A Multiagent Architecture for Fuzzy Modeling*

M. Delgado
Departamento Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada, Spain

A. F. Gómez-Skarmeta,† J. Gómez Marín-Blázquez, and H. Martínez Barberá
Departamento Informática y Sistemas, Universidad de Murcia, Apartado Correos 4021, 30001 Murcia, Spain

In this paper a hybrid learning system that combines different fuzzy modeling techniques is being investigated. In order to implement the different methods, we propose the use of intelligent agents, which collaborate by means of a multiagent architecture. This approach, involving agents which embody the different problem solving methods, is a potentially useful strategy for enhancing the power of fuzzy modeling systems. © 1999 John Wiley & Sons, Inc.

I. INTRODUCTION

In recent years, fuzzy modeling, as a complement to the conventional modeling techniques, has become an active research topic and found successful applications in many areas. However, most fuzzy models are presently built based only on operator's experience and knowledge, but when a process is complex there may not be an expert available.[1] In this kind of situation the use of unsupervised learning techniques is of fundamental importance. The problem can be stated as follows. Given a set of data for which we presume some functional dependency, the question arises whether there is a suitable methodology to derive (fuzzy) rules from the data that characterize the unknown function as precisely as possible. Recently, several approaches have been proposed for automatically generating fuzzy if–then rules from numerical data without domain experts.[2]

As we attempt to solve real-world problems, we realize that they are typically ill-defined systems, difficult to model and with large-scale solution spaces. In these cases precise models are impractical, too expensive, or nonexis-

*This work has been partially supported by CICYT project TIC97-1343-C002-02, and Gómez Marín-Blázquez's grant from the Séneca Foundation for applied research.

†Author to whom correspondence should be addressed. e-mail: skarmeta@dif.um.es.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 14, 305–329 (1999)
 CCC 0884-8173/99/030305-25

tent. The relevant available information is usually in the form of empirical prior knowledge and input–output data representing instances of the system's behavior. Therefore, we need approximate reasoning systems capable of handling such imperfect information. The term soft computing describes the combination of different emerging computing disciplines, within which we have fuzzy logic, probabilistic reasoning, neural networks, and genetic algorithms. In many situations the use of only one of these multiple possible approximate techniques is not practical. We need the collaboration between them and this is the approach taken by hybrid systems. Over the past few years we have seen an increasing number of hybrid algorithms, in which two or more soft computing technologies have been integrated to improve the overall algorithm performance.[3]

Those hybrid systems are composed of several different and interchangeable techniques. The best combination of such techniques may vary from problem to problem. This leads us to the idea of a multiple loosely coupled coprocessing distributed system. Each technique can be developed as a system component and encapsulated to offer a homogeneous interface in a distributed environment. Mechanisms apt for intelligent cooperation among these components (agents) should be supported to allow the design and implementation of cooperative distributed applications.

In this paper we propose the use of intelligent agents which collaborate by means of a multiagent architecture, as a framework to investigate different learning techniques in a fuzzy modeling context. This approach involving agents which embody the different problem solving methods, is a potentially useful strategy for enhancing the power of fuzzy modeling systems. Our objective is to show how this framework let us experiment with different techniques in order to select the combination that better approximated the system behavior.

In Section II we describe the fuzzy modeling process from the perspective of intelligent agents and multiagent architectures, and a general perspective of the architecture we propose. In Section III we present the techniques we have implemented within our agents in the contexts of the fuzzy modeling process. Next in Section IV we show the behavior of different combinations of agent's techniques in a simple but representative problem, where the focus of the test is to show the possibilities of the architecture in order to allow different interrelations between the learning techniques. Finally we present some conclusions and indications of future trends.

II. FUZZY MODELING AND AGENTS

In recent times, several different techniques, which may be subsumed in the frame of soft computing, have appeared in the literature related with fuzzy modeling. Thus we find proposals that use fuzzy neural networks,[4,5] fuzzy subset theory combined with descent gradient techniques, or with clustering techniques,[7,6,8] etc.... Recently combinations, of those techniques have been used to try to solve different problems related with systems modeling.[1,9,10]

Fuzzy modeling is an approach used to form a fuzzy systems model. In fuzzy modeling, the most important problem is the identification method of a system.

The identification of a fuzzy model using input–output data such as the ones we are concerned with consists of two aspects: structure identification and parameter identification.[11] Fuzzy modeling is based on the idea of finding a set of local input–output relations describing a process. So it is expected that the method of fuzzy modeling can express a nonlinear process better than an ordinary method. A fuzzy model consists of a number of fuzzy if–then rules. There are two parts in a fuzzy rule: the premise and the consequent part. These parts are formed by the combination of fuzzy sets defined in the different domains of the variables. Each input–output relation is described by a fuzzy rule. The fuzzy rules are formed by partitioning, in a fuzzy way, the input spaces and associating to each of them a consequent expression that could be a fuzzy set, a linear relation, or a singleton value. Therefore, the premise of a fuzzy rule indicates a fuzzy subspace of the input variables to which a relation with the output variables can be established.[12]

The structure identification consists of the premise structure identification and the consequent structure identification. When we consider a multiple inputs, single output (MISO) system, the structure identification of a system has to find the input variables which affect the output from a collection of possible variables. Once the variables are identified the structure identification is concerned with the input–output relations identification. This is one of the most crucial problems in the fuzzy modeling, and it is concerned with the identification of an optimal number of fuzzy partitions of the input space, where the number of fuzzy subspace corresponds to that of the number of fuzzy rules. The parameter identification is concerned with the identification of the parameters of the membership functions of the fuzzy sets that are used in the fuzzy rules. In many cases some form of parameter identification is performed within the structure identification, and then a tuning parameter identification is separately performed after the structure identification.[11]

In the context of fuzzy modeling, the use of clustering based techniques has had a great success. The objective of this clustering is to detect the behaviors present in the data obtained in a system under observation with no other additional information (to perform a process of fuzzy modeling through a set of fuzzy rules). Some different approximate and descriptive methods have been proposed in Refs. 13–15. With the results described in those works, what we get is a rule set that is only a first approach to the problem of fuzzy modeling of the system we are studying. The next objective is to approach the problem of how to optimize the rules generated using some of the proposed methods. To do so we propose the use of tuning techniques based on the modification of some of the parameters of the fuzzy rules obtained with the clustering techniques and give sense to the creation of hybrid systems applied to fuzzy system modeling.

It is clear that to obtain a good fuzzy model, it is not possible to apply the same kind of techniques to the different problems outlined in the fuzzy modeling process. The need of a combination of alternative and complementary techniques during the structure and parameter identification has shown the possibilities of the use of hybrid systems in fuzzy modeling, and particularly the use of intelligent agents as the component of these hybrid systems.

A. Intelligent Agents and Agents Architecture

In recent years there has been a growing interest in AI research toward the distributed artificial intelligence, due to the growth of computer networks. In distributed artificial intelligence the aim is to create multiple intelligent entities, which may interact and communicate possibly by sharing a common knowledge source, in order to provide additional problem solving facilities. Such entities are known as intelligent agents.[16]

There is not a viable definition of what an agent is. Neither the research community nor the software developers have a reasonable working definition of what it takes for some particular program or artifact to be qualified as an "agent." An intelligent agent may be considered to be any program or device that has the capability of reasoning and decision making. Important properties of an agent are autonomy, in the sense of the ability to operate without human intervention, and rationality, in the sense of being able to maximize its performance with respect to some objective. In any way an agent to be considered intelligent must exhibit some form of intelligence behavior from the user point of view.

