NORTH-HOLLAND

# A Three-Stage Evolutionary Process for Learning Descriptive and Approximate Fuzzy-Logic-Controller Knowledge Bases From Examples*

**O. Cordón and F. Herrera**
*Department of Computer Science and Artificial Intelligence, University of Granada, Spain*

## ABSTRACT

*Nowadays fuzzy logic controllers have been successfully applied to a wide range of engineering control processes. Several tasks have to be performed in order to design an intelligent control system of this kind for a concrete application. One of the most important and difficult ones is the extraction of the expert known knowledge of the controlled system. The aim of this paper is to present an evolutionary process based on genetic algorithms and evolution strategies for learning the fuzzy-logic-controller knowledge base from examples in three different stages. The process allows us to generate two different kinds of knowledge bases, descriptive and approximate ones, depending on the scope of the fuzzy sets giving meaning to the fuzzy-control-rule linguistic terms, taking preliminary linguistic-variable fuzzy partitions as a base. The performance of the method proposed is shown by measuring the accuracy of the fuzzy logic controllers designed in the fuzzy modeling of three three-dimensional surfaces presenting different*

*Address correspondence to Francisco Herrera, Department of Computer Science and Artificial Intelligence, E.T.S. de Ingeniería Informática, University of Granada, 18071-Granada, Spain. E-mail:* herrera@decsai.ugr.es.

*This research has been supported by CICYT TIC96-0778.

*characteristics, and by comparing them with others generated by means of three methods based on Wang and Mendel's knowledge-base generation process.* © *1997 Elsevier Science Inc.*

KEYWORDS: ***fuzzy logic controllers, fuzzy-logic-controller knowledge base, genetic algorithms, evolution strategies, niching, inductive learning***

## 1. INTRODUCTION

Fuzzy logic controllers (FLCs), initiated by Mamdani and Assilian in the work [35], are now considered as one of the most important applications of fuzzy set theory. FLCs are knowledge-based controllers that make use of the known knowledge of the process, expressed in the form of fuzzy linguistic control rules collected in the FLC knowledge base (KB), to control it (for further information about FLCs see [16, 31]). The advantage of this approach with respect to classical control theory is that it does not need to express the relationships existing in the system by means of a mathematical model, which constitutes a very difficult task in many real situations presenting nonlinear characteristics or complex dynamics.

Several tasks have to be performed in order to design an intelligent control system of this kind for a concrete application. One of the most important and difficult ones is the extraction of the expert known knowledge of the controlled system. The difficulty reported by human process operators in expressing their knowledge in the form of control rules has made researchers develop automatic techniques for performing this task. In the last few years, many different approaches have been presented, taking genetic algorithms (GAs) as a base, and obtaining the so-called genetic fuzzy systems (GFSs).

In this paper we present an evolutionary process based on GAs and evolution strategies (ESs) for learning the KB from examples. The process presents two variants allowing us to generate a KB either with a descriptive nature (the classical working mode characterized by the existence of a fixed relationship between the linguistic labels and the fuzzy sets giving meaning to them, i.e., the meaning of the labels is the same for all the fuzzy control rules contained in the KB) or with an approximate nature (a different approach in which this relationship does not exist and the linguistic variables take different fuzzy sets as values depending on the concrete fuzzy control rule). The performance of both variants is compared with that of other processes based on the Wang-Mendel fuzzy rule generation method by using them to develop a fuzzy modeling of three three-dimensional control surfaces derived from mathematical functions.

In the next section we briefly present some preliminary concepts: FLCs, evolutionary algorithms (with a short introduction to GAs and ESs), and genetic fuzzy systems. Some considerations about the KB learning process are discussed in Section 3. The proposed method is introduced by describing the first stage composing it in Sections 4, 5, and 6, and the remaining two in Sections 7 and 8, respectively. In Section 9, we show the experiments developed and the different results obtained. Finally, in Section 10 some concluding remarks are made.

