FuGeNeSys: A Fuzzy Genetic Neural System for Fuzzy Modeling

Article *in* IEEE Transactions on Fuzzy Systems · September 1998

DOI: 10.1109/91.705506 · Source: IEEE Xplore

1 author:

Marco Russo
University of Catania
166 PUBLICATIONS **1,598** CITATIONS

FuGeNeSys—A Fuzzy Genetic Neural System for Fuzzy Modeling

Marco Russo

***Abstract*—The author has developed a novel approach to fuzzy modeling from input–output data. Using the basic techniques of soft computing, the method allows supervised approximation of multi-input multi-output (MIMO) systems. Typically, a small number of rules are produced. The learning capacity of FuGeNeSys is considerable, as is shown by the numerous applications developed. The paper gives a significant example of how the fuzzy models developed are generally better than those to be found in literature as concerns simplicity and both approximation and classification capabilities.**

***Index Terms*— Fuzzy neural networks, genetic algorithms, learning systems.**

I. Introduction

THE term soft computing (SC) was recently coined by Zadeh [1]. In the paper in which he uses this new term, he introduces the concept of machine intelligence quotient (MIQ) and states that the current trend in research is toward producing machines with an increasingly higher MIQ.

An increase in MIQ is, in fact, being achieved in two totally separate directions.

On the one hand, there are machines based on classical logic and, thus, on precision, which are evolving toward faster and faster machines with an increasing degree of parallelism.

On the other hand, new forms of logic are being developed such as fuzzy logic (FL), neural networks (NN's), and probabilistic reasoning (PR) [comprising genetic algorithms (GA's)] the strength of which (by contrast) lies in their capacity to deal with uncertainty and imprecision.

In the former, case solutions are found for problems that are very precise and, consequently, have a high computational cost. This is why Zadeh speaks of hard computing (HC).

In the latter case, on the other hand, he speaks of SC as imprecise or uncertain solutions can be found at a much lower cost in terms of calculation effort.

There are a number of cases in which excessive precision is quite useless so a nontraditional approach is preferable. In some problems, this is the only way because the computational complexity of a classical approach would be prohibitive.

Fig. 1 shows, at least as far as FL, NN's, and GA's are concerned, how the various components of SC can be approximately ordered on a time scale and on a scale relating to their learning capacity.

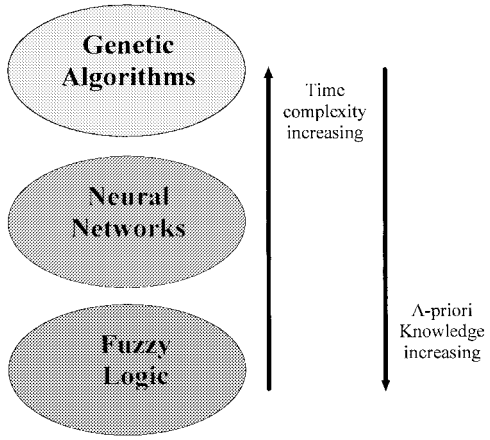

Fig. 1. Soft computing.

The time scale shown in Fig. 1 is ordered according to the learning time. FL, as conceived of by Zadeh, is not capable of learning anything. NN's and GA's (on the other hand) have this capacity although it can be said that on average pure GA's generally need a longer learning time. From another viewpoint, the order is inverted. GA's, in fact, need no *a priori* knowledge, NN's need very little and FL at times needs quite detailed knowledge of the problem to be solved.

In effect, each of these three areas of SC has its advantages and disadvantages. FL does not share the inherent concept of NN's, i.e., automatic learning. So it is impossible to use when experts are not available. It does, however, have a great advantage over the other two techniques. Expressed according to fuzzy canons, the knowledge base is computationally much less complex and the linguistic representation is very close to human reasoning.

NN's, at least as regards the typical features of gradient descendent learning networks, are quite different. In the first place, they were conceived of specifically for learning. They are, therefore, fundamental when all that is available is some significant examples of the problem to be solved rather than a solution algorithm. There are two evident disadvantages in using NN's. In general, they can learn correctly from examples, but what is learned is not easy for humans to understand, i.e., the knowledge base extracted from them does not have such an intuitive representation as that provided, for example, by FL. Second, the type of functions that can be used in NN's have to possess precise regularity features and the derivative of these functions has to be known *a priori*.

Manuscript received April 2, 1996; revised June 29, 1997.

The author is with the Institute of Computer Science and Telecommunications, Faculty of Engineering, University of Catania, Catania, 95125 Italy.

Publisher Item Identifier S 1063-6706(98)05153-4.

Similar considerations hold for GA's, with certain clarifications. Their learning speed is usually slower. However, they have two great advantages over NN's. The functions that can be used in GA's can be much more general in nature and knowledge of the gradient of the functions is not usually required. Finally, as these algorithms explore in several directions at the same time, they are affected much less than NN's by the problem of local extremes; that is, a GA has far less likelihood than an NN of finding a local extreme rather than a global one. Even if the extreme found is not a global one, it is likely to correspond to a less significant learning error.

On the basis of these considerations, it is the author's opinion that a technique which makes use of a combination of SC aspects, i.e., GA's, NN's, and FL, would be an interesting prospect. A hybrid technique, in fact, would inherit all the advantages, but not the less desirable features of the single SC components.

It has to possess a good learning capacity, a better learning time than that of pure GA's, and less sensitivity to the problem of local extremes than NN's. In addition, it has to generate a fuzzy knowledge base, which has a linguistic representation and a very low degree of computational complexity.

This paper will give a detailed illustration of a tool developed by the author called FuGeNeSys [2], which is based on a mixed technique GA's+NN's+FL.

The tool allows supervised learning of fuzzy rules from significant examples. The main features of FuGeNeSys are as follows.

- The number R of necessary rules is always very low. In the various author developed applications [2]–[15], this number has almost always been below ten. This means that R is not tied to the number of inputs as in the previous coding methods, but only to the complexity of the application itself. In the Section V-C he will explicitly show this peculiarity.
- The original genetic coding method proposed does not explode as the number I of input variables and the number O of outputs increase. The total number of bits is proportional to I and O and not to the power of I.
- Simplified fuzzy knowledge bases can be generated, i.e., the tool is capable of eliminating the unnecessary antecedents in any rule. This is proved in all fuzzy inferences showed in the paper and in those presented in author's previous works.
- Significant features are correctly identified. A whole section (V-B) is devoted to show this characteristic.
- The tool can be used in both classification and interpolation problems (see Sections V-A and V-C).
  a) The misclassification error is comparable to, if not better than, that of other similar techniques described in literature.
  b) Also, the function approximation error is very good. The author compared his results with neuro fuzzy techniques. In the author's opinion, with this kind of problem, the neuro fuzzy approaches are the most accurate among all hybrid fuzzy learning methods present in literature.

The tool's learning capacity is also demonstrated by the various research activities the author has taken part in and in which FuGeNeSys has been used. The fields of application in which successful use has been made of FuGeNeSys range from pharmacology [3], [16] to telecommunications [4]–[8], from Hw-Sw codesign [9], [10] to high-energy physics [11]–[13], and from the design of Hopfield Networks [14], [15] to robotics [2].

FuGeNeSys has also been used to show the effect in [17] of an incorrect choice of the performance index for any supervised learning method in order to determine the validity of the learning method adopted.

The paper is organized as follows. Section II gives a detailed introduction to GA's. Section III is an introduction to FL and Section IV is a detailed description of FuGeNeSys. In Section V, FuGeNeSys performance is analyzed with varying user-defined parameters. There is also a subsection comparing FuGeNeSys with other techniques. Finally, Section VI gives the author's conclusions.

II. Genetic Algorithms

GA's are a vast class of stochastic algorithms used to solve optimization problems. They use models based on the natural process of adaptation of living species, the first concepts of which were introduced by Darwin [18] in 1859 with his theory of the evolution of species.

Various authors have proposed algorithms that simulate genetic systems [19]–[21]. Interesting overviews can also be found in [19], and [22]–[25].

A. The Natural Processes Behind Darwin's Theory

Behind the theory of evolution there are only four basic processes:

- selection;
- reproduction;
- mutation;
- competition.

Selection is the natural process whereby individuals that can reproduce are chosen.

Reproduction is the mechanism nature uses to transfer the genes of an individual to its offspring.

Mutation is the phenomenon by which, because of nondeterministic factors, the process of reproduction does not transfer genetic characteristics faithfully; that is, the resulting genes contain a number of "copying" errors.

Competition is a direct consequence of the growth of populations in environments with a limited number of resources.

B. Standard GA's

In this section, a summary of the basic steps on which classical GA's are based will be made.

Let us assume, for the sake of simplicity, that the function to be optimized is

$$f: \Re^n \rightarrow \Re^m \text{ con } m, n \in N \tag{1}$$

and, thus, that the space of solutions is represented by the vectors $\vec{s}$: $\vec{s} \in \Re^n$.

1) Choice of the Fitness Function: The first step is that of analyzing the optimization problem and identifying a suitable fitness function

$$\text{fit}: \Re^n \rightarrow \Re \tag{2}$$

which will make it possible to discern the better of two possible solutions $\vec{s_1}, \vec{s_2} \in \Re^n$.

It has to be chosen in such a way that the extreme of the function to be optimized (1) corresponds to the absolute minimum or maximum of the fitness function.

Sometimes the fitness function chosen is the function to be optimized if it has $\Re$ as its codomain, but this does not generally happen because, as we shall see, the type of fitness function chosen may greatly affect the result of the search for a solution.

2) Definition of Suitable Coding: One of the most interesting problems in GA's is coding the solution space.

Typically, any potential solution (*individual*) is coded as a vector $\vec{s}$ called *chromosome*, the elements of which are called *genes* and are situated in well-defined positions, usually indicated as *alleles*.

The best-known forms of coding [20], [21], [26] are strictly binary. Every possible solution is simply represented by a binary string $\vec{s} \in \{0,1\}^h$ with $h \in N$. In this type of coding we speak of a Boolean allele value.

Nothing, however, prevents allele values from being continuous or even structured (in [27] and [28], applications are described in which allele values are even LISP (processing language) expressions).

3) Initialization of the Population: The next step consists of generation, usually random, of a certain number P of individuals. These represent a set of solutions, which, due to the properties of GA's, evolve in such a way as to possess an increasing fitness value.[1]

4) Decoding and Calculation of Fitness: Each individual $\vec{s_i}$ in the population is decoded in such a way that it is possible to calculate the fitness function.

In the case of binary coding, for this to be done, a certain function has to be defined

$$\text{dec}: \{0,1\}^h \rightarrow \Re^n \text{ with } h, n \in N \tag{3}$$

which, when applied to a string $\vec{s}$ of h bits, returns a potential solution $\vec{x} \in \Re^n$ for which the fitness function fit $(\vec{x})$ can be calculated.

5) Calculation of the Reproduction Probability: There are various methods for choosing the individuals that are to reproduce. Typically, but not always, selection is random. One of the best-known criteria for random choice is that of roulette wheel selection [19]. By this method, the individuals with the highest fitness are selected stochastically. Hence, the "fittest"

[1] This assertion is true when the fitness function must be raised; vice versa the evolution evolves toward its minimization.

Parent # 1:	0 1 0	1 0 1 0 1 1 1	0 1 1 1
Parent # 2:	1 1 1	0 0 1 1 1 0 1	1 1 0 0
Offspring #1:	0 1 0	0 0 1 1 1 0 1	0 1 1 1
Offspring #2:	1 1 1	1 0 1 0 1 1 1	1 1 0 0

Fig. 2. An example of crossover with two "cuts."

individuals have a higher probability of reproducing. Each chromosome $\vec{s_i}$ of the population is chosen with a certain probability prob $(\vec{s_i})$ of reproduction. The better the fitness function, the higher this probability has to be. The reproduction probability is often defined directly from the fitness function [23]

$$\text{prob}(\vec{s_i}) = \frac{\text{fit}(\vec{s_i})}{\sum_{j=1}^{P} \text{fit}(\vec{s_j})} \tag{4}$$

when the fitness function is nonnegative. Using this reproduction probability makes it possible to simulate a sort of roulette, which stochastically indicates which individuals in the population can reproduce.

6) Generation of New Individuals: On the basis of the reproduction probability values, a new population of chromosomes is generated. The scheme followed in "constructing" these new chromosomes is quite simple. Starting from the chromosomes selected by the roulette, new strings of offspring are generated using specific genetic operators such as crossover and genetic mutation.

The crossover operator comprises the basic mechanism for redistributing genetic characteristics between the individuals. It is applied to two chromosomes called *parents* and creates two new individuals. They are generated by selecting one or more points at which the parents' strings are "cut." The new individuals will have a genetic code in which substrings of the genetic codes of both parents alternate. An example of crossover with two cuts is shown in Fig. 2.

The crossover algorithm described is the classical one. There are a number of different versions that various authors have used to enhance the performance of GA's when applied to specific optimization problems.

The mutation operator simply introduces the artificial concept of copying error. It serves to emulate the natural phenomenon whereby offspring sometimes happen to have genes with totally different characteristics from their parents because of errors due to various factors. This operator is simulated by introducing a certain mutation probability p_m according to which an offspring bit is stochastically negated.

Immediately after the generation of a new individual, the decoding operation has to be performed and its fitness calculated (see Section II-B.4).

7) Termination Condition: At this point, GA termination tests are performed. The global process is usually stopped when a valid solution to the problem has been found or when the time available for the search is up. If none of the

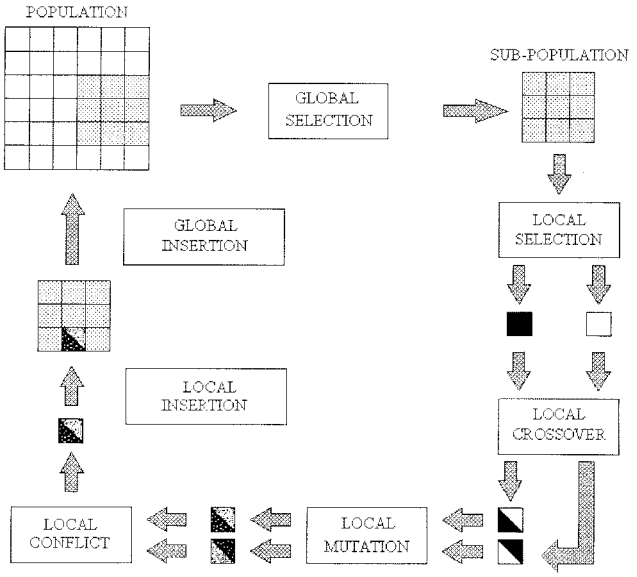

Fig. 3. Typical implementation of a fine-grain model.

termination tests gives a positive result, the steps outlined previously (starting from Section II-B.5) are repeated.

C. Distribuited and Parallel GA's

A population in which each individual can cross over with any other in each generation is called *panmictic*. This category includes the classical models such as those described above (Section II-B). The nature of GA's suggested implementing them on networks of computers or parallel computers (SIMD or MIMD machines). The main problem to be dealt with in this kind of implementation almost always relates to communication between parallel processes. It was, therefore, decided to use strategies with *apomictic* structures, i.e., ones which do not involve complete mixing. The idea behind apomictic populations is that of dividing the populations up into a certain number of subpopulations (often indicated as *demes*), which are connected to each other in some way.

There are three apomictic models.

- *Islands*—in this model migration can be between any pair of subpopulations;
- *Stepping Stone*—in this model genetic migration can only occur between adjacent demes;
- *Continuous*—in this model selection and crossover are local in an area with a preestablished radius and the isolation of demes is a direct consequence of the distance between them.

The last of these models gives rise to what are called *fine-grain* GA's in which the individuals are typically placed on a planar grid.

Fig. 3 shows a typical implementation of a fine-grain model. First, there is global selection to identify a subpopulation in the genetic grid. Then there is local selection within the selected deme to find two parents. Subsequently, the reproduction, crossover, and genetic mutation operations are performed. The last step is a local conflict to choose which of the offspring is to survive and which member of the subpopulation is to be replaced.

D. Considerations on Selection Algorithms

The task of the selection operator is to concentrate the computational effort toward regions of the solution space that appear to be promising.

To explain the concept of selection more clearly, the following are two diametrically opposed examples.

Let us assume that we are adopting a selection procedure which concentrates all the hardware resources on the most promising region in the search space. It is evident that in such a case, no computational resources are devoted to the search for other possibilities that, at present, seem less interesting but which in the future may lead to much better solutions. In this case, the GA can be said to degenerate into a local search method such as the well-known gradient descending method.

A uniformly random selection algorithm, on the other hand, distributes calculation resources over the whole genetic population. In this case, selection guarantees a search in several areas, but generally disperses a large amount of the hardware resources searching in sectors of the solution space that are of no use. In practice, this kind of selection behaves like a Monte Carlo method, randomly sampling the space of admissible solutions.

A compromise between these two types of selection is what is generally attempted in evolution-based algorithms.

Now the description of some of the best known selection methods follows.

1) Fitness-Proportional Selection: This selection model is the most widely used and was devised by Holland; it is a good tradeoff between the completely global method and an exclusively local one [see (4)].

The method does have some drawbacks, however. If, for instance, a certain fitness function $f_1(i)$ is used, use of another fitness function $f_2(i)$ built simply by adding an additional component to the first one $(f_2(i) = f_1(i) + c)$ should not make any diference to the performance a GA can achieve. This, however, does not happen. The selection probabilities may change considerably. Let us assume, for example, that we have three individuals with respective fitness values of 0, 10, and 90. Their probability of selection according to (4) is 0, 10, and 90%. If the fitness function adopted is equal to the previous one, except for an additional factor of 100, the probabilities decidedly change, becoming 25, 27.5, and 47.5%.

Another problem typical of fitness-proportional selection is that of the so-called *superindividuals*. A superindividual is a member of the population whose fitness value is much higher than the average value. In this case, in a very short time the population will become homogeneous because at each genetic iteration the number of individuals similar to the superindividual will increase considerably. If the superindividual represents coding of a solution close to the one being sought, the search time will be reduced drastically. Unfortunately, however, although the superindividual has a much higher fitness value than the other individuals, it often represents an unacceptable solution to the problem to be optimized. In this

case, the GA fails as the search proceeds very slowly. The crossover operator is, in fact, of no use as the individuals are very similar and evolution only proceeds thanks to the mutation operator. The GA thus degenerates into a Monte Carlo-type search method. This phenomenon is known as *premature GA convergence*.

2) Selection by Linear Ordering: Linear selection [24], [29] [often called linear ranking selection (LRS)] was first suggested by Baker [30] to eliminate some of the problems inherent in the fitness-proportional selection.

In the LRS, the P individuals are sorted according to their fitness values. The maximum rank (P) is assigned to the best individual and the lower rank (1) to the worst. The selection probability is linearly assigned to the individuals according to their rank

$$\mathrm{p}(i) = \frac{1}{P}\left(\beta - 2(\beta-1)\frac{i-1}{P-1}\right) \tag{5}$$

where β is a real number comprised in the interval [0,2].

3) Tournament Selection: An additional selection method, called tournament selection [19], [25] combines the idea of ranking in a very interesting and efficient way.

According to this method, first it needs to select h individuals and then the best one among those. This process is repeated a number of times P equal to the population size.

It is clear that large values of h increase selective pressure of this procedure; typical value accepted by many applications is $h = 2$ (so-called tournment size).

E. Classification of Selection Models

A scheme classifying the various selection models can be found in [31]. Using the same taxonomy, the following orthogonal classes emerge.

- *Dynamic static*—static selection is when the probability of selection depends on the fitness value, as in the case of fitness-proportional selection. Dynamic selection, on the other hand, is represented by ordering or local tournament;
- *Preservative Extinctive*—preservative selection guarantees that each individual will have a probability other than zero of being selected. This category includes fitness-proportional selection (if it is strictly positive) and selection by linear ordering if $\beta < 2$. In extinctive selection, on the other hand, some individuals are precluded from the possibility of reproducing. This is the case in selection by local tournament and linear ordering when $\beta \geq 2$.
- *Elitist Pure*—elitist selection guarantees selection of the fittest individual, while in pure selection the same individual may be discarded. It is important here to mention the work of Rudolph [32]: classical algorithms based on pure selection never converge, regardless of the type of crossover or mutation operators, the fitness value, or the type of initialization. Elitistic selection, on the other hand, allows convergence to the absolute extreme being sought.
- *Generational Stationary State*—in generational selection, the set of parents is determined only once until a new population of offspring is determined. Stationary-state selection is conceived of in such a way that the parents are extracted step by step and the offspring are gradually introduced into the population.

F. Nonstandard Genetic Operators

At times, to improve performance in terms of both convergence speed and accuracy, nonclassical genetic operators are introduced. A large number of such operators are described in literature, but the best-known would seem to be the hill-climbing operator. Given a certain individual, this operator finds an alternative similar individual who represents a local minimum close to the original individual in the space of solutions. Typical techniques are gradient descending ones. The author has used this algorithm and objectively confirmed its importance (see Section V).

G. Previous Fuzzy Logic Genetic Coding Methods

In Park *et al.* [33], only the conclusion part of the fuzzy inference and some control gains are coded. The total number of rules is a power of the number of inputs.

In Karr and Gentry [34], GA's are used to alter just the fuzzy set shapes of a predefinite rule base.

They adopt the so-called concatenated, mapped, unsigned binary coding. They use seven bits for each parameter. They work with trapezoidal membership functions (four parameters) or triangular membership functions (three parameters).

They fix the number n_i of the linguistic terms for each of the I inputs and the n_o one for each output. In the triangular membership function case, the number of bits in each individual is $3(\Sigma_{i=1}^{I}\ n_i + n_o)$.

The number of rules is equal to $\Pi_{i=1}^{I}\ n_i$. This number increases dramatically with the number of the I inputs.

In [35] and [36], there is another interesting genetic representation. They use the Takagi–Sugeno–Kang (TSK) fuzzy model [37] where the consequent part of a fuzzy rule is a weighted linear combination of the input values. For each rule with I inputs, it needs I membership functions for the antecedent part and $I+1$ weights for the consequent part.

They fix the maximum number n_i of triangular fuzzy sets for each input variable. For each one, they defined the membership function chromosome (MFC), which contains the parameters of a triangular shape. These are the eight bits of left base and right base and the distance from the previous center point (see Fig. 4).

For the consequent part, they introduce the rule–consequent parameters chromosome (RPC) that contains eight bits of each rule weight.

The total number of bits in each individual is $8(3\ \Sigma_{i=1}^{I}\ n_i + (I+1)\ \Pi_{i=1}^{I}\ n_i)$.

The maximum number of rules is $\Pi_{i=1}^{I}\ n_i$. This number increases dramatically with the number of the I inputs as well.

In the fitness function, they included an additional term to penalize systems according to the number of rules of the system.

In Ishibuchi *et al.* [38], GA's are used to select fuzzy rules from a given fuzzy database. Starting from the work in [39], they generate multiple fuzzy rule tables. Each table

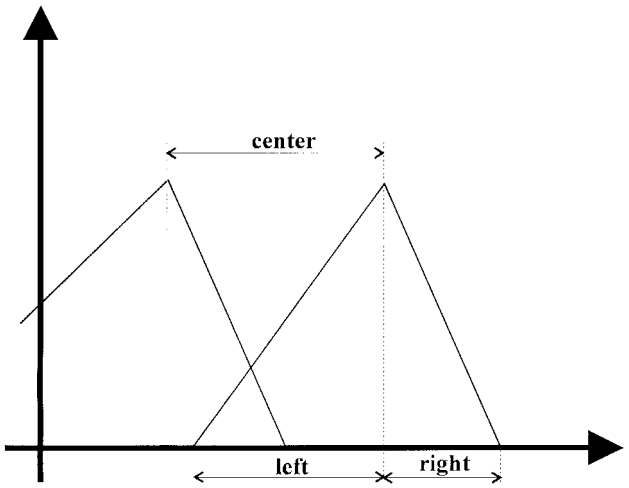

Fig. 4. Takagi and Lee's membership function representation.

corresponds to a different fuzzy partition. They start from a very coarse partition—only two membership functions per input—and continue with finer partitions. They consider a three values coding for each rule.

- $+1$ denotes that that rule belongs to the compact final fuzzy inference.
- -1 denotes that that rule does not belong.
- Zero denotes a dummy rule; that is, a rule that does not include any pattern and therefore is not useful.

Therefore, the final fuzzy inference is that composed of all the rules with its coding equal to one.

The main drawback of this method is that the number of starting fuzzy rules increases dramatically with the number of inputs, so the method cannot work with high-dimensional pattern spaces. The total number of bits is $\Sigma_{p=2}^{p_T} p^I$, where p_T is the finest partition of the input term set and I is the number of inputs. For example, with only two inputs and a partition equal to six, they need $90 \log_2 3 \approx 143$ bits. If $I = 6$ and $p_T = 6$, they need $\approx 106\,462$ bits.

III. Fuzzy Logic

A. Fuzzy Sets

In classical set theory, given a value $x \in X$ and a subset A of X ($A \subseteq X$), the element x either belongs to the set A or not. So it is possible to associate a Boolean numerical value (that is one belonging to the set $\{0, 1\}$) to each of the elements of X, specifying whether or not it belongs to A.

If we consider some of the most common adjectives used in daily conversation such as tall, intelligent, tired, ill, etc., we see that none of them is particularly precise: they are not used in terms of classical logic but in terms of a generalization with multiple values.

The concept of a *fuzzy set* is a step toward this kind of generalization. A fuzzy set [40] is a set whose membership function can assume all the values in the closed interval $[0, 1] \subseteq \Re$. A fuzzy subset A of a generic set X called the *universe of discourse* has the following membership function:

$$\mu_A \colon X \to [0, 1] \tag{6}$$

which associates a number $\mu_A(x)$ between zero and one to each element x of X.

B. Fuzzy Reasoning

In general, a fuzzy conditional rule is made up of a *premise* and a *conclusion*

$$\textbf{If } \text{premise } \textbf{then } \text{conclusion.} \tag{7}$$

The premise is made up of a number of fuzzy predicates P_i (henceforth, also called antecedents) of the general form (Tom IS *fast*) that are eventually negated or combined by different operators such as AND or OR computed with t-norms or t-conorms.

In the latter example, Tom is the value of the linguistic variable defined in the universe of the discourse of men and *fast* is one of the names of the *term set* of the linguistic variable (for example $\{slow,\ normal,\ fast\}$).

The following is an example of a fuzzy conditional rule using such operators:

$$\textbf{If } \mathrm{P}_1 \textbf{ and } \mathrm{P}_2 \textbf{ or } \mathrm{P}_3 \textbf{ then } \mathrm{P}_4 \tag{8}$$

where, for example

$$\begin{aligned} &\mathrm{P1} = (\text{car IS light}),\\ &\mathrm{P2} = (\text{hp IS average}),\\ &\mathrm{P3} = (\text{hp IS high}),\\ &\mathrm{P4} = (\text{car IS fast}). \end{aligned}$$

A fuzzy inference is made up of several rules with the same output variables. The inference procedure [41] establishes how the conclusions are to be inferred from this set of rules.

Fig. 5 gives a practical example of an inference, known as the max–min method. The figure outlines the calculation of the two rules

$$\begin{aligned} &\textbf{if} (x_1 \text{ is } A_{11}) \textbf{ and } (x_2 \text{ is } A_{12}) \textbf{ then } (y \text{ is } B_1)\\ &\textbf{if } (x_1 \text{ is } A_{21}) \textbf{ and } (x_2 \text{ is } A_{22}) \textbf{ then } (y \text{ is } B_2) \end{aligned}$$

assuming that $(x_1 \text{ is } A_1')$ and $(x_2 \text{ is } A_2')$ we obtain $(y \text{ is } B')$.

By this method the membership function of the fuzzy output set B' (still with reference to Fig. 5) is obtained as [42]

$$\mu_{B'}(y) = \overbrace{\max_r}^{\text{aggregation}} \underbrace{\min}_{\text{implication}} (\theta_r, \mu_{B_r}(y)) \tag{9}$$

where

$$\theta_r = \overbrace{\min_i}^{\text{conjunction in premise}} \alpha_{ir} \tag{10}$$

$$\alpha_{ir} = \overbrace{\overbrace{\sup_x}^{\text{projection}} \underbrace{\min}_{\text{combination}} (\mu_{A_i'}(x_i), \mu_{A_{ir}}(x_i))}^{\text{composition}}. \tag{11}$$

In (9), we can see an operation to *aggregate* the results (max) of the whole rulebase, preceded by an *implication* operation (min) based on the degree of activation of the single rules [see (10)] θ_r. This value is obtained by applying the logical connector min to the activation values α_{ir} of the single antecedents (11).

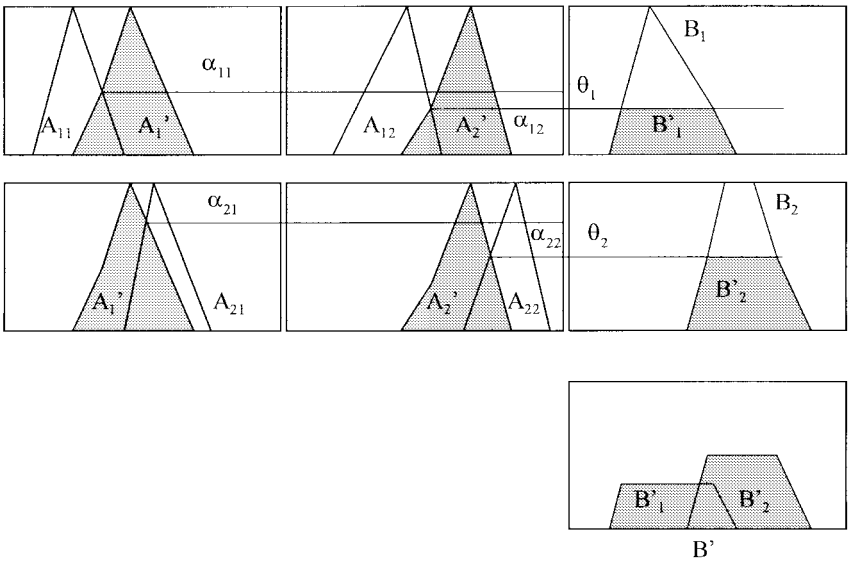

Fig. 5. The max–min method.

C. Fuzzification and Defuzzification

In a fuzzy system, the desired inputs and outputs are usually numerical values. It is, therefore, necessary to make a sort of "translation" from numerical inputs to fuzzy inputs and then from fuzzy outputs to the desired numerical outputs (*crisp* values). The first transformation is universally called *fuzzification* and the second *defuzzification*.

1) Fuzzification: Fuzzification consists of building the relation for the fuzzy input.

Inputs are normally crisp so this phase is necessary. Given n values x_i' of crisp inputs in a fuzzy system, it is necessary to construct the same number of fuzzy sets A_i', i.e., a fuzzy operator is needed for which

$$A_i' = \text{fuzz}(x_i'). \tag{12}$$

The fuzzification operation is often greatly simplified. In such cases, the numerical value is associated with the corresponding *singleton*

$$\mu_{A_i'}(x_i) = \begin{cases} 1, & \text{f } x_i = x_i' \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

At times, *fuzzy numbers* [40] are used to model uncertainty and imprecision.

2) Defuzzification Methods: There are various defuzzification methods.

Defuzzification is usually achieved by average calculation. One of the best-known methods is the center of gravity (COG) method, which is the method used to calculate the barycenter of a mass.

A discrete representation of COG is

$$\text{COG}(B') = \frac{\sum_{q=1}^{N_q} \mu_{B'}(y_q) y_q}{\sum_{q=1}^{N_q} \mu_{B'}(y_q)} \tag{14}$$

where N_q is the number of quantization steps by which the universe of discourse Y of the membership function $\mu_{B'}(y)$ is discretized.

Besides the classical methods, based on the concept of fuzzy output sets, there are a number of computationally simplified defuzzification methods by which it is possible to combine aggregation and defuzzification into a single phase.

One of them is the weighted-mean (WM) method in which

$$\text{WM}(B') = \frac{\sum_{r=1}^{R} \theta_r b_r}{\sum_{r=1}^{R} \theta_r} \tag{15}$$

where R is the number of rules, θ_r is the degree of activation of the rth rule, and b_r is a numerical values associated with the consequent B' of the rule. This value is usually calculated using another defuzzification method. In the Yager method [43], the value b_r is the mean value of the *alpha-level set* equal to θ_r; that is, b_r is the mean point of the segment obtained as the intersection between the fuzzy set of the consequent and a line parallel to the universe of discourse with a height of θ_r. The fuzzy sets are, of course, assumed to be convex.

The weighted-sum (WS) method [44] is even less computationally onerous, but this time the output of the single rule is relative, not absolute. The contributions of all the rules are, in fact, simply summed

$$\text{WS}(B') = \sum_{r=1}^{R} \theta_r b_r \tag{16}$$

where the symbols used are the same as those used to describe (15).

IV. FuGeNeSys

A. The Coding Used

Each individual in the population is made up of a set of R rules, each of which comprises I inputs (antecedents) and O outputs (consequents).

The membership functions are Gaussian of the following form:

$$\mu_i(x) = e^{-((x-c)^2/2\sigma^2)} \qquad 1 \le i \le I. \tag{17}$$

Therefore, to code these functions we need two kinds of information: the center c and the sigma σ.

In FuGeNeSys, it was decided to code c and the inverse of σ. This was done for two different reasons. The first was to eliminate the division operation and thus optimize the learning time. The second was to eliminate the annoying singularity in the equation of the Gaussian in the proximity of $\sigma = 0$. From the point of view of implementation, the Gaussian function considered is written as

$$\mu_i(x) = e^{-\gamma^2(x-c)^2} \qquad 1 \le i \le I. \tag{18}$$

For each antecedent it is therefore necessary to code information concerning a center c and the width value γ. A 16-bit coding was chosen for each of them. If the value of γ is equal to zero, it is assumed that in the rule in question the antecedent relating to the null γ is missing.

FuGeNeSys only considers crisp inputs, so the degree of membership α_{ir} of the ith antecedent of the rth rule corresponding to the ith crisp input x_i will be given by

$$\alpha_{ir} = e^{-\gamma_{ir}^2(x_i - c_{ir})^2} \qquad 1 \le i \le I, \quad 1 \le r \le R. \tag{19}$$

Various versions of FuGeNeSys have been implemented. Substantially, the difference between them is in calculation of the conclusion and the method used for defuzzification.

The consequents have been coded in two different ways. The first, which proved to be the most computationally complex, was used in the early versions of FuGeNeSys [3], [6]–[8], [10].

In the second type of coding, two defuzzification methods are possible, one of which is computationally quite simple.

The first method for coding the information about the consequents is based on the use of a sort of weighted mean which additionally associates a weight A_{or} to each rule and consequent. The following formula was used for defuzzification:

$$y_o = \frac{\sum_{r=1}^{R} \theta_r A_{or} C_{or}}{\sum_{r=1}^{R} \theta_r A_{or}} \tag{20}$$

where θ_r is the degree of activation of the premise of the rth rule, A_{or} is the weight of the oth consequent of the rth rule, and C_{or} is the output singleton. This formula coincides with the MAX–DOT method if it is assumed that the parameters C_{or} and A_{or} are, respectively, the center and area of any fuzzy set symmetrical with C_{or}.

It is useful to note here that to calculate the degree of truth of a generic premise, it was assumed that all the connectors in the premises were AND's for which the algebraic minimum operator was chosen, The degree of activation of a rule is thus given by the minimum degree of truth of the various antecedents in the rule

$$\theta_r = \min_{i=1}^{R}(\alpha_{ir}). \tag{21}$$

Using the defuzzification method shown in (20), two separate values were again stored for each consequent, both coded with 16 bits.

It became apparent that the learning capacity of FuGeNeSys did not deteriorate significantly if a single value—the output singleton—was coded for each consequent. The defuzzification method in this case is still of the centroid type, but the formula is

$$y_o = \frac{\sum_{r=1}^{R} \theta_r C_{or}}{\sum_{r=1}^{R} \theta_r}. \tag{22}$$

This method is clearly a WM defuzzification and is similar to the Yager method. So for each consequent a single value is coded using 16 bits. With this coding it is also possible to use a more simple defuzzification method such as WS

$$y_o = \sum_{r=1}^{R} \theta_r C_{or}. \tag{23}$$

Currently, one of these two methods can be chosen. In the various applications in which FuGeNeSys has been used, it has been found that the second one works better as far as the approximation of functions is concerned, while the first one is preferable in classification problems.

It is important to note that the first method (22) has the advantage that the knowledge base learned seems easier for a human to understand.

The WS defuzzification method, however, uses on average less time for a genetic iteration than the other methods as it is computationally less onerous.

Currently, each individual in the genetic population is represented by $(2I + O)R$ words, each of 16 bits. I, O, and R are, respectively, the number of inputs, outputs, and rules. $2 \cdot I \cdot R$ words are required to store the fuzzy sets of all premise of all rules $(\gamma_{ir}, c_{ir} \quad 1 \le i \le I, 1 \le r \le R)$. $O \cdot R$ words are required to store each consequrnt of each rule $(C_{or} \quad 1 \le o \le O,\ 1 \le r \le R)$.

B. The Evolution Algorithm Used

To implement FuGeNeSys, an apomictic, continuous, and fine-grain evolution algorithm was used. This was to allow the user to choose to slow down the process of genetic homogenization.

In practice, the individuals are assumed to be on a rectangular grid and are selected from a deme with a user-defined radius. The selection is static, preservative, elitist, steady-state, and fitness proportional.

As mentioned previously, in fine grain models there are two types of selection—global and local. In this case, global selection is fitness-proportional as in (4) and identifies the center of the subpopulation deme. Local selection uses the same equation but only in a subgrid with a pre-established radius.

Two parents generate a single offspring using a crossover operator with a single cut. Then the mutation operator is

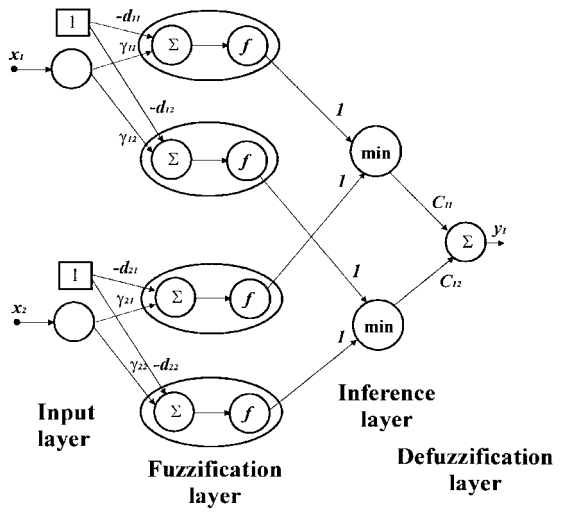

Fig. 6. The architecture of the neuro fuzzy system. The system has two inputs, one output, and two rules. It is based on the WS defuzzification.

applied and the offspring generated in the deme goes to replace the individual with the lowest fitness value.

Finally, a hill-climbing operator was introduced.

It starts whenever an individual is generated with a fitness value higher than the best one obtained so far.

In a first step, the individual selected is transformed in a neuro fuzzy system. That is, the equivalent neural network is built and all trainable weights are initiated. Then the system is trained. In this phase, it is possible that some neurons can be deleted, too. Finally, the trained system is retransformed in a genetic individual and introduced in the genetic grid at the place of the starting individual selected.

For simplicity, in the case of WS defuzzification the neuro fuzzy system has four layers. In Fig. 6, there is an example with only two inputs, one output, and two rules.

- The first is the input layer and it has at the maximum I neurons.
- The second one is the fuzzification layer. In this layer, there are calculated the activation degree α_{ir} of the antecedents. It is possible to distinguish the activation function $f = e^{-\mathrm{net}_{ir}^2} = \alpha_{ir}$ where $\mathrm{net}_{ir} = x_i\gamma_{ir} - d_{ir}$. Of course, $d_{ir} = c_{ir}\gamma_{ir}$. The weights that can be learned are γ_{ir} and d_{ir}. The total number of neurons is at the maximum $R \cdot I$.
- After, there is the inference layer. It needs to calculate the degree of truth of the R rules. It has at the maximum R neurons and weights fixed to one.
- Last, there is the linear output layer in which the WS defuzzufication is calculated. The weights that can be learned are C_{or}. The neurons are O.

The neural network is trained with a backpropagation procedure. This automatically adjusts all the learning speeds (one for each type of weight) and proceeds until the error decreases.

C. The Fitness Function Used

The fitness function was derived from a series of considerations dictated by experience. First of all, a function that was always greater than zero was chosen so as to be able to use stochastic selection (see Section II-B.5).

The function to be maximized is simply a rational fraction in which some parameters can be varied directly by the FuGeNeSys user.

The author has added some correction terms to improve the quality of the solution FuGeNeSys obtains, using a term whose weight is user-defined and forces the solution to search for simplified rules. In practice, the correction factor added privileges rules with few antecedents and/or consequents. Another correction factor that has been introduced allows the user to force the program to search for solutions that take the smallest possible number of features into account. Finally, a third factor has been introduced, again with a user-defined weight, which prevents the various rules from including some whose contribution is of negligible importance. In certain situations, this factor, which has to be used with extreme caution, allows the learning time to be speeded up considerably.

The analytical expression of the fitness function of an individual i is given by

$$\mathrm{fit}(i) = \frac{K_{\mathrm{num}}}{K_{\mathrm{den}} + K_{\mathrm{err}} \cdot e + K_{\mathrm{rid}} \cdot r + K_{\mathrm{tet}} \cdot f + K_{\mathrm{rid}_2} \cdot r_2}. \tag{24}$$

In (24), the coefficients K_{xxx} are the user defined numerical coefficients mentioned above. Let us define the parameters that appear in (24). The parameter e represents the learning error given by

$$e = 100\left(\frac{1}{OP}\sum_{p=1}^{P}\sum_{o=1}^{O}\left(\frac{y_{op} - y_{op}^d}{\Delta_o}\right)^2\right)^{1/2} \tag{25}$$

where P represents the total number of patterns, O the number of outputs for the function being approximated, and Δ_o the difference between the maximum and minimum values the oth output can have. Of course, y_{op} and y_{op}^d represent the oth output for the calculated and desired pth pattern. As e is the denominator of the fitness function expression, it is clear that the fitness value increases as the error decreases.

The parameter r indicates the ratio between the number of antecedents and consequents used in all the rules and the total number $R(I + O)$. If the coefficient K_{rid} is greater than zero and the other parameters are equal, the fitness of an individual representing coding of a "simpler" fuzzy inference is greater. So, making the coefficient $K_{\mathrm{rid}} > 0$ means forcing the FuGeNeSys program to search for less complicated solutions. This option can be used whenever it is necessary to reduce (as far as possible) the complexity of the fuzzy inference learned. It is also possible to use this feature to analyze the dependence of the fitness function on the features chosen. It may, in fact, happen that by forcing K_{rid} to sufficiently high values, some features are eliminated from the rules resulting from the learning phase.

Care must be taken, however, not to let the $K_{\mathrm{err}}/K_{\mathrm{rid}}$ ratio take excessively small values. When the ratio decreases, in fact, although increasingly simple rules are found, there is a collateral effect of an increase in the error e.

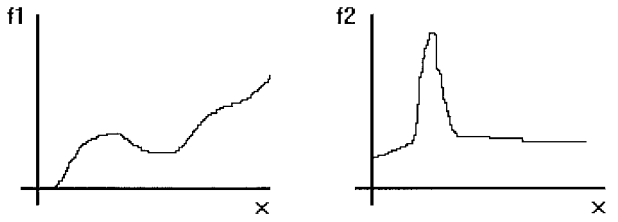

Fig. 7. Examples in which it doesn't need to utilize the option f (f2) and where it can help (f1).

The parameter r_2 is given by the percentage of inputs used in a fuzzy inference as compared with the total number. The considerations made for the parameter r hold again here, but this time they are restricted to the selection of features. The difference between the two parameters is that while the former simplifies the fuzzy rules by forcing the program to eliminate antecedents (and as an indirect consequence may lead to the elimination of some features), the latter is expressly meant to eliminate features. Consequently, it is more efficient in achieving this goal.

The last parameter f was introduced to try to give as reasonably equal as possible a "weight" to all the rules. The introduction of this parameter privileges fuzzy inferences in which each rule contributes significantly to the determination of a certain minimum number of patterns, as defined by the user. In other words, the use of f makes each rule contribute to the defuzzification process with a significantly high degree of truth for the premise θ for a certain number of patterns. So making the coefficient K_{tet} greater than zero forces FuGeNeSys to search for solutions, which present rules which all contribute significantly to the approximation or classification of a certain minimum number of learning patterns.

Use of this possibility does not necessarily lead to good solutions. If, in fact, we consider a function that presents quite a high derivative in a certain limited area of the domain (see $\text{f2}(x)$ in Fig. 7), it will probably be necessary to use one or more rules that make a nonnull contribution for a limited set of learning points as compared with the total number. In this case, an option such as the one just described has to be used, because the solutions found will very probably present a high error in the proximity of the area mentioned. This option must clearly be used with great caution.

The parameter f is defined by the following expression:

$$f = \begin{cases} 0, & \text{if } f > \text{freq} \\ 100\left(1 - \dfrac{f_{\text{req}}}{p_{\text{rule}}}\right), & \text{otherwise.} \end{cases} \tag{26}$$

The value freq is calculated as follows. R counters one for each rule, which we will indicate as freq_r with $r \in [1 \cdots R]$, are set to zero. For each learning pattern p we see which of the rules contributes with the highest truth value for the premise $\alpha_p^{\max}$. Then, still for the pattern p and each rule r whose premise has a truth value α_{pr}: $\alpha_{pr} \geq p_{teta}\alpha_p^{\max}$ the value freq_r is increased by one unit. So $\text{freq} = \min_{r=1}^{R}(\text{freq}_r)$. The two values p_{teta} and p_{rule} are user definable.

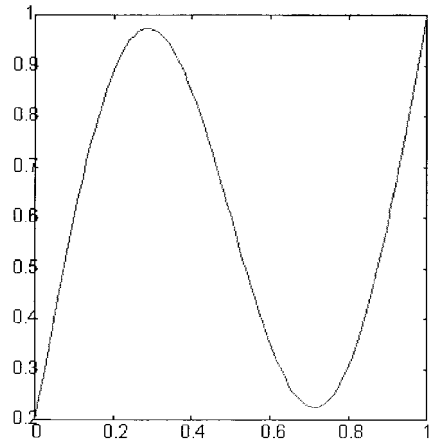

Fig. 8. The function chosen.

V. FuGeNeSys Evaluation

A. Relationship Among the Variable Parameters of FuGeNeSys and Its Learning Performance

This section gives results which relate the various magnitudes modifiable in FuGeNeSys to its learning performance.

As we shall see, numerous learning phases were run. For most of them a learning problem that is quite well known in literature was chosen

$$y = 0.2 + 0.8(x + 0.7\sin(2\pi x)). \tag{27}$$

In [45] and [44], the function (27) is approximated (see Fig. 8).

Both the learning and testing examples are taken to be uniformly distributed in the interval $[0, 1] \subseteq \Re$ as defined for the dependent variable x.

In this section, the author gives the results obtained relating the performance of FuGeNeSys to the size of the genetic grid and the number of iterations.

Fig. 9 shows the results obtained. The left-hand side of the histogram shows the percent learning error [see (33)], while the right-hand side shows the testing error (the learning and testing patterns are always the same and contain 20 and 101 points respectively). The x axis shows the number of iterations, the y axis the error obtained, and the remaining axis the size of the grid. All the simulations were performed using the WS defuzzification method. The procedure was as follows. Having established the size of the grid, a certain number of runs were performed until a pre-established confidence interval was reached, discarding all the runs where the learning error was higher than 5% and performing at least 50 runs. A few runs were discarded because the results are to be attributed to the phenomenon of premature convergence normally encountered in genetic algorithms. The confidence interval value chosen was one with which on average, 95% of the solutions fell within the value given in Fig. 9—more or less 0.1%.

The error depends directly on the two parameters considered. It decreases as the number of iterations and the size of the grid increase. In addition, as the size of the population

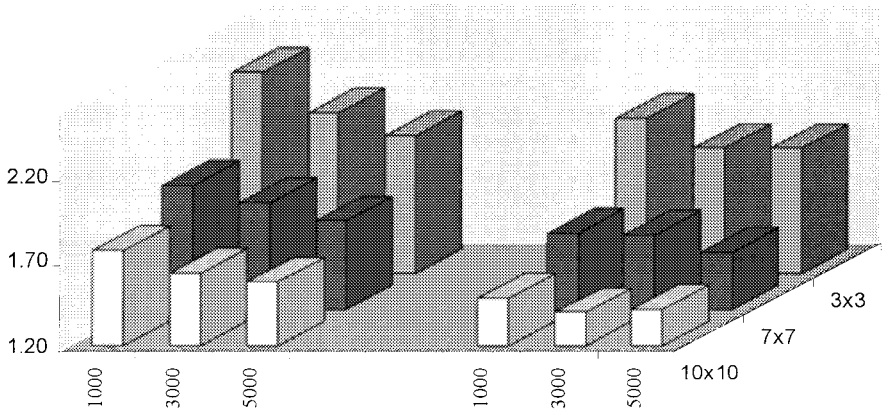

Fig. 9. Percent learning and testing error in function of the iteration number and grid dimension (WS defuzzification method).

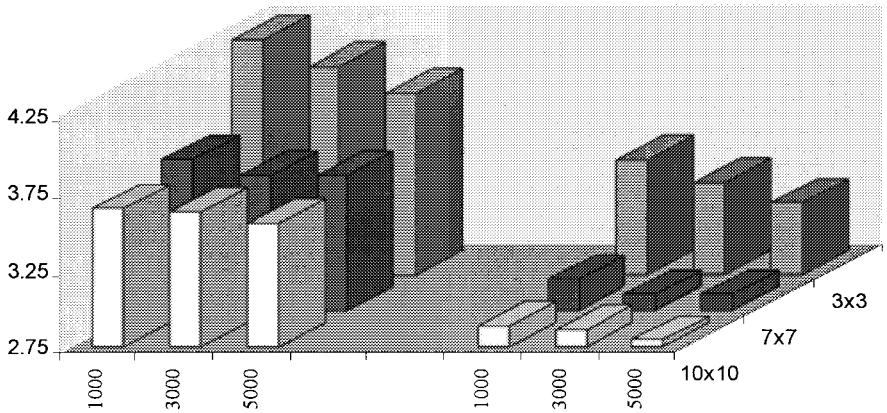

Fig. 10. Percent learning and testing error in function of the iteration number and grid dimension. (WM defuzzification method).

increases the performance of FuGeNeSys gets better, but to obtain the best performance, the number of genetic iterations must also be increased.

Using WM as the defuzzification method, we obtain the histogram shown in Fig. 10.

The behavior is similar to that shown in Fig. 9, but performance has clearly deteriorated. As the results were on average worse, the few simulations discarded were those which led to a learning error greater than 10%.

The obvious difference between the two results is to be attributed to the fact that the second defuzzification method is less suitable for this kind of function.

Where the WM method is suitable, on the other hand, is in problems of classification rather than interpolation. Fig. 11 shows an example of a function to be approximated in which the learning performance is decidedly in favor of this defuzzification method.

Figs. 12 and 13 give the results relating to the FuGeNeSys capability of generalization (testing) when the WS and WM methods are used.

In general, it can be said that the more "softly" a function varies, the more convenient it is to use the WS method. In contrast, the more "sharply" it varies, the more convenient it is to use the WM method (for example, see f1(x) and f2(x) in Fig. 7).

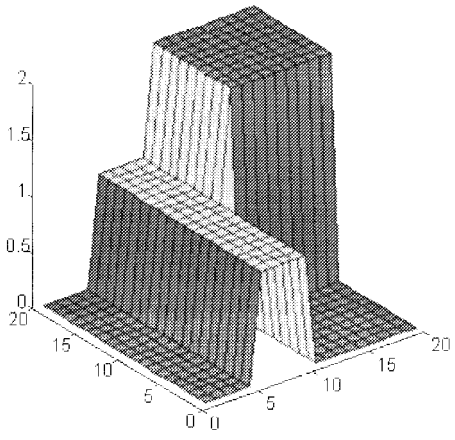

Fig. 11. The function used as problem of classification.

It should, however, be pointed out that the average amount of time required for an iteration clearly suggests using the less computationally onerous method, i.e., WS.

Another study performed is related to the importance of the introduction of the hill-climbing genetic operator, which involves backpropagation to the beginning of the generation of the grid every time a new, better individual is born.

Let us recall that backpropagation is performed as long as there are subsequent improvements. As soon as the opposite

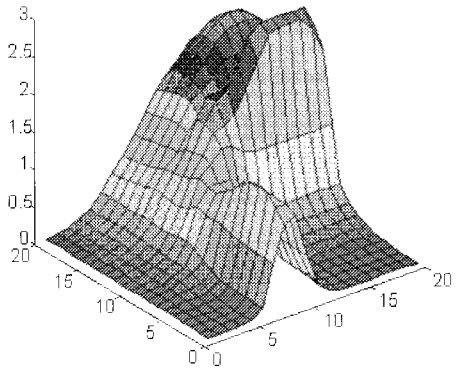

Fig. 12. Testing results using WS.

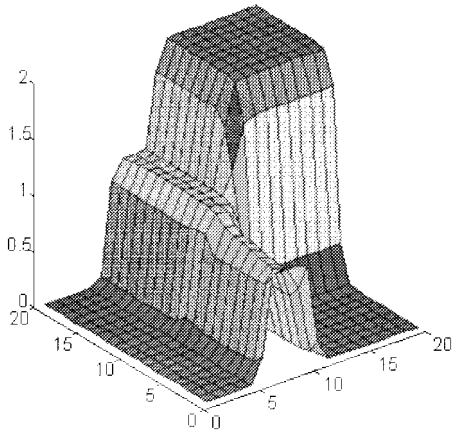

Fig. 13. Testing results using WM.

happens, the best solution obtained is kept. The weights of each type of parameter to be learned are dynamically adapted by means of an automatic procedure.

Fig. 14 shows the results of ten runs without the hill-climbing operator, using a grid of 10×10, five rules, 5000 iterations, and WS as the defuzzification method. The function to be approximated is still (27) The results obtained (using WS) are on average five to six times worse (see Fig. 9).

B. Learning with Features Detection

This section will illustrate how FuGeNeSys is able to identify features correctly.

As a significant example, we chose to construct some patterns to extract a fuzzy knowledge base which approximates the function

$$z = x^2 + y^2. \tag{28}$$

The 50 patterns generated to teach FuGeNeSys this pattern were built as follows. Each pattern has eight inputs $(x_i: 1 \leq i \leq 8)$ and one output (z). Two of the inputs correspond to the independent variables x and y, both randomly and uniformly generated in the interval [0,100]. Three inputs are linear combinations of x and y. The other three were generated

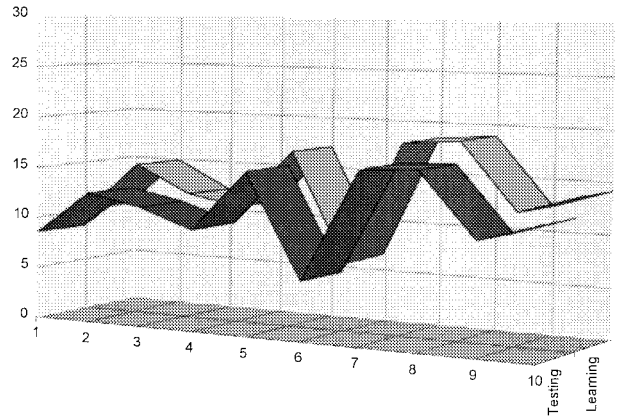

Fig. 14. Ten runs without backpropagation using WS as defuzzification method.

quite randomly. More precisely, we have

$$\begin{aligned} x_1 &= x \\ x_2 &= y \\ x_3 &= \text{random} \\ x_4 &= \text{random} \\ x_5 &= \text{random} \\ x_6 &= 2x \\ x_7 &= 3y \\ x_8 &= x + 2y. \end{aligned} \tag{29}$$

Table I gives the learning examples in detail. Of course, a learning program that can detect features correctly has to be able to eliminate the three completely random inputs. It also has to eliminate three more of the five remaining features so as to end up with only two independent ones.

Various runs were performed and each time the result obtained was as expected.

Table II gives the results of three different runs (1000 iterations) using WS as the defuzzification method. As can be seen, only two independent features are selected. Once again, the error is a monotonically decreasing function of the number of rules. The values attributed to the fitness function are the same, specifically $k_{\text{num}} = 100, k_{\text{den}} = 0.1, k_{\text{rid}} = 0$ and $k_{\text{rid}_2} = 0.1$.

Fig. 15 gives the results obtained with three rules. The universes of discourse of the three variables, x_6, x_7, and z, in the figure are [0,200], [60 300] and [0,17 000] respectively. For z, the singleton corresponding to the crisp output value is shown.

C. Comparisons

Some comparisons were made between the performance obtained with FuGeNeSys and that described in the literature.

1) Classification Problems: There are a lot of methods for generating fuzzy rules, but only a few approaches deal with classification tasks.

Typically, the generation method consists of two phases. The first is the fuzzy partition of the input pattern space

TABLE I
THE LEARNING SAMPLES

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Output
x	y	rnd	rnd	rnd	$2x$	$3y$	$x+2y$	z
92	92	82	99	44	184	276	276	16928
76	3	28	42	61	152	9	82	5785
41	43	46	88	89	82	129	127	3530
19	96	74	31	81	38	288	211	9577
71	83	38	91	79	142	249	237	11930
7	31	37	20	40	14	93	69	1010
64	94	70	33	15	128	282	252	12932
8	32	1	39	71	16	96	72	1088
8	97	84	33	22	16	291	202	9473
95	4	30	64	6	190	12	103	9041
72	68	82	85	52	144	204	208	9808
33	79	27	1	82	66	237	191	7330
54	29	74	87	53	108	87	112	3757
6	10	60	48	47	12	30	26	136
61	64	13	89	42	122	192	189	7817
19	38	2	75	15	38	114	95	1805
26	63	92	36	14	52	189	152	4645
0	78	11	88	7	0	234	156	6084
19	95	98	91	28	38	285	209	9386
91	70	56	23	34	182	210	231	13181
18	93	31	19	27	36	279	204	8973
76	80	18	51	25	152	240	236	12176
62	61	66	53	40	124	183	184	7565
16	27	48	48	66	32	81	70	985
85	98	96	92	92	170	294	281	16829
28	6	71	66	52	56	18	40	820
98	10	36	99	51	196	30	118	9704
4	36	22	51	51	8	108	76	1312
83	53	68	11	88	166	159	189	9698
12	57	41	47	28	24	171	126	3393
36	92	63	80	30	72	276	220	9760
97	41	21	30	47	194	123	179	11090
0	95	70	74	61	0	285	190	9025
3	66	38	43	22	6	198	135	4365
2	40	22	56	32	4	120	82	1604
61	10	3	73	48	122	30	81	3821
65	16	56	33	33	130	48	97	4481
60	13	33	15	28	120	39	86	3769
60	90	49	13	66	120	270	240	11700
14	17	20	60	93	28	51	48	485
63	99	70	11	22	126	297	261	13770
82	36	59	64	17	164	108	154	8020
9	75	39	66	62	18	225	159	5706
91	18	77	27	28	182	54	127	8605
51	86	0	25	3	102	258	223	9997
60	2	21	96	43	120	6	64	3604
57	96	29	2	79	114	288	249	12465
95	2	57	3	7	190	6	99	9029
2	75	77	66	92	4	225	152	5629
11	48	16	59	92	22	144	107	2425

TABLE II
FuGeNeSYS BEHAVIOR AS A FUNCTION OF THE NUMBER OF RULES

Rules	Error	Features
2	6.16%	1 and 8
3	3.76%	6 and 7
4	2.63%	2 and 6

into fuzzy subspaces and the second phase deals with the determination of the rule for each subspace.

The performance of this system depends on the choice of the fuzzy partition. If it is too fine, many rules cannot be generated because of the lack of training patterns. If the partition is too coarse, the performance may be too low.

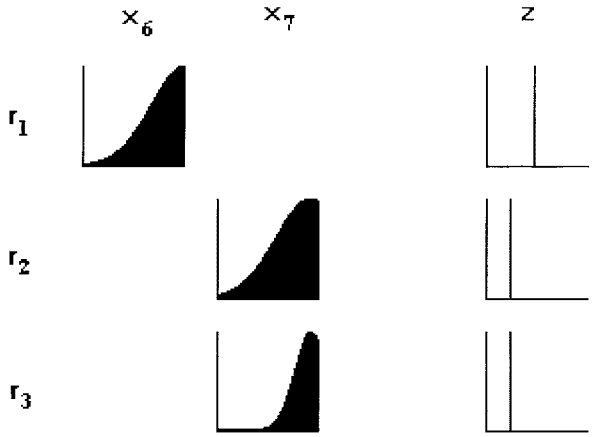

Fig. 15. The three rules learned.

This approach needs a lot of rules as the number of inputs increases as shown in Section II-G.

In [38], the authors use the concept of distributed fuzzy rules [39] where all fuzzy rules corresponding to several fuzzy partitions are simultaneously tested. They mix coarse and fine fuzzy partitions and with a GA select the minimum number of rules that permits the maximization of the number of correctly classified patterns.

This method works well. In fact, for example, with the well-known iris data [46] they obtain the correct classification of all the 150 patterns with only 13 rules.

The result obtained outperformed all the results reported in [47] for nine classification methods: fuzzy k nearest neighbor, fuzzy c-means, hierarchical fuzzy c-means, fuzzy integral with perceptron, fuzzy pattern matching with minimum operator, and others.

Iris data are a four input one output example. The number of rules was 2274 including 1582 dummy rules. So they constructed their compact fuzzy classification system by selecting only significant fuzzy rules from the remaining 692.

In this section the author shows the results obtained with the aid of FuGeNeSys for the same problem.

To the three classes iris setosa, virginica, and versicolor He respectively associated the integer numbers one, two, and three. He successfully trained a only five rules system using WM defuzzification that correctly classify all the 150 patterns.

In Fig. 16 the five rules are shown. The inputs are in the order sepal length (SL) in cm (in the range [4.3,7.9]), sepal width (SW) in cm (in the range [2.0,4.4]), petal length (PL) in cm (in the range [1.0,6.9]), and petal width (PW) in cm (in the range [0.1,2.5]). Naturally, the output is in the range [0,2]. If the output is below 0.5, the class is iris setosa; if it is comprised in [0.5,1.5], the class is iris virginic. Otherwise it is the remaining class.

FuGeNeSys thus has an excellent classification capacity and is capable of extracting extremely compact fuzzy inferences.

2) Approximation Problems: In [44] and [45], function (27) is used to test the learning capacity of the method proposed. In both cases, 21 points were taken as learning

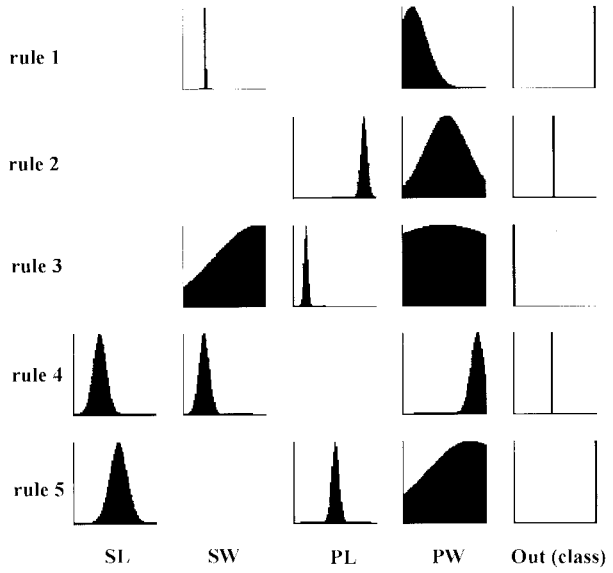

Fig. 16. The iris fuzzy classifier trained.

patterns and 101 uniformly distributed in the domain of the independent variable as testing patterns. The following formulas were taken as performance indexes for the learning and generalization capacity, respectively:

$$J_1 = \frac{100}{21} \sum_{i=0}^{20} \frac{|z_i - g(x_i)|}{g(x_i)} \% \tag{30}$$

$$J_2 = \frac{100}{101} \sum_{i=0}^{100} \frac{|z_i - g(x_i)|}{g(x_i)} \% \tag{31}$$

where z_i is the output produced by the tool after the learning phase corresponding to x_i and $g(x_i)$ is the desired value.

Fig. 17 gives the results of ten runs (using WS), none of which were discarded, ordered according to their testing error. 20 000 iterations were performed with five rules. The number of iterations has to be higher than that used in the two methods described due to the genetic nature of the algorithm and the total absence of a precalculation phase prior to application of the algorithm.

The number of rules was determined as follows. In [44], where the best performance was obtained in terms of both learning capacity and compactness of the fuzzy knowledge base, only four rules were used for a total of 16 parameters to be optimized. As FuGeNeSys only requires three parameters per rule the number of parameters to be optimized was set to 15. Therefore, five rules are chosen.

As can be seen, in all cases performance is better than that reported in [3] ($J2 = 3.19\%$), while two out of the ten runs give better performance than the second method (in [44] we have $J2 = 0.987\%$). Also, in this case, FuGeNeSys has an excellent learning capacity (function approximation) and is capable of extracting extremely compact fuzzy inferences. In this case, the number of rules extracted possesses a number of parameters even lower than those reported in [44].

3) Fuzzy Control: Typically, the fuzzy control applications need a number of fuzzy rules, which becomes very large as soon as the number of inputs increases. However, the most recent works are gaining in compactness.

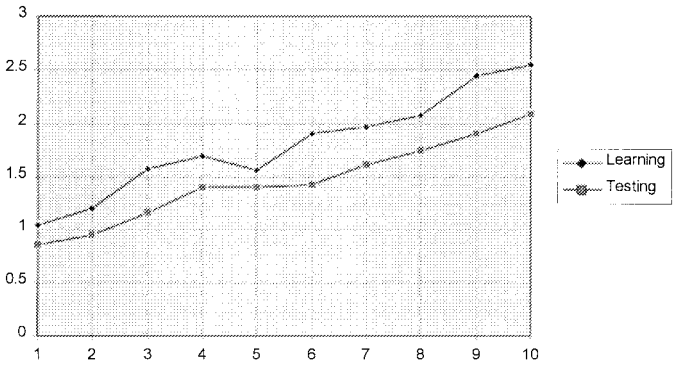

Fig. 17. Ten runs arranged according to testing error. The ordinate value is the index defined by Ralescu [45].

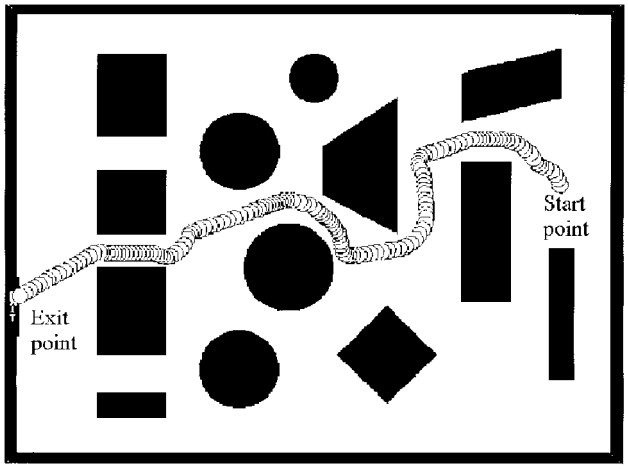

Fig. 18. An example of the robot trajectory.

Homaifar and McCormick [48] needed 25 rules to control the two inputs one input fuzzy cart-centering controller and 35 rules to control the two inputs one input fuzzy truck-backing controller.

An approach that produces more compact fuzzy databases is described in [49]. Lin and Lin needed ten fuzzy rules in the action network and six in the critic network to control the cart-pole system. Further, they needed 14 rules in the action network and nine in the critic network to control the four input—one output ball and beam system.

Actually, it is impossible to compare directly FuGeNeSys with other GA approachs for fuzzy control.

In fact, FuGeNeSys is a tool that is able to extract fuzzy knowledge only from samples of the function or of the classification problem to be approximated.

In a future version of FuGeNeSys the author thinks he will modify the fitness function, in a similar way as done in the related works, to obtain a system that works in control fields.

However, he already used FuGeNeSys in a motion planning problem [2]. He considered the problem of navigation of a mobile robot in unknown two-dimensional environments with obstacles.

The robot is memoryless, i.e., it does not build an internal map of the environment. Obstacles are convexes and comparable with robot size.

The five inputs of the fuzzy controller the author developed are the output of three proximity sensors, the rotation angle required to get the right output direction and the previous fuzzy controller direction output.

The fuzzy controller outputs are two: the direction and the speed.

The author manually generated several samples of the desired controlling sequence in some typical cases. The system that he obtained is made of only five rules.

In Fig. 18 the author shows an example of the result of the robot trajectory he obtained.

VI. Conclusions

The FuGeNeSys program presented in the paper generates simple, effective fuzzy models of complex systems from knowledge of the input–output data. The learning technique used is essentially based on GA's. To enhance the learning speed, a hill-climbing genetic operator based on neural techniques has been used. FuGeNeSys is also capable of correctly selecting significant features. It has been demonstrated that the results achieved represent a significant advance in the use of mixed techniques in soft computing.

Acknowledgment

The author would like to thank Prof. G. V. Russo and the reviewers for their helpful suggestions in improving the quality of the final manuscript.

References

[1] L. A. Zadeh, "Fuzzy logic, neural networks, and soft computing," *Commun. ACM*, vol. 37, pp. 77–84, Mar. 1994.
[2] M. Russo, "Metodi hardware e software per logiche di tipo non tradizionale," Ph.D. dissertation, Univ. Catania, Catania, Italy, 1996.
[3] N. A. Santagati, E. Lo Pinto, and M. Russo, "Fuzzy logic can help medicinal chemistry," in *Proc. Artificial Neural Networks Eng. Conf.*, St. Louis, MO, June 1995, pp. 297–302.
[4] F. Beritelli, S. Casale, and M. Russo, "A voice/unvoiced speech discrimination technique based on fuzzy logic," in *Proc. 4th Eur. Conf. Speech Commun. Technol.*, Madrid, Spain, Sept. 1995, vol. 1, pp. 389–392.
[5] ——, "Encoding of PARCOR coefficients using fuzzy rules," in *8th IEEE Mediterranean Electrotech. Conf.*, Bari, Italy, May 1996, pp. 1659–1662.
[6] ——, "Robust phase reversal tone detection using soft computing," in *IEEE Proc. 3rd Int. Symp. Uncertainity Modeling Anal. Annu. Conf. North Amer. Fuzzy Inform. Processing Soc.*, College Park, MD, Sept. 1995, pp. 589–594.
[7] F. Beritelli, S. Casale, and M. Russo, "Multilevel speech classification based on fuzzy logic," in *Proc. IEEE Workshop Speech Coding Telecommun.*, Annapolis, MD, Sept. 1995, pp. 97–98.
[8] ——, "Multimode speech coding decision based on fuzzy logic," in *Proc. Int. Conf. Signal Image Processing*, Las Vegas, NV, Nov. 1995, pp. 72–75.
[9] V. Catania, M. Malgeri, and M. Russo, "Applying fuzzy logic to codesign partitioning," *IEEE Micro*, vol. 17, pp. 62–70, May 1997.
[10] V. Catania and M. Russo, "Analog gates for a VLSI fuzzy processor," in *8th Int. Conf. VLSI Design*, New Delhi, India, Jan. 1995, pp. 299–304.
[11] A. Gabrielli, E. Gandolfi, M. Masetti, and M. Russo, "Design of a VLSI very high speed reconfigurable digital fuzzy processor," in *Proc. ACM Symp. Appl. Comput.*, Nashville, TN, Feb. 1995, pp. 477–481, invited talk.
[12] G. V. Russo, C. Petta, D. Lo Presti, N. Randazzo, and M. Russo, "Silicon drift detectors readout and ON-LINE data reduction using a fast VLSI dedicated fuzzy processor," in *Proc. 2nd Annu. Joint Conf. Inform. Sci.*, Wrightsville Beach, NC, Sept. 1995, pp. 64–67.
[13] ——, "Silicon drift detector readout and ON-LINE data reduction using a fast VLSI fuzzy processor," *Inform. Sci.*, vol. 95, pp. 233–260, Dec. 1996.
[14] V. Catania, S. Cavalieri, and M. Russo, "Tuning Hopfield neural network by a fuzzy approach," in *Proc. IEEE Int. Conf. Neural Networks*, Washington, DC, June 1996, vol. II, pp. 1067–1072.
[15] S. Cavalieri and M. Russo, "Improving Hopfield neural network performance by fuzzy logic—Based coefficient tuning," *Neurocomput.*, vol. 18, pp. 107–126, 1998.
[16] M. Russo, N. A. Santagati, and E. Lo Pinto, "Medicinal chemistry and fuzzy logic," *Inform. Sci.*, vol. 105/1–4, pp. 299–314, May 1998.
[17] M. Russo, "Comments on 'A new approach to fuzzy-neural system modeling'," *IEEE Trans. Fuzzy Syst.*, vol. 4, pp. 209–210, May 1996.
[18] C. Darwin, *On the Origin of Species by Means of Natural Selection*. London, U.K.: Murray, 1859.
[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[20] J. H. Holland, "Adaptive plans optimal for payoff-only environments," in *Proc. 2nd Int. Conf. Syst. Sci.*, HI, Feb. 1969, pp. 917–920.
[21] ——, *Adaption in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.
[22] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Networks*, vol. 5, pp. 3–14, Jan. 1995.
[23] E. Rietman, *Genesis Redux—Experiments Creating Artificial Life*, 1st ed. New York: McGraw-Hill, 1994.
[24] A. Tettamanzi, "Algoritmi evolutivi per l'ottimizzazione," Ph.D. dissertation, Univ. Milan, Milan, Italy, 1995.
[25] Z. Michalewics, *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1994.
[26] S. Uckun, S. Bagchi, and K. Kawamura, "Managing genetic search in job shop scheduling," *IEEE Expert*, vol. 8, pp. 15–24, Oct. 1993.
[27] J. R. Koza, "Evolution and co-evolution of computer programs to control independently-acting agents," in *From Animals to Animats*, J. A. Meyer and S. W. Wilson, Eds. Cambridge: MA: MIT Press, 1991.
[28] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of NAtural Selection*. Cambridge, MA: MIT Press, 1993.
[29] T. Blinkle, "Theory of evolutionary algorithms and application to system synethesis," Ph.D. dissertation, Swiss Fed. Inst. Technol., Zurich, Switzerland, 1996.
[30] J. E. Baker, "An analysis of the effects of selection in genetic algorithms," Ph.D. dissertation, Graduate School Vanderbilt, Univ. Nashville, Nashville, TN, 1989.
[31] D. Back and F. Hoffmeister, "Extended selection methods for genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms*, San Matteo, CA, 1991.
[32] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Trans. Neural Networks*, vol. 5, pp. 96–101, Jan. 1994.
[33] S. H. Park, Y. H. Kim, Y. K. Choi, H. C. Cho, and H. T. Jeon, "Self-organization of fuzzy rule base using genetic algorithms," in *Proc. 5th IFSA World Congress*, Seoul, Korea, July 1993, pp. 881–886.
[34] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 46–53, Feb. 1993.
[35] M. A. Lee and H. Takagi, "Integrating design stages of fuzzy systems using genetic algorithms," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, Mar. 1993, pp. 612–617.
[36] ——, "Embedding *a priori* knowledge into an integrated fuzzy system design method based on genetic algorithms," in *Proc. 5th IFSA World Congress*, Seoul, Korea, July 1993, pp. 1293–1296.
[37] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, pp. 116–132, 1985.
[38] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if–then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 260–270, Aug. 1995.
[39] H. Ishibuchi, K. Nozaky, and H. Tanaka, "Distributed representation of fuzzy rules and its qapplication to pattern classification," *Fuzzy Sets Syst.*, vol. 52, pp. 21–32, 1992.
[40] L. A. Zadeh, "Fuzzy sets," *Inform. Contr.*, vol. 8, pp. 338–353, 1965.
[41] J. L. Castro, "Fuzzy logic controllers are universal approximators," Tech. Rep. DECSAI-93101,6/1993, Dept. Comput. Sci. Artificial Intell., Univ. Grenada, Spain, 1993.
[42] R. Jager, "Fuzzy logic in control," Ph.D. dissertation, Univ. Delft, Delft, The Netherlands, 1995.
[43] M. Figueiredo, F. Gomides, A. Rocha, and R. Yager, "Comparison of Yager's level set method for fuzzy logic control with Mamdani and Larsen methods," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 156–159, May 1993.

[44] Y. Lin and G. A. Cunningham, III, "A new approach to fuzzy-neural system modeling," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 190–198, May 1995.
[45] H. Narazaki and A. L. Ralescu, "An improved synthesis method for multilayered neural networks using qualitative knowledge," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 125–137, May 1993.
[46] R. A. Fisher, "The use of multiple measurements in toxonomic problems," *Ann. Eugen.*, vol. 7, pp. 179–188, 1936.
[47] M. Grabish and F. Dispot, "A comparison of some methods of fuzzy classification on real data," in *Proc. 2nd Int. Conf. Fuzzy Logic Neural Networks*, Iizuka, Japan, July 1992, pp. 659–662.
[48] A. Homaifar and E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 129–139, May 1995.
[49] C. J. Lin and C. T. Lin, "Reinforcement learning for an ART-based fuzzy adaptive learning control network," *IEEE Trans. Neural Networks*, vol. 7, pp. 709–731, May 1996.

Marco Russo received the "laurea" degree in electronic engineering and the Ph.D. degree in electronics, computer science and telecommunications engineering both from the University of Catania, Italy, in 1992 and 1996, respectively.

He is currently an Assistant Researcher at the University of Catania. His research areas are in fuzzy logic, neural networks, genetic algorithms, hybrid systems, dedicated hardware, signal processing, high-energy physics, and artificial intelligence applications. He is on the editorial board of the *International Journal of Knowledge–Based Intelligent Engineering Systems*.