By providing relatively small intelligent components that may be more easily integrated into more general software products, we can try to solve problems that could not be solved using a large and complex software system. The provision of facilities for the inclusion of multiple fuzzy modeling problem solving paradigms in a single software system has several advantages: different strategies may be applied to solve different aspects of a problem, with the most appropriate approach being selected for each aspect; different problem solving functions may proceed in parallel, to speed up problem solution times; two or more problem solving methods may concurrently be active in attempting to solve the same problem, and then the best solution chosen. This last feature is particularly relevant in the case of, for example, safety-critical systems, since if one method fails, a solution may still be found; also solutions provided by different methods may be compared, to provide an additional degree of safety assurance.[16]

Agents of this kind can be inserted into a multiagent system. Several agents can be created with different initial parameters. The agents can compete between each other to determine the strongest or the most dominant. A multiagent system is composed of largely autonomous and decentralized components or agents, cooperating together in performing complex solving problem tasks. The use of multiagent systems creates a framework, which allows the interoperation of a vast set of heterogeneous solutions to carry out the complex desired tasks. That is, offering black box interfaces, and thus high degrees of encapsulation and modularity, and of course, supporting several interaction schemes.

B. Multiagent Platform

It is necessary to pay attention to the components that compose the backbone of the development of an intelligent system. Those pillars are the

learning algorithms that are able to deal with the uncertainty and vagueness, the technology that allows interaction among the agents that are going to compose the system, the communication languages, and the multiagent architectures.

Hence, we found two topics of interest:

(a) The interaction among agents.
(b) The integration of the intelligence in software agents.

In an agent based system the communication among agents is of paramount importance. At the moment, several international research groups are proposing several standards for agent communication. For example, with regard to the communication among agents topic there are proposals such as KQML from the ARPA KSE[17,18] and the open agent architecture proposed by SRI. After the evaluation of the proposals we opted to use KQML for this task. The KQML language[19,20] proposed by the ARPA knowledge sharing effort (KSE)‡ is becoming the standard language to exchange information among agents and knowledge bases. Among its more relevant features it has a high level communication language to exchange syntax and ontology-independent information. It allows encapsulating information and its set of performatives (instructions) is extensible to allow a later extension.

To integrate intelligence there is no recent proposal broadly accepted so we propose the creation of a multiagent system for learning in a fuzzy modeling environment, focusing all this technology with an objective: to develop a distributed software agent group that can collaborate asynchronously among themselves in the most autonomous and efficient way possible.

1. Distributed Multiagent Architecture

Multiagent architectures investigate knowledge representation models like communication and inference techniques to let an independent group of agents solve problems together. Those agents form a multiagent system that can be defined as "a loosely coupled network of problem solvers that work together to solve problems that are beyond their individual capabilities."[21] Agents are autonomous entities capable of carrying out specific tasks by themselves or through cooperation with other agents. Multiagent systems offer a decentralized model of control and use the mechanisms of message passing for communication purposes.

We will use the approach of cooperative multiagent system (CMAS) where the agents act to globally improve the system versus the self-interested multiagent systems where each agent acts looking for a self benefit and the system

‡The ARPA knowledge sharing effort is a consortium to develop conventions that ease the reutilization of knowledge bases and knowledge based systems.

benefit is derived of the individual benefit of its components. Our agents are designed to cooperate and, in the case of a conflict among them, ask a decisor agent that evaluates the proposals and choose.

One of the advantages of agent utilization is that those programs can be specialized to very specific tasks and, later, communicate to solve more difficult problems in their entirety (similar to the combination of several specialized parts of our brain to generate an answer to a problem). In this way, moreover, we can add to the systems new features as new specialized agents without any change in the system. This is the reason for (as we comment earlier) the division of the different parts of the system in independent blocks, each one of those specialized in a specific task.

The multiagent systems (MAS) should deal with three basic problems: communication, coordination, and negotiation.

The agents not only have to be able to communicate among them to transmit the information they process, but also have to be able to communicate their existence to the others, offer them its services or ask for services to other agents that exist in the system. So we need a collaboration architecture among agents. A common representation language (KQML) is used by agents to exchange messages.

The coordination is of paramount importance in a MAS. The agents have to be coordinated to be able to combine their individual capabilities. In our approximation we have agents very specialized to solve specific problems and other agents that are in charge of dividing the problems into tasks that are performed by those specialized agents. Most of the tasks have dependencies among them (some have to be performed before others), a fact that forces us to have a coordination mechanism that indicates to each agent what to do and when to do it (and, as we mention later, what to do if it is unable to do it).

The negotiation arises when there exist goal conflicts among agents. As we mentioned above we solve the problem through decisor agents that evaluate some measure about the proposals of the agents and select one (or a combination) of them.

From an external perspective, agents are structured as a set of elements; services (functionality offered to other agents), goals (self-imposed tasks), resources (external sources of information), internal objects (data structures shared by all the processes launched by an agent), and control (specification of how service requests are handled).

At the architecture level, coordination is achieved in the platform through specialized agents (facilitators, planners, decisors...). At birth, agents register to a particular facilitator, informing about their net address, the services they offer, the services they would need, etc.

Now we have defined the features we want for our agents, have solved the problem of communication among them, and have the learning techniques (we will explain them later) and algorithms to be applied in our systems, we propose a kind of architecture for the system. The architecture we propose is a multilayer architecture. One that has at least two layers.

(1) An internal layer of task agents that will take the necessary steps to let the information requests made by the interface agents be performed. This layer will also take charge of the learning part of the system, processing the information to improve the results as much as possible.
(2) A layer of interface agents: These agents are in charge of dealing with human operators. They adapt to them, request additional information or confirmation if it is necessary, and present to them the information they request in a useful way, etc.

As we mentioned above, we also have several service agents distributed over the system. An example of a service agent is the facilitator agent. The facilitator acts as a Yellow pages to other agents, indicating where to find agents that perform specific tasks. If a task agent needs any service and it does not know where to find it, it requests from the facilitator agent a list of agents that can help it. The facilitator returns that information (if it does not have it, it will look for it) and later, it is the task agent who is in charge of the activation of those agents, and sending them the jobs, retrieving the results later.

2. Interface Layer

This layer is in charge of dealing with the world. It is composed of the software in charge of getting the users requests and eliminating inconsistencies and ambiguities. This layer is also in charge of showing the results and asking the user about the quality of the service offered (a basic feedback system). The interface layer gives or retrieves all the information the system produces or needs. It deals with the human users and presents the information to them in a comprehensive way. It also deals with external databases or information resources. It is composed of what we generally call an interface or information agents.

3. Task Layer

This layer is responsible for planning and decomposition of the information request into subtasks, as well as the activation of the agents in charge of information retrieval. (See Fig. 1.) The learning software also composes the layer. It consists of two parts. The agents that are in charge of knowledge extraction from the information that the system has§ compose one of these parts. To do so some agents exist that are in charge of the detection of groups of data with similar behavior. With such data another agent tries to create a collection of fuzzy rules applying several learning techniques (fuzzy clustering, gradient descent, genetic algorithms). The system will evolve depending on the acceptance or not from the feedback system. This feedback can be from a human user–expert or training data files. Due to the use of several learning techniques, the system will tend to use those techniques that produce better

§It is supposed that the system acquires the information in some way. In particular, we have a complete layer in charge of information acquisition and preprocessing.

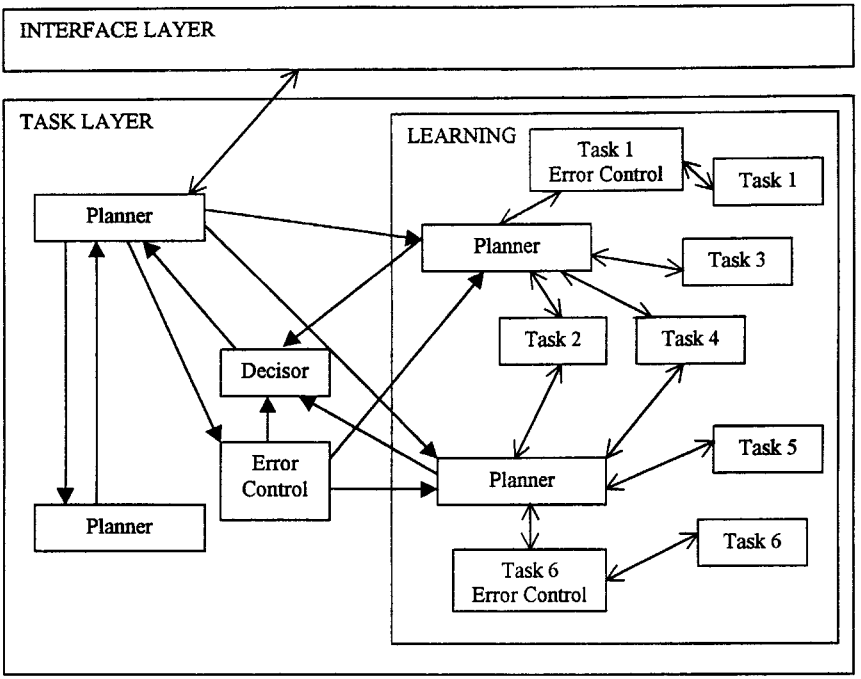

Figure 1. A schema of the Task layer.

results for each particular task. Our fuzzy modeling schema is based on:

- Variable identification
- Clustering generation on such features
- Creation of fuzzy rule systems
- Tuning of such rules
- Use of the rule systems for inference

For each of those steps there are several techniques. Each technique has been implemented in a specialized agent. Those agents usually have only the basic KQML communication interface to allow them to be activated, receive the parameters and, in the case where the techniques allow it, give intermediate results. This is useful if we have time restrictions. If we need a result immediately from a rule system (even if it is not optimally tuned) our tuning agent allows us to give an intermediate result and continue tuning the rules while the system gives an answer, not breaking the time limit imposed.

So we have a parallel application with different ways of processing information. At the end, a decisor agent is in charge of choosing, among several options, the one it considers the most interesting. It is also in charge of informing the planners of the positive or negative result of the choice to allow, in the future, choosing some techniques over others or to experiment with changes in the

many parameters that each technique has. For example, in the case of the rule tuning it can modify the learning rates, the number of iterations, etc.

4. *Agents of Our Proposal*

In this section we describe the agents that our system has. We classify them by the role they have in the system, that is; services (functionality offered to other agents), goals (self-imposed tasks), resources (external sources of information), internal objects (data structures shared by all the processes launched by an agent), and control (specification of how service requests are handled).

Service Agents

Facilitator. This agent acts as a Yellow pages to others agents. Every new agent in the system should register its existence to the facilitator and also the services that the agent offers to the system. The facilitator will provide to any agent information about the services offered by the system, and how to obtain them.

Task Agents (*Goal Agents*)

We enumerate the different task agents our system has. In the next section there is a complete explanation of its operation.

Clustering Agents. This kind of agent performs a clustering in a data set. It will try to find the behavior tendencies of the data, grouping data into several clusters. Each cluster will be used to generate one or more rules that model the data.

Rule Generation Agents. This agent creates a fuzzy rule system. There are several kinds of rule generation agents. They differ in the way they create them and also the kind of information they take as input. The parameters they usually have are the number of rules to be generated and the type of fuzzy number used in the rule.

Tuning Agents. These agents tune the fuzzy rules systems and try to optimize them using training data. This training data is provided by the system in some way. There is a different agent for each different tuning method. This way makes it easier to add to the system new features and methods simply by registering them in the facilitator.

Evaluator Agent. The evaluator is an agent that given a particular fuzzy rule system infers results on new data. Due the different kinds of fuzzy rule systems there is also several evaluators.

Resource Agents

Information Agents. They provide information to the system. There are as many of these agents as there are different information resources available.

Internal Objects

Rule Systems Database. In this database we store all the fuzzy rule systems we produce or have.

Information Database. This database mainly stores the training data the system has.

Control Agents

Planner Agent. The planning agents are in charge of the activation and the synchronization of the different agents. Those agents elaborate a work plan and are in charge of ensuring that such a work plan is fulfilled. They receive the assignments from the interface layer. They activate the information agents, learning agents and, if something goes wrong, the error control agents.

Decisor Agent. When a planner agent decides to perform a task that can be fulfilled in several ways it usually launches several agents to perform the same task. When those agents finish the assigned task they return the results to a special agent called the decisor. The decisor chooses one of the different results, the one it considers to fit best. It can also perform a fusion of the different results into a final one.

Error Control Agent. The error control agents are in charge of controlling the possible malfunctions of the system. There is usually an error control associated to each task agent that can fail. All the messages to the task agents are then redirected to the error agent that filters them and also processes the error control directives (as time limits, sequentiality, etc.) that the messages may contain. The error control then takes charge of reporting the success or failure of the task. They normally control these situations.

(i) *Time limit restrictions*. Many tasks have a time limit imposed on them. In the tasks susceptible of being interrupted they request of the task agent an answer when the limits are reached. In a task that can not be interrupted it informs of the failure to the planners.
(ii) *Unreachable conditions*. Some tasks require that some conditions have to be fulfilled to work properly. Sometimes those conditions can be unreachable by the system. The error agent has to inform the planners that the task it controls can not start or finish due to the condition that it is likely the system cannot reach.
(iii) *Sequential tasks*. Some tasks have to be performed in a given order. The planners have to ensure that the tasks are performed in the correct order. Sometimes, the planners allow the error control agent to perform this action.
(iv) *Information not available*. Before launching the agent it controls, the error control agent has to verify that all the information the agent needs is available in the system. It is the responsibility of the planners that the parameters are ready, but the error agent will check the additional information the agent may request.

Besides the error control assigned to each fallible agent there is a global system error control agent in charge of avoiding the degradation of the condition of the system. It kills nonproductive agents, set limits on the workspace of agents, set priorities, etc.

III. FUZZY MODELING IN OUR MULTIAGENT SYSTEMS ARCHITECTURE

Once we have defined the multiagent architecture we return our attention to the learning techniques our agents implement.

We will consider, without losing generality, a MISO system (multiple input, single output). We want to find a system that approaches the function φ: $X^p \to Y$ that models the system. We have a sample set of the behavior of the system in the space $(X^p \times Y)$, $\Omega = ((x_{t1}, x_{t2}, K, x_{tp}), y_t)$, $t = 1, 2, K, n$, where X_1, X_2, K, X_p are the domains of the inputs and Y is the domain of the output.

Supposing we have a collection of data that represent the behavior tendencies of the system, we want to obtain a characterization of such behavior using k fuzzy rules that have this form,

$$R_h\text{: } \textbf{If } x \textbf{ is } A_h \textbf{ then } y \textbf{ is } B_h \qquad h = 1, \ldots, k \tag{1}$$

where A_h and B_h are, respectively, the fuzzy sets in X^p and Y. Once all of those rules have been created (in the next section we will show how we will create them) an inference mechanism can be used for any new input, using approximate reasoning.

A. Background Rules

When we try to obtain an approximation of a function φ: $X^p \to Y$ by means of a fuzzy rule system using a collection of input–output sample data, we find an inherent problem of this kind of modeling. With a limited number of rules (and even in systems with many rules) it is very difficult or even artificial to completely cover the input space. When we create the rule systems we may force it to cover all the input space making the rules sufficiently broad but this is not a good strategy. The first thing that is tried when we create a rule system is to find the function zones where the samples have a similar behavior, to cover such zone with a rule. To find such zones a clustering technique is used. The rules generated by the clustering will try to better cover the points that have a greater membership value to such cluster. It is not very probable that the overlapping of all such generated rules will completely cover the input space. Besides, even if we force the system to cover all the input space, when we try to tune the system by any tuning technique (to try to model in a more precise way the underlying function) the modification of the limits of the rules quickly leads us to a situation in which there are some holes in the input space, what is called a sparse rule system.

To solve this problem we use a background rule system (BRS) associated to a fuzzy rules system. Our background rules behave in a similar way to a normal fuzzy rule. The mechanism is a little bit different. When a sample does not activate any rule of the normal fuzzy rules system what we do is to activate all the background rules. Those rules may be from a single default output value up to a complete system based on distance to background rules (the membership is always greater than zero so it always has a value). Using the background rules

we improve the system because we do not restrict the freedom of tuning of the fuzzy rules of the normal system. Many background rule systems allow tuning, for example, our gem background rules system.

Our gem (global enveloping method) background rules system is based in a set of what we call gem rules. A point in the input space and a value for the output space composes those rules, which means, as can be seen, a minimal memory waste. If the normal fuzzy rule system does not cover a given point, that is, when there is not any rule R_h that verifies $(x \text{ is } A_h)$ $h = 1, 2, \ldots, k$ for a given x, we calculate the output value of the system using the gem rules. We describe in each of the following sections how gem rules are created, tuned, and how they infer.

1. *Clustering Agents*

Next we explain some of the different clustering agents our multiagent architecture can integrate.

We consider that we know, in some way, the existence of k clusters in the data. We do not need to know any other information of the data structure (for example, linearity or nonlinearity). We will use a clustering algorithm to find k centroids. An important aspect to stand out is that the clustering process is performed over the input and output product space. The objective of this process is, as is shown in several works,[22,23] to perform a better detection of the clusters that exists in the data. This is because not only is the interactivity between input product spaces taken into account but also the consequences of its interrelation with the output space. So the clustering will be performed over the $(X^p \times Y)$, that is, we will also take into account the output values of the examples. In this way, we obtain k centroids C_h with this form $(c_{h_1}, c_{h_2}, \ldots, c_{h_p}, c_{h_{p+1}})$, $h = 1, 2, \ldots, k$ where $c_{h_1}, c_{h_2}, \ldots, c_{h_p}$ belong, respectively, to the domain X_1, X_2, K, X_p, and $c_{h_{p+1}}$ belongs to the domain of Y.

Method CL1. This method gets as input a set of pairs of input and output values. Our objective is to perform a data clustering, and be able to detect in this way the associated centroids to the different detectable clusterings. We are going to use a modified version of the classical Kohonen self-organizative algorithm as proposed in Ref. 5. So in each loop it is not only the "winner" centroid or the closest to the learning sample that it is modified, but a set of centroids of the vicinity, vicinity which decreases when the number of iterations increases.

The method works in this way. We first generate k random centroids inside the data space. Then, for each input sample $x(t)$ we measure the Euclidean distance between each center $c_i(t)$, $i = 1, 2, \ldots, k$ and that input sample $x(t)$ and then select the closest winner center $c_c(t)$ according to $\|x(t) - c_c(t)\| = \min\{\|x(t) - c_i(t)\|\}$. The winner center $c_c(t)$ and some centers belonging to the set $N_c(t)$ are moved toward the input sample $x(t)$ according to

$$c_i(t+1) = \begin{cases} c_i(t) + g_c(t) \cdot [x(t) - c_i(t)] & \forall i \in N_c(t) \\ c_i(t) & \forall i \notin N_c(t) \end{cases} \tag{2}$$

where $g_c(t)$ is a monotonously decreasing learning rate ($1 > g_c(t) \geq 0$), and the centers set $N_c(t)$ is composed by the nth closest centers to $c_c(t)$, the winner center. This number n decreases with each iteration down to zero. After it reaches zero only the winner center is modified. The parameters of this method are the starting n, and the number of iterations it needs to reach the zero value. The other parameters are the data set, the number of global iterations and the starting and ending learning rate $g_c(t)$.

Method CL2 (Fuzzy Tabu Clustering). The tabu search is a heuristic that can be used to solve combinatorial optimization problems. It is different from the well known hill climbing local search techniques in the sense that it does not become trapped in local optimal solutions, i.e., the tabu search allows moves out of a current solution that makes the objective function worse in the hope that is eventually will achieve a better solution.[24]

In Ref. 25 we have presented a tabu search-based algorithm for fuzzy clustering. In this algorithm the objective function, the centroid and membership calculation are similar to that of the fuzzy C-means algorithm,[26] where the parameter m is the fuzziness of the clustering. Thus,

$$J = \sum_{i}^{k} \sum_{j}^{n} \|x_j - c_i\|^2 \cdot \mu_{c_i}^m(x_j)$$

$$c_i = \frac{\sum_{j=1}^{n} x_j \mu_{c_i}^m(x_j)}{\sum_{j=1}^{n} \mu_{c_j}^m(x_j)}$$

$$\mu_{c_h}(x) = \frac{1}{\left(\sum_{i=1}^{n} \|x - c_h\|^2 / \|x - c_i\|^2\right)^{1/(m-1)}}$$

To simplify the explanation we will name as *configuration* a tuple composed of a centroids vector and the corresponding degrees of membership matrix, and it will be denoted as A_k. The following parameters are used: MTLS maximum tabu list size, P probability threshold, NTS number of trial solutions, ITMAX maximum number of iterations, and TLL tabu list length.

The algorithm can be described, by means of four simple steps, as follows:

(s1) *Initialization step*. Let A_0 be an arbitrary solution and J_0 be the corresponding objective function value computed using Eq. (1). Let $A_b = A_0$ and $J_b = J_0$. Select values for the following parameters: MTLS, P, NTS, and ITMAX. Let $k = 1$, let TLL = 0, and go to step (s2).

(s2) *Generation step*. Using A_b generate NTS trial solutions $A_k^1, A_k^2, \ldots, A_k^{\text{NTS}}$, and evaluate their corresponding objective function values $J_k^1, J_k^2, \ldots, J_k^{\text{NTS}}$, and go to step (s3).

(s3) *Selection step*. Order $J_k^1, J_k^2, \ldots, J_k^{\text{NTS}}$ in ascending order and denote them by $J_k^{[1]}, J_k^{[2]}, \ldots, J_k^{[\text{NTS}]}$. If $A_k^{[1]}$ is not tabu, or if it is tabu but $J_k^{[1]} < J_b$ then let $A_c = A_k^{[1]}$ and $J_c = J_k^{[1]}$, and go to step (s4); otherwise let $A_c = A_k^{[L]}$ and $J_c = J_k^{[L]}$, where $J_k^{[L]}$ is the best objective function of $J_k^{[1]}, J_k^{[2]}, \ldots, J_k^{[\text{NTS}]}$ that is not tabu and go to step (s4). If all $J_k^{[1]}, J_k^{[2]}, \ldots, J_k^{[\text{NTS}]}$ are tabu go to step (s2).

(s4) *Modification step*. Insert A_c at the bottom of the tabu list and let TLL = TLL + 1 (if TLL = MTLS + 1, delete the first element in the tabu list and let TLL = TLL − 1). If $J_b > J_c$, let $A_b = A_0$ and $J_b = J_c$. If $k >$ ITMAX, stop (A_b is the best solution found and J_b is the corresponding best objective function value); otherwise, let $k = k + 1$ and go to step (s2).

Different alternatives exist to generate the trial solutions, for example, to move the centroids over the space and then to calculate the new memberships matrix. For each centroid, if a random number is greater than P, the move is performed.

2. *Rule Generation Agents*

For simplicity we have chosen the simplified fuzzy rule form[27] where the fuzzy set of the consequent for each rule is changed for a simple value or singleton consequent (being, in our case, a real number). So all our methods will produce rules of this type,

$$R_h\text{: } \textbf{If } x \textbf{ is } A_h \textbf{ then } y \textbf{ is } v_h \qquad h = 1, \ldots, k \tag{3}$$

The membership value A_h is an interpretation of the expression of the antecedent of the rules. Particularly, the rules can be expressed as

$$R_h\text{: } \textbf{If } x_1 \textbf{ is } A_{1h} \textbf{ and } x_2 \textbf{ is } A_{2h} \textbf{ and } \cdots \textbf{ and } x_p \textbf{ is } A_{ph} \textbf{ then } y \textbf{ is } v_h \qquad h = 1, \ldots, k \tag{4}$$

Each one of the expressions (x_j **is** A_{jh}) is interpreted as a *membership degree* $\mu_{A_{jh}}(x_j)$ of input value x_j to the fuzzy set A_{jh}. Such membership degrees are defined by the membership function that depends of the kind of fuzzy number used. Each rule also needs an initial v_h output value. So, basically, each method has to:

(a) Define the number of rules to be created.
(b) Set the centers of the A_{jh} fuzzy numbers that constitute the antecedent of the rules.
(c) Define the *membership degree* $\mu_{A_{jh}}(x_j)$ to be used.
(d) Set the output value (v_h).

Now we can explain how we create our gem background rules because they are independent to the fuzzy rule system the gem is the background of. In our case, we create as many gem rules as fuzzy rules above them, but this number could be different. We also use as centers of the gem rules the centers of the fuzzy rules.‖ As the output value we use the mean of the outputs of the training data multiplied by a membership function that preserves $\sum_j \mu_{c_h}(x) = 1$, $\forall x$. In

‖There is no special reason to do this. We just use them to avoid calculating other centers. In our conclusion and future trends section we propose other ways to find centers for such gem rules, but we have not yet tested or formalized them.

this way, any input data x (not only the training data) will have a gem rule that covers it (actually it is covered by all of them). So the output value is calculated by

$$v_h = \frac{\sum_j \mu_{c_h}(x_j) \cdot y_j}{\sum_j \mu_{ch}(x_1)} \tag{5}$$

and the membership function we use is

$$\mu_{c_h}(x) = \frac{1}{\left(\sum_{i=1}^{n} \|x - c_h\|^2 / \|x - c_i\|^2\right)} \tag{6}$$

Independently of the gem rules, two methods to generate the fuzzy rules have been implemented:

Method RG1 (Pyramid). This method gets as input a set of centroids. It will create a rule for each centroid, and it will use triangular fuzzy numbers. The fuzzy sets of A_h are generated with center equal to the centroid. To calculate the width we assign to each example a centroid (particularly the centroid closest to the example) and check, for each centroid, among the examples assigned to it, which one is the farthest. The width of the fuzzy set generated by such a centroid will always be greater than this distance. The reason to do so is to ensure that all training data samples are members of at least one fuzzy set.¶ It has to be slightly greater to avoid having an element (the one that is furthest away) with a membership value of zero (see the membership functions below), which is equivalent to a nonmembership. This will also ensure that at least one rule will be fired for each example.

This method will use triangular fuzzy numbers. Two numbers, center (c) and base (b) define such fuzzy numbers. Its membership function is

$$\mu_{A_{jh}}(x_j) = \begin{cases} 1 - \dfrac{2 \cdot \|x_j - c_{jh}\|}{b_{jh}} & c_{jh} - b_{jh}/2 \le x_j \le c_{jh} + b_{jh}/2 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The output value is calculated as the mean of the outputs of the examples that fire it, weighted by the membership value of such an example. If we have n examples of p dimensions in the form $e = [(x_1, y_1), (x_2, y_2), K, (x_n, y_n)]$ with $x_i \in X^p$ and $y_i \in Y$, the initial rule output value v_h is expressed in this way,

$$v_h = \frac{\sum_{j=1}^{n} \mu_{A_h}(x_j) \cdot y_j}{\sum_{j=1}^{n} \mu_{A_h}(x_j)} \tag{8}$$

¶We should note here that this does not ensure that all of the input space is covered, it only ensures that all the training data samples are covered. However, there is no problem because the gaps are covered by the gem rules.

where $\mu_{A_h}(x_j)$ is the *firing strength* of the rule R_h for the example x_j. There are several ways to express such firing strength. It can be defined as a measure of the global coincidence of each of the components of the example x_{ij}, $i = 1, 2, \ldots, p$ with each one of the fuzzy numbers A_{ih} that compose the rule (as we have shown before, such coincidence is measured by the membership value). In our case we will use the t-norm product (9) for the firing strengths. Thus,

$$\mu_{A_h} = \prod_{i=1}^{p} \mu_{A_{ih}}(x_{ij}) \tag{9}$$

Method RG2 (Matrix). This method is analogous to RG1 but the centers of the rules are not extracted from a clustering. What we do is to divide the input space into a matrix (each cell will be an hypercube of range equal to the dimension of the input) and we define the centers of the hypercubes to be the centers of the rules. This method creates an homogeneous distribution of the rules. The width of the rules is slightly greater than the hypercubes to allow a little overlapping of the rules. The rest of the parameters are calculated as in RG1.

3. *Rule Tuning Agents*

The set of rules generated with any of the previous methods could be a good approximation to the system to be modeled. We can calculate the error we have when we use them. As we only know the outputs of the examples, we can estimate the error we have when we approximate the function with them. We will use the mean square error,

$$E = \frac{\sum_{j=1}^{n} (y_j^* - y_j)^2}{n} \tag{10}$$

with y_j^* being the real value of the jth example and y_j the fuzzy rule system's inferred value for the jth example. (The inference process will be described below). Starting with those measures we can try to improve the fuzzy system trying to tune the rules. In particular, we should try to get the inferred values to be closer to the real ones. To do so we have created two approximations, one based on output tunes, and the other on gradient descent.

We must note that the gem rules can be tuned in a similar way as the fuzzy system, particularly the gem background rules are modified as follows. Each method that modifies the output value of a rule can also tune the gem background rules system. Instead of modifying the output value of a fuzzy rule we modify the output of all gem rules in this way. Hence,

$$v_h(t+1) = v_h(t) + \frac{\sum_j \mu_{c_h}(x_j) \cdot m}{\sum_j \mu_{c_h}(x_j)} \tag{11}$$

where m is the modification we would use to adjust the output of the fuzzy rule, as is shown next, and t is the iteration of the tuning process.

Method RT1. The method RT1 is an algorithm that tries to improve the global output by a set of individual improvements. This method modifies the v_h of the rules to get a y_j closer to y_j^*.

To fulfill this objective we will use the examples $e_j = (x_j, y_j^*)$ one by one. For each example we check how many rules are fired with the example and we obtain in this way a subset of them $P_j = \{R_h / \mu_{A_h}(x_j) > 0\}$. Among all the fired rules, we choose the one with the strongest firing strength $R_k \in P_j / v_h = \text{Max}(v_i \in P_j)$. We tune the output of this rule to make the rule output value v_k closer to the output value of the example y_j^* weighted by a learning rate $g_s(t)$ with $1 \geq g_s(t) \geq 0$ and weighted also by the firing strength of the rule $\mu_{A_k}(t)$, using the following criteria $v_k(t+1) = v_k(t) + g_s(t) \cdot \mu_{A_k}(t) \cdot [y_j^* - y_j]$. In this way the system global output, for the example will get closer to y_j^*.

If we use this method to tune the gem rules we use $m = g_s(t) \cdot [y_j^* - y_j]$.

The strong point of this method is its simplicity, but it does not ensure that the tune made by an example does not prejudice tunes made by other examples.

This method can be applied to any rule system with a singleton output value. It can also give intermediate results. At each iteration we know the best solution found so far, this means that the process can be interrupted without any working problem, given this solution, and resume the tuning process later at the same point.

Method RT2. This second method tries to improve the error using gradient descent. Using the error expression we obtain expressions of the dependence of y_j with regard to the output of the rules (the v_h) and with regard to the fuzzy numbers that fire them. So we can modify both the v_h and the fuzzy sets of the antecedent of rules. In particular, a descent method seeks for the vector Z, which minimizes an objective function $E(Z)$ where Z is a p-dimensional vector $Z = (z_1, z_2, K, z_p)$ of the tuning parameters. In this method, the vector that decreases the objective function $E(Z)$ is expressed by $(-\partial E/\partial z_1, -\partial E/\partial z_2, K, -\partial E/\partial z_q)$, and the learning rule is expressed in the following way,

$$z_i(t+1) = z_i(t) - K\frac{\partial E(Z)}{\partial z_i} \qquad (i = 1, 2, \ldots, q) \tag{12}$$

where t is the tune process iteration and K is a constant. In our case, the objective function to minimize is the error (2) of the fuzzy rule system.

In this method if we use triangular fuzzy numbers, as we described before, we see that the system depends on three different groups of variables: the centers of the fuzzy numbers (c_{ij}), the base of the fuzzy numbers (b_{ij}), and the consequents of the rules (v_i) with ($i = 1, 2, \ldots, k$) and ($j = 1, 2, \ldots, p$). So the vector Z has the form $(z_1, z_2, \ldots, z_q) = (c_{11}, \ldots, c_{np}, b_{11}, \ldots, b_{np}, v_1, \ldots, v_n)$. Developing we obtain

$$c_{ij}(t+1) = c_{ij}(t) - K_c \cdot \frac{\partial E}{\partial a_{ij}}$$

$$b_{ij}(t+1) = b_{ij}(t) - K_b \cdot \frac{\partial E}{\partial b_{ij}}$$

$$v_i(t+1) = v_i(t) - K_v \cdot \frac{\partial E}{\partial v_i}$$

The rest of the process is to perform an iterative application of these three expressions on the described variables. The process will finish when the reduction of the error between two iterations is less than a determined value. The constants that appear in the expressions let us tune the sharpness of the method.

In the gem rules system we use $m = -K_v \cdot \partial E / \partial v_i$.

This method can be developed for other kinds of fuzzy numbers. It can give intermediate results.

4. *Inference*

Once all the rules of the system have been defined we can use them to produce results with new examples. In the search for more accurate systems we have developed several kinds of fuzzy rules systems. In particular we have added a special feature called multiple fuzzy rule system with error that we describe in the next section.

Also we must remember that in order to enhance our fuzzy rule systems we have added to them background rules. The background rules solve the problem of fully covering the input space.

Method IN1 (Mizumoto Rule). One way to infer results is to use the Mizumoto's simplified reasoning method[27] which is defined as the mean of the singleton outputs of the rules fired by the point to be inferred weighted by the firing strength of such a rule for such an example. That is

$$y = \frac{\sum_{h=1}^{k} \mu_{A_h} v_h}{\sum_{h=1}^{k} \mu_{A_h}} \tag{13}$$

Of course, as the firing strength is defined through the membership values, if a different approach is used, the results can be different so it is very important to keep the same criteria when creating, tuning, or performing inference with a system.

If $\sum_{h=1}^{k} \mu_{A_h}$ is equal to zero then y is undetermined, which means that no rule covers the point, so our gem background rule system is activated. We then calculate the output value using the same expression as above (Mizumoto's simplified reasoning method) but with the firing strength of each gem rule, taken not as the multiplication of the membership degrees to each fuzzy number in the input space (9), but as the fuzzy membership degree to the center of the gem rule (6). If we have s gem rules c_j with $j = 1, 2, \ldots, s$, and each gem rule has an output value v_j associated, the we will calculate the membership degrees with

the expression (6). In addition, we will calculate the final output value with

$$y = \frac{\sum_{j=1}^{s} \mu_{c_j} v_j}{\sum_{j=1}^{s} \mu_{c_j}} = \sum_{j=1}^{s} \mu_{c_j} v_j \tag{14}$$

With this method, our system covers all the input space because $\sum_{j=1}^{s} \mu_{c_j} = 1$ so there is always an output value.

5. *Special Features*

Multiple Fuzzy Rules System with Error (MFRSWE). When we build our fuzzy rules systems we try to find behavior lines from a data set. If we have a data set that also includes the expected values for the data we can train our system and get a tuning of such rules to minimize the error we make when we approximate the underlying function. Due to the fact that the error we make with such data is measurable, it seems interesting to create a new rule system that model the error we make with the first rule system. If the system that models the error is good enough we can expect that the addition of both systems will produce a more accurate approximation to the original system. This process can be repeated to decrease the estimated mean error of the system up to the point where adding new error systems increases the mean error. The features of this approach fit very well to a distributed agent system. Given a rule system, and supposing that the agents system deduces that the estimated error of the rule system can be decreased we can request that an agent create a new rule system that models the error. When this agent finishes the work (it may be possible also that several agents may be concurrently modeling the same error using different parameter combinations as number of rules, number of iterations, etc...), the original rule system may include the rule subsystem and will incorporate it if the estimated mean error has in fact been decreased. The process may be repeated to model the system in the best possible way.

So we have fuzzy rule system that models a function φ: $X^p \to Y$ and a training set of pairs of (z_j, y_j^*) with $j = 1, 2, \ldots, m$, where y_j^* is the expected output of the system for each x_j. Let us name the fuzzy rules system as $f(x) = y$. We can create a new fuzzy rules system $f'(x) = f(x) + \varepsilon_f(x) = y'$ with $\varepsilon_f(x) = e$ being a fuzzy rule system trying to model the function φ': $X^p \to Y$ with $\varphi'(x) = (y - y^*)$, that is, the difference between the output of $f(x)$ and the real value of the function y^*. We can also create a new $f''(x) = f'(x) + \varepsilon_{f'}(x) = y''$ with $\varepsilon_{f'}(x) = e'$ modeling $\varphi''(x) = (y' - y^*)$ and repeat the process indefinitely up to a $f^n(x) = f^{n-1}(x) + \varepsilon_{f^{n-1}}(x) = y^n$ with $\varepsilon_{f^{n-1}}(x) = e^{n-1}$ modeling $\varphi^n(x) = (y^{n-1} - y^*)$, the point in which we have $n - 1$ error layers. We should stop the process when we detect that the error value of φ^n is greater than the error value of φ^{n-1} where the error is calculated using (10). When this happens it means that the error we were producing with $n - 2$ error layers was less than the error we produce with n layer so it means it is a good moment to stop.

We should note that this kind of system is additive, that is, that $f^n(x) = f(x) + \varepsilon_f(x) + \varepsilon_{f^1}(x) + \cdots + \varepsilon_{f^{n-1}}(x) = y + e + e' + \cdots + e^{n-1} = y^n$ and each e^g is calculated independently due to the fact that each layer is an independent fuzzy rule system so they can be calculated concurrently, hence it is parallelizable. This means that the addition of new layers will not delay an answer. So when we infer with this system we have an agent inferring at the same time for each layer and the results are added together.

Due to the fact that the quality of the model of the function to approximate depends a lot on some parameters (as, for example, the number of rules used) the system may be testing several combinations simultaneously (while there exists resources to do so), with different agents specialized in different techniques and/or tuning methods. An agent may be in charge of supervising the evolution of such tries, exploiting promising lines, and killing agents whose preliminary results are not satisfactory.

IV. EXPERIMENTAL RESULTS

The example presented in this section has been selected just to show the flexibility of the proposed architecture, not looking for accuracy of the function modeled. Hence the objective is to experiment with different combinations of our agents. We will show some results of our system. We use a single planner trying to model a function, the nonlinear Sinc function,

$$z = \text{Sinc}(x, y) = \frac{\text{Sin}(x)}{x} \cdot \frac{\text{Sin}(y)}{y}$$

As training data we have 225 random samples from an area insider the interval $[-10, 10] \times [-10, 10]$. Inside such an interval we have values of z, which are inside the $[-0.210401, 0.991247]$, as can be seen in Figure 2. The pairs (x, y) are used as input examples and the z as output examples, so we have $e_j = (x_j, y_j, z_j)$ with $j = 1, 2, \ldots, 225$.

Combining the use or not of gem rules, the clusterings CL1 and CL2, the use of error layers, and the tuning methods RT1 and RT2 we have five different fuzzy rules systems.

Name	Gem Rules	Error Layers	Clustering	Rule Generation	Rule Tuning	Inference
FRS1	Yes	5	CL1	RG1	RT2	IN1
FRS2	Yes	None	CL2	RG1	RT2	IN1
FRS3	No	None	CL1	RG1	RT2	IN1
FRS4	No	None	CL2	RG1	RT2	IN1
FRS5	Yes	None	CL2	RG1	RT1	IN1

Figure 2. The method CL1 and CL2 was applied to create 10 clusters. We applied the tuning method RT1 using $g_s(t)$ from 0.005 up to 0.001. We applied the tuning method RT2 using the constrants $K_c = K_b = K_v = 0.2$. The error layers of FRS1 use the same characteristics as the base layer, that is, CL1, RG1, RT2, and IN1.

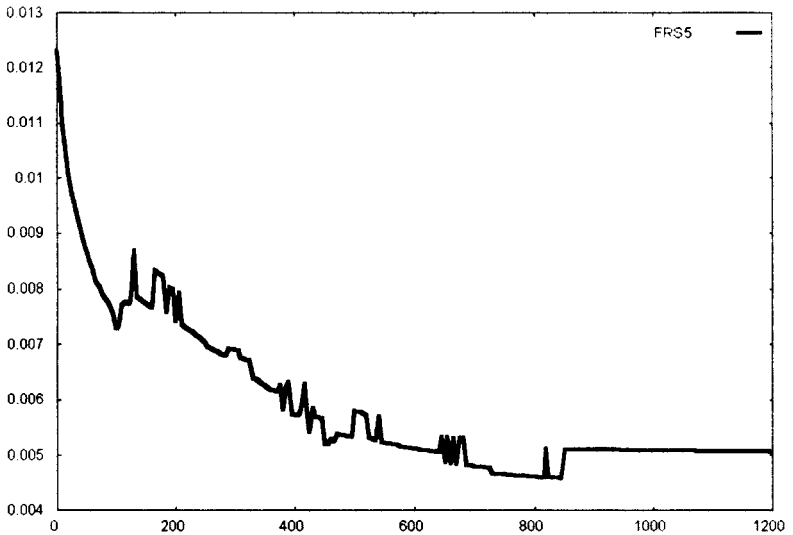

Figure 3. Evolution of the error of FRS5 forcing the tuning method to go on even if that worsens the error.

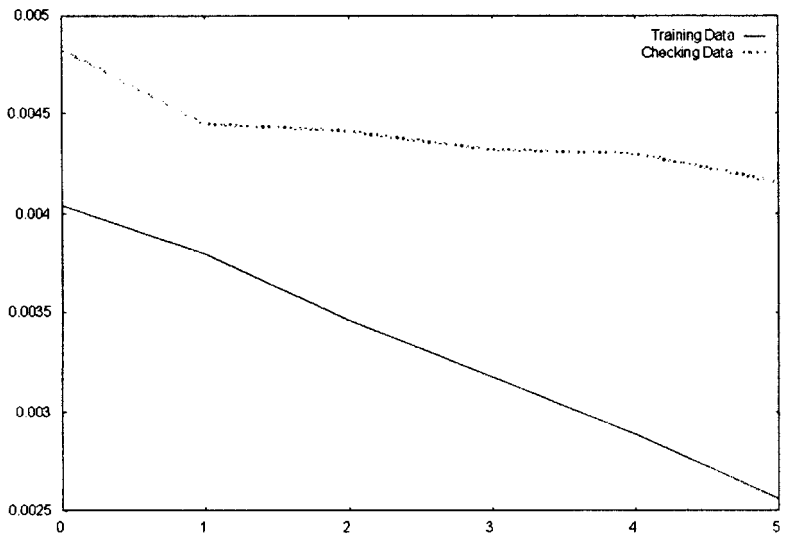

Figure 4. Comparison of the error obtained by FRS1 using from 0 to 5 error layers with the training data and with the check data.

We will now show several graphs of the results. Those results have been obtained using a validation set of 650 data samples different to the training sample.

In Figure 2 and Figure 5(a) we can see the differences between all methods. FRS5 uses RT1 which, as can be seen, decrease the error not as fast as RT2. It stops abruptly in epoch 71, this does not mean that it cannot be adjusted better but that is has been adjusted such that it became worse than epoch 70. As RT1 does not use gradient descent we do not ensure that the RT1 method continuously decreases the error so there may exist some hills in the error evolution as can be seen in Figure 3 where the same method has been forced to reach 1200 epochs.

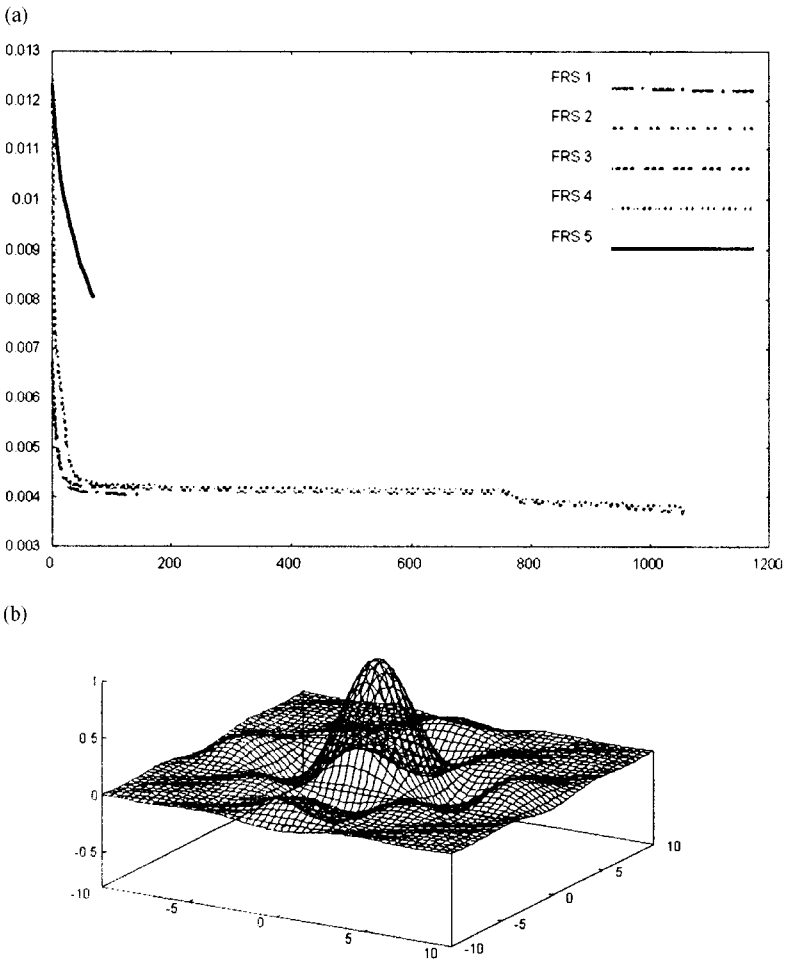

Figure 5. (a) Comparison of the error obtained by the fuzzy rules systems and the number of epochs needed in the tuning. (b) Three-dimensional representation of the function Sinc.

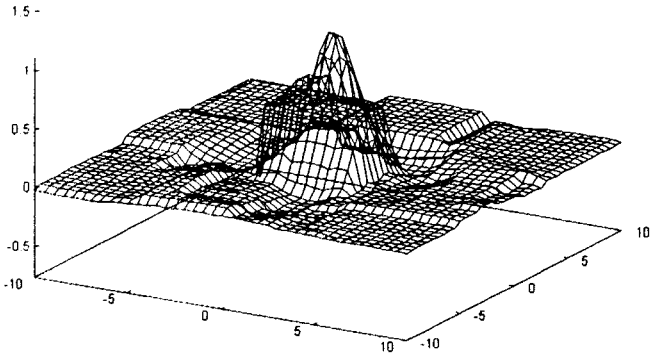

Figure 6. Function Sinc approximation using FRS5.

In Figure 4 we see how the error we obtain using error layers is less than when we do not use them, so our error layers increases the performance of the system.

In Figures 5–7 we appreciate the shape that has the real function and the approximation using FRS5 and FRS1.

V. CONCLUSIONS AND FUTURE TRENDS

The multiagent architecture proposed in this paper has proved a flexible tool to be used in the fuzzy modeling process. The possibility of adding new agents that take care of either the learning, the inference, the creation, or the tuning process gives us the possibility of testing different combinations of soft-computing techniques without continuously modifying the system.

Within the different methods there are several improvements that could be accomplished. The RT1 may be improved in two ways, modifying not only the output of the rule but also the fuzzy numbers of the antecedents and also avoiding the hills in the error. Buffering the best fuzzy rules group the tuning

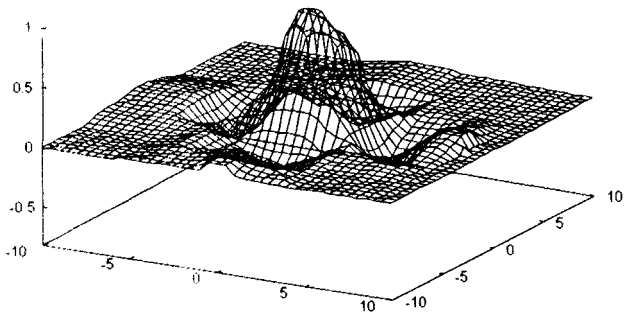

Figure 7. Function Sinc approximation using FRS1.

process has had and letting the process continue, even if it worsens the error, can do the latter. Basically it is a hill ascendant method then. It will continue giving intermediate results because it buffers the best one reached.

Another improvement that needs to be studied is how important the situation of the centers of the gem rules are. In the proposed methods the situation of the gem rules is not focused in the zones of the input space where there is not a rule covering or where the fuzzy rules do not give a good characterization of the system under study. We are studying a method that tries to discover these zones that are not well described by the fuzzy system (zones of high error), in order to be better covered by the gem rules.

We will also focus our research in the development of a hierarchical fuzzy rule system. These systems start with a number of rules but they can, in the tuning method, create new rules or eliminate them. They will expand the rules with high error, dividing them, trying in this way to better model the influence area of the rule. A natural improvement of this system will be the fusion of close rules with similar behavior to simplify the system. These systems will try to find a compromise between the number of rules (trying to use the minimum possible) and the error we make (trying to minimize it).

References

1. Wang, L; Langari, R. IEEE Trans Syst Man Cybern 1996, 26(1), 100–105.
2. Klawonn, F.; Kruse, R. Fuzzy Sets Syst 1997, 85, 177–193.
3. Bonissone, P. J Soft Comput 1997, 1 (1).
4. Ahn, T. Oh, S.; Woo, K. Fifth IFSA World Congress, 1993; pp 1181–1185.
5. Vourimaa, P. Fuzzy Sets Syst 1994, 66, 223–231.
6. Yager, R. R.; Filev, D. P. Technical Report MII-1318; Iona College, 1993.
7. Yoshinari, Y.; Pedrycz, W.; Hirota, K. Fuzzy Sets Syst 1993, 54, 157–165.
8. Berenji, H. R.; Khedkar, P. S. Proc. of the IEEE Int. Conf. on Fuzzy Systems, 1993; pp 1402–1407.
9. Ishibuchi, H.; Nozaki, K.; Yamamoto, N.; Tanaka, H. Fuzzy Sets Syst 1994, 65, 237–253.
10. Sun, C. T. IEEE Trans Fuzzy Syst 1994; 2(1), 64–73.
11. Sugeno, M.; Yasukawa, T. IEEE Trans Fuzzy Syst 1993, 1(1), 7–31.
12. Sugeno, M.; Kang, G. T. Fuzzy Sets Syst 1988, 28 15–33.
13. Delgado, M.; Gómez Skarmeta, A. F.; Martín F. Third European Congress on Fuzzy and Intelligent Technologies and Soft Computing, Aachen, Germany, 1995; pp 810–814.
14. Delgado, M.; Gómez Skarmeta, A. F.; Martín, F. V Congreso Español sobre Tecnología y Lógica Fuzzy, Murcia, Spain, Sep 1995; pp 43–48.
15. Delgado, M.; Gómez Skarmeta, A. F.; Vila, A. Int J Approx Reason 1996, 14(4), 237–259.
16. Corchado, J. M.; Lees, B.; Fyfe, C.; Rees, N. An Automated Agent-Based Reasoning and Learning System Proc. of the Ninth Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'97), Madrid, June 1997.
17. Finin, T.; Labrou, Y.; Mayfield, J. Proc. Third Int. Conf. Information and Knowledge Management, ACM Press: New York, 1994.
18. Finin, T.; McKay, D.; Fritzson, R.; McEntire, R. In Knowledge Building and Knowledge sharing; Fuchi, K.; Yohoi, T. Eds.; Ohmasha and IOS Press: 1994.
19. Finin, T.; Weber, J. ARPA Knowledge Sharing Inititaive, External Interfaces Working Group, Group working paper, July 1993.

20. Finin, T.; Fritzson, R.; McKay, D. McEntire, R. Technical Report, Department of Computer Science, University of Maryland, 1992.
21. Jennings, N.; O'Hare, G. Foundations of Distributed Artificial Intelligence; Wiley: New York, 1996.
22. Kroll, A. Fuzzy Sets Syst 1996, 80, 149–158.
23. Langari, R.; Wang, L. Fuzzy Sets Syst 1996, 79, 141–150.
24. Al-Sultan, K. S. Pattern Recognit 1995, 28(2), 1443–1451.
25. Delgado, M.; Gómez Skarmeta, A. F.; Martinez Barberá, H. 6th IEEE Int. Conf. on Fuzzy Systems, Barcelona, Spain, 1997.
26. Liu, J.; Xie, W. Proc. of the IEEE Int. Conf. on Fuzzy Systems, 1995.
27. Mizumoto, M. J Soc Instrument Contr Eng 1989, 58, 959–963.
28. Delgado, M.; Gómez Skarmeta, A. F.; Gómez Marín-Blázquez, J.; Martinez Barberá, H. VII Congreso Español sobre Técnologias y Lógica Fuzzy, Estylf97, Tarragona, Spain, 1997.
29. Etzioni, O.; Weld, D. Commun ACM July 1994, 37(7), 72–76.
30. Etzioni, O.; Segal, R. Eds. Working Notes of the AAAI Spring Symp.: Software Agents, AAAI Press: 1994.
31. Gómez Skarmeta, A. F.; Delgado, M.; Martín, F., Sixth Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems IPMU'96 Granada, Spain, pp 563–569.
32. Lander, S. E. IEEE Expert Mar.–Apr. 1997, 18–26.
33. Nwana, H. S. Knowledge Eng Rev 1996.
34. Sycara, K.; Pannu, A.; Williamson, M.; Zeng, D.; Decker, K. IEEE Expert Dec. 1996, 36–45.
35. Wang. L.; Mendel, J. M., IEEE Trans Syst Man Cybern 1992, 22(6), 1414–1427.
36. Zadeh, L. A. Fuzzy Sets Inform Contr. 1965, 8, 338–353.