## 2. PRELIMINARIES: FUZZY LOGIC CONTROLLERS, EVOLUTIONARY ALGORITHMS, AND GENETIC FUZZY SYSTEMS

### 2.1. Fuzzy Logic Controllers

An FLC is composed of a *knowledge base*, which comprises the information given by the process operator in the form of linguistic control rules; a *fuzzification interface*, which has the effect of transforming crisp data into fuzzy sets; an *inference system*, which uses these together with the knowledge base to make inference by means of a reasoning method; and a *defuzzification interface*, which translates the fuzzy control action thus obtained into a real control action using a defuzzification method.

The *knowledge base* is the FLC component comprising the expert knowledge known about the controlled system. Thus, it is the only component of the FLC depending on the concrete application, and it makes the accuracy of the FLC depend directly on its composition. It has two components, a *data base* (DB), containing the definitions of the fuzzy-control-rule linguistic labels and the universe-of-discourse scaling factors, and a *rule base* (RB), constituted by the collection of fuzzy control rules representing the expert knowledge. There are different kinds of fuzzy control rules proposed in the specialized literature regarding the expression of the consequent. Below we suppose a KB constituted by $n$ Mamdani-type rules with the form

$$\text{IF } X_1 \text{ is } A_1 \text{ and} \ldots \text{and } X_n \text{ is } A_n \text{ THEN } Y \text{ is } B,$$

with $X_i$ and $Y$ being linguistic system variables, and $A_i$ and $B$ the linguistic labels associated with fuzzy sets which specify their meaning.

The *fuzzification interface* defines a mapping from an observed input space to fuzzy sets in certain input universes of discourse, thereby obtaining the membership function associated to each one of the crisp system inputs.

The *inference system* is based on the application of the generalized modus ponens, an extension of the classical-logic modus ponens. It is done by means of the compositional rule of inference (CRI) given by the following expression:

$$\mu_{B'}(y) = \operatorname*{Sup}_{x \in X} \left\{ T'\left( \mu_{A'}(x), I\left( \mu_{A_i}(x), \mu_{B_i}(y) \right) \right) \right\},$$

with $\mu_{A_i}(x) = T(\mu_{A_{i1}}(x), \ldots, \mu_{A_{in}}(x))$, $T$, $T'$ being connectives, and $I$ being an implication operator.

As the input $x$ corresponding to the state variables of the controlled system is crisp, $x = x_0$, the fuzzy set $A'$ is a singleton, that is, $\mu_{A'}(x) = 1$ if $x = x_0$ and $\mu_{A'}(x) = 0$ if $x \neq x_0$. Thus the CRI is reduced to

$$\mu_{B'}(y) = I\left( \mu_{A_i}(x_0), \mu_{B_i}(y) \right).$$

Since from each rule $R_i$ is obtained a fuzzy set $B_i'$ from the inference process, the *defuzzification interface* uses an aggregation operator $G$ which composes them and applies a defuzzification method $D$ to translate the fuzzy sets obtained in this way into values corresponding to the control vsariables of the system. Hence, denoting by $S$ to FLC, by $x_0$ the input value, and by $y_0$ the crisp value obtained from the defuzzification, we have

$$\mu_{B'}(y) = G\left\{ \mu_{B_1'}(y), \mu_{B_2'}(y), \ldots, \mu_{B_n'}(y) \right\},$$
$$y_0 = S(x_0) = D(\mu_{B'}(y)).$$

At present, the commonly used defuzzification methods may be described as the *max criterion*, the *mean of maximum* (MOM), and the *center of area* (COA) [12, 16, 31].

Several factors with a significant influence have to be analyzed in order to design an FLC for a concrete process. Concretely, there are two main decisions that have to be made: to derive a KB for the system and to decide the reasoning method to use. The first one depends directly on the concrete application. The design tasks that have to be developed in order to decide the FLC reasoning method are the selection of the fuzzy operators $I$, $T$, and $G$, and the defuzzification operator $D$. The problem of selecting them has been analyzed by the authors in prior work [9]. For more information about FLCs see [12, 16, 31].

## 2.2. Evolutionary Algorithms

*Evolutionary computation* (EC) uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem-solving systems. There are a variety of evolutionary computational modes that have been proposed and studied, which are referred to as *evolutionary algorithms* (EAs). There have been three well-defined EAs which have served as the basis for much of the activity in the field: GAs [19, 25], ESs [1, 40], and *evolutionary programming* (EP) [17, 18].

An EA maintains a population of trial solutions, imposes random changes to these solutions, and incorporates selection to determine which ones are going to be maintained in future generations and which will be removed from the pool of the trials. There are however important differences between EAs. GAs emphasize models of genetic operators as observed in nature, such as crossover (recombination) and point mutation, and apply these to abstracted chromosomes. ESs and EP emphasize mutational transformations that maintain the behavioral linkage between each parent and its offspring.

In the following we briefly review the GAs and the ESs, both of which will be used in this paper.

## 2.3. Genetic Algorithms

GAs [19] are theoretically and empirically proven algorithms that provide a robust search in complex spaces, thereby offering a valid approach to problems requiring efficient and effective searches.

Any GA starts with a population of randomly generated solutions (chromosomes) and advances toward better solutions by applying genetic operators, modeled on the genetic processes occurring in nature. In these algorithms we maintain a population of solutions for a given problem; this population undergoes evolution in the form of natural selection. In each generation, relatively good solutions reproduce to give offspring that replace the relatively bad solutions, which die. An evaluation or fitness function plays the role of the environment to distinguish between good and bad solutions. The process of going from the current population to the next population constitutes one generation in the execution of a GA.

## 2.4. Evolution Strategies

ESs [1, 40] were initially developed by Rechenberg and Schwefel in 1964 with a strong focus on building systems capable of solving difficult real-valued parameter optimization problems. The natural representation was a vector of real-valued genes that were manipulated primarily by mutation operators designed to perturb the real-valued parameters in useful ways.

The first ES algorithm, the so-called $(1 + 1)$-*ES*, was based on only two individuals per generation, one parent and one descendent. This algorithm is based on evolving the parent string by applying a mutation operator to each one of its components. The mutation strength is determined by a value $\sigma$, the standard deviation of a normally distributed random variable. This parameter is associated to the parent and it is evolved in each process step as well. If the evolution has been performed successfully, then the descendent replaces the parent in the next generation. The individual adaptation is measured by using a fitness function. The process is iterated until a determined finishing condition is satisfied.

The mutation operator **mut** has two components. The first one, $\mathbf{mu}_\sigma$, evolves the value of the standard deviation $\sigma$ using Rechenberg's $\frac{1}{5}$-success rule:

$$\sigma' = \mathbf{mu}_\sigma(\sigma) = \begin{cases} \sigma/\sqrt[n]{c} & \text{if} \quad p > \frac{1}{5}, \\ \sigma\sqrt[n]{c} & \text{if} \quad p < \frac{1}{5}, \\ \sigma & \text{if} \quad p = \frac{1}{5}, \end{cases}$$

where $p$ is the relative frequency of successful mutations and $c$ is a constant determining the updating amount of $\sigma$. The second one, $\mathbf{mu}_x$, mutates each component of the real coded string by adding normally distributed variations with standard deviation $\sigma'$ to it:

$$x' = \mathbf{mu}_x(x) = (x_1 + z_1, \dots, x_n + z_n)$$

where $z_i \sim N_i(0, \sigma'^2)$.

## 2.5. Genetic Fuzzy Systems

The KB derivation is a task, directly depending on the controlled system, that has to be performed in order to design an FLC, and it has significative importance in the design process [12, 16, 31]. The most used method for performing this task is based directly on extracting the expert experience from human process operators. The problem arises when these are not able to express their knowledge in terms of fuzzy control rules. In order to avoid this drawback, automatic learning methods for designing FLCs by automatically deriving an appropriate KB are needed.

GAs have been demonstrated to be a powerful tool for automating the definition of the KB, since adaptive control, learning, and self-organization may be considered in a lot of cases as optimization or search processes. Their advantages have extended the use of GAs in the development of a wide range of approaches for designing FLCs over the last few years. In particular, the application to the design, learning, and tuning of KBs has produced quite promising results. These approaches can be given the general name of *genetic fuzzy systems* (GFSs) [8]. Figure 1 shows this idea.

EAs (specially GAs in the great majority of the cases) are applied to modify/learn the DB and/or the RB. Therefore, they may act on one or both KB components introduced in Section 2.1. It is possible to distinguish

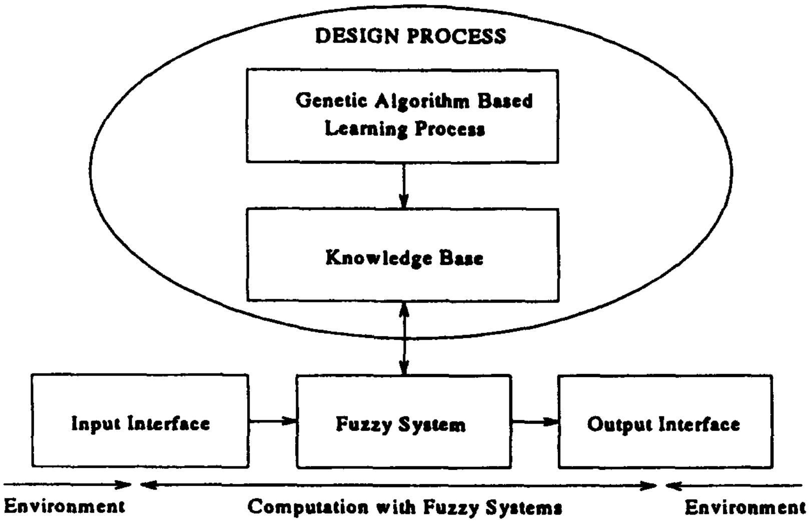

**Figure 1.** Genetic fuzzy systems.

three different groups of *genetic FLC design processes* according to the KB components included in the learning process. These are the following [8]:

1. Genetic definition of the fuzzy logic controller data base [3, 20, 26, 41].
2. Genetic derivation of the fuzzy logic controller rule base [4, 24, 27, 30, 42].
3. Genetic learning of the fuzzy logic controller knowledge base [6, 9, 14, 21, 22, 28, 32–34, 37, 39].

For a wider description of approaches belonging to each one of them see [8], and for an extensive bibliography see [13] (Section 3.13). Different approaches may be found in [23].

Carse et al. [7] divide the third family into two different subgroups depending on the simultaneity in the learning of the two KB components. Namely, they differentiate between learning the DB and the RB in stages (for example, the approaches presented in [9, 14, 21, 22, 28]) and learning them simultaneously (as in the remaining approaches cited).

In this paper we present a GFS belonging to this third family and to the first subgroup. In this way, the process allows us to automatically generate a complete KB (when a training set formed by numerical input-output problem variable pairs is available) in three stages.

## 3. ON THE GENETIC LEARNING PROCESS

In this section we are going to introduce the basis followed by the proposed GFS by analyzing the following aspects in depth.

### 3.1. Type of Fuzzy Models and Structure of the Input-Output Data Sets

We shall focus on Mamdani's model for multiple-input single-output (MISO) systems, where the knowledge base of a fuzzy controller consists of a collection of fuzzy rules (with the logical connective ALSO between them) describing the control actions in the form

$$R_i : \text{IF } x_1 \text{ is } A_{i1} \text{ and} \ldots \text{and } x_n \text{ is } A_{in} \text{ THEN } y \text{ is } B,$$

where $x_1, \ldots, x_n$ and $y$ are the process state variables and the control variable respectively; and $A_{i1}, \ldots, A_{in}$, $B$ are fuzzