\forall i} \mu_{\tilde{s}^i}(s_t) V(\tilde{s}^i) - V_{t-1}(\tilde{s}_{t-1}))$$

5 Experimental results

In this section, we first present the application, then the experimental settings, and, finally we discuss some of the results we have obtained.

5.1 The application

The application we have selected is navigation of a mobile robot in a corridor. The learning activity is done on a simulated model of the actual mobile robot CAT [1], which has a car-like kinematics, is 700 mm long, 600 mm wide, and can run at a maximum speed of 300 mm/sec both forward and backward. The maximum steering degree is 30° on each side, and this is an important limitation to the maneuvering capability. In the configuration we adopted for these experiments, CAT had 8 bumpers (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 (*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 3,230 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 pair of two Polaroid sensors available on the market, each having a detection cone of about 30°.

Data from sonars are interpreted either in terms of fuzzy sets, or in terms of crisp intervals, as 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. A 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, as in figure 3. In the experiments we present here, the robot goes at a constant speed of 150 mm/sec. Whenever the robot get stuck on the wall, we bring it three steps back, so that there is a probability that it comes in a different sensorial cluster and may try different actions.

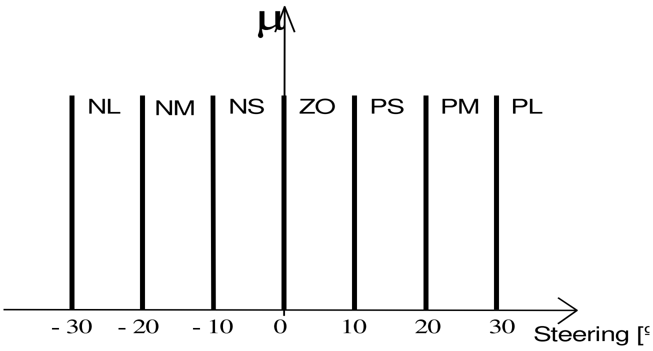

Fig. 3. The output values of steering.

5.2 Experimental settings

Since the aim of these experiments is to discuss the impact of cluster aliasing on the learning performance, we keep fixed all the learning parameters and procedures not directly related to this aspect. The exploration strategy is: take the best matching classifier for each subpopulation with a probability equal to $(N-1)/N$, and a classifier randomly selected with a probability equal to $1/N$, where N is a design parameter to tune exploration. In the experiments here presented, we excluded generalization, and, with it, the need for crossover: the classifiers are generated only by cover detection. The number of classifiers in a subpopulation matching a state is inversely proportional to the top performance of the subpopulation: the worst classifiers matching the current state are deleted when the performance of the subpopulation increases. Mutation is limited to the consequent part. The aim of the learning system is to find the best classifier for each state.

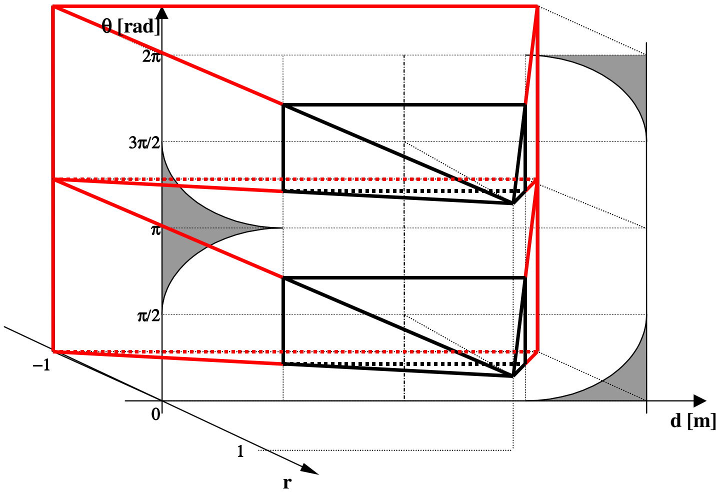

Fig. 4. The reinforcement function for the experiments reported in this paper, drawn on the c-space, along a third dimension coming towards the reader.

The reinforcement function r is the same in all the experiments, and it is designed to evolve a controller that keeps the agent moving as close as possible to the center of a corridor. In figure 4 we represent it on the c-space. You can see that it is maximum when the animat is to the center of the corridor, parallel to the walls, in any of the two directions, and minimum when the animat is bumping on the walls. Notice that it does not take into account the "no return" zones (in gray in the c-space). In this paper, we are not concerned with the optimization of the reinforcement function, and we only show the results obtained with this reinforcement function, that we consider to be *continuous*, since it can produce a reinforcement for any point in the c-space, and *complete*, since it completely describes the task, giving a reinforcement for any of the values of the variables in the c-space. A more detailed discussion about the design of the reinforcement function is presented in a forthcoming paper [5].

Each learning trial lasts 500 control steps, and sets of trials to be compared with each other have the same randomization seed. Each experiment consists of trials sequentially starting from the 11 different positions shown in figure 5, in the order mentioned there; each sequence is repeated 10 times for a total of 110 trials and 55,000 control steps. From the set of classifiers we have at the end of the learning process, we consider only the best classifier for each subpopulation that has been tested enough, and we evaluate the performance of the controller they implement in 27 trials, lasting 700 control steps each, starting from the 11 positions shown in figure 5, and from 16 intermediate ones, in order to test the generality of the learned controller.

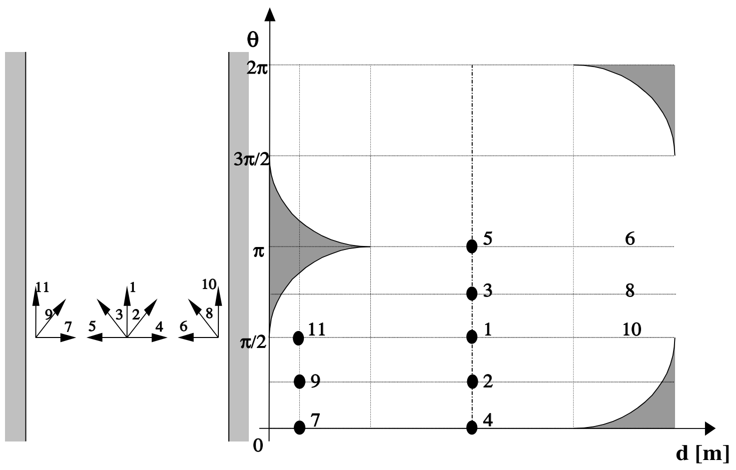

Fig. 5. The starting position for the learning trials, in the environment (left) and on the c-space (right).

5.3 Results

In this section, we discuss results from experiments each done as mentioned in section 5.2, since we believe these are more interesting than results averaged over a set of similar experiments. Moreover, we have run experiments for more than 8 millions control steps and the difference among analogous ones is non significant. In figure 6 we show the istantaneous performance (computed as the instantaneous reinforcement R), over the above mentioned 27 trials, of a learned controller whose control step lasts 600 ms. As you can see, the behavior is almost satisfactory for both the crisp and the fuzzy LCS, although the latter shows a better behavior, since its performance averaged over the 27 trials (covering all the reasonable starting positions) is higher, and it does not bring the animat on the walls, as it happens in one case (position 8) with the crisp LCS. This confirms what we have written in section 3: the fuzzy LCS is more robust with respect to cluster aliasing.

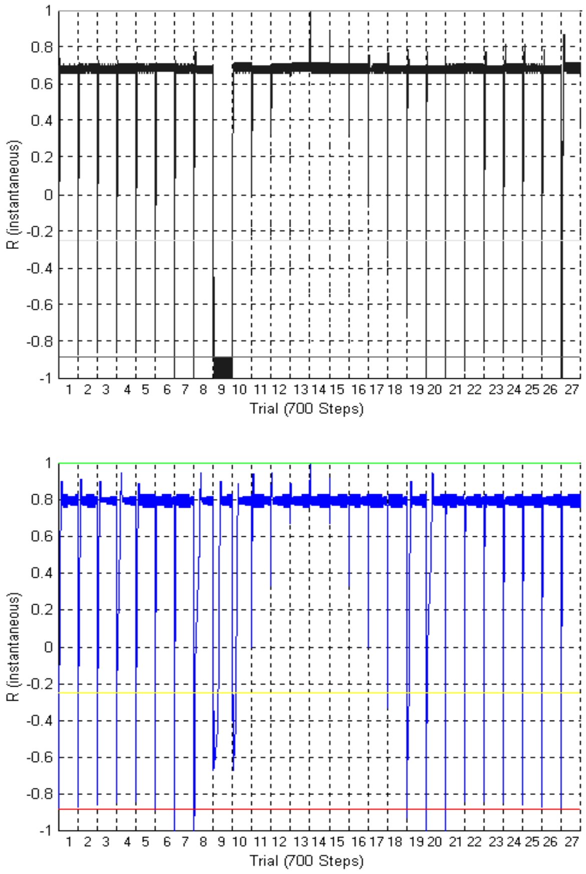

Fig. 6. Performance of a controller learnt by crisp (above) and fuzzy (below) Q-learning, control step 600ms.

In figure 7 we show the results of the same experiment, but with a control step of 100 ms. This is a way to increase cluster aliasing, since we select, and evaluate, actions at a higher rate: the probability that, at the next step, the animat is still in the same state is higher than in the previous case. In this situation, the next classifier will be selected from the same subpopulation. As we can see, the performance of both LCS decreases, although that of the crisp LCS is still slightly worse than that of the fuzzy LCS.

The same qualitative comments apply to the results obtained by adopting $TD(\lambda)$ instead of Q-learning. In this application $TD(\lambda)$ has a slightly worst performance with respect to Q-learning. This is due to our continuous reinforcement function, that slows down learning with $TD(\lambda)$, since this algorithm suffers more than Q-learning from the cluster aliasing problem. A continuous reinforcement

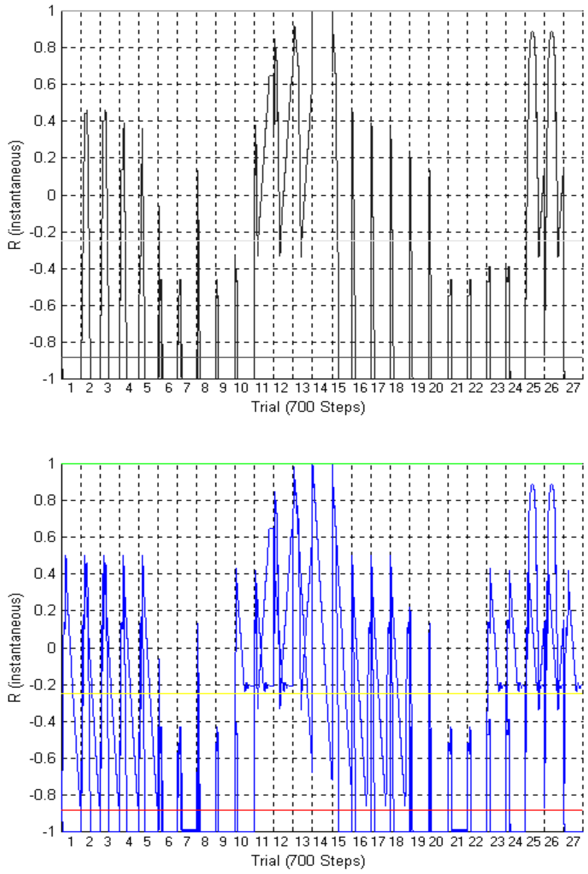

Fig. 7. Performance of a controller learnt by crisp (above) and fuzzy (below) Q-learning, control step 100ms.

function increases the confusion in the evaluation of the performance inside a cluster, since the reinforcement is similar for all the actions taken from close real-valued states, and quite different from actions taken from far real-valued states inside the same cluster.

In figure 8, we show the plots of the average reinforcement obtained during learning by Q-learning with control steps of 600ms and 100ms with the crisp and fuzzy models. In both cases, we may notice that learning reaches a stable point, but we see that they are different, due to the higher cluster aliasing present in the second situation.

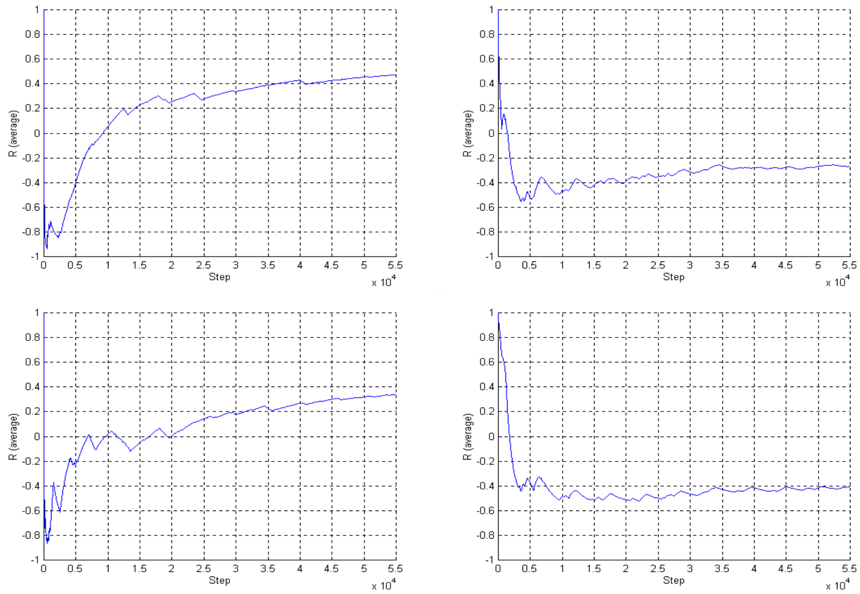

Fig. 8. Average reinforcement during learning: crisp (above) and fuzzy (below) Q-learning, control step 600ms (left), 100ms (right).

6 Conclusions

We have presented some of the problems arising when applying learning classifier systems to real-valued environments. We have focused on the impact of an interval-based representation on learning, discussing the problem of cluster aliasing that arises when more than one action is possible from one state and action execution does not imply state change.

We have introduced a fuzzy representation to get closer to a real-valued representation, while maintaining a small, symbolic, search space. Finally, we have discussed the results we have obtained to learn a relatively simple robotic task: fuzzy LCS seem more robust than interval-based LCS with respect to the problem of cluster aliasing, and, at least in the cases presented in this paper, it seems also more effective.

In future papers we will present the impact of other factors on the learning performance, such as the exploration strategy, generalization, discrete, incomplete, or accuracy-based reinforcement functions. The only consideration we would like to anticipate here is that fuzzy LCS require, in general, a higher computational effort, and a higher level of exploration; moreover they work better with crisp information, such as discrete reinforcement functions.

Acknowledgements This project is partially supported by the Politecnico di Milano Research Grant "Development of autonomous agents through machine learning", and partially by the project "CERTAMEN" co-funded by the Italian Ministry of University and Scientific and Technological Research. We are indebted with Nicolino De Marco, who implemented the first version of the tool supporting this research.

References

1. A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy modeling: paradigms and practice*, pages 265–284, Norwell, MA, 1996. Kluwer Academic Press.
2. A. Bonarini. Anytime learning and adaptation of hierarchical fuzzy logic behaviors. *Adaptive Behavior Journal*, 5(3–4):281–315, 1997.
3. 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.
4. A. Bonarini. Comparing reinforcement learning algorithms applied to crisp and fuzzy learning classifier systems. In *IWLCS99*, Cambridge, MA, 1999. AAAI Press.
5. A. Bonarini, C. Bonacina, and M. Matteucci. A framework to support the reinforcement function design. In preparation, 2000.
6. Andrea 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, D, 1993. Verlag der Augustinus Buchhandlung.
7. M. Dorigo and M. Colombetti. *Robot shaping: an experiment in behavior engineering*. MIT Press / Bradford Books, 1997.
8. G. J. Klir, B. Yuan, and U. St. Clair. *Fuzzy set theory: foundations and applicatons.* Prentice-Hall, Englewood Cliffs, MA, 1997.
9. Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transaction on Computers*, C-32(2):26–38, feb 1983.
10. S. P. Singh and R. S. Sutton. reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1):123–158, 1996.
11. R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
12. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.
13. S. D. Whitehead and D. H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.
14. S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
15. L. A. Zadeh. Fuzzy sets. *Information and control*, 8:338–353, 1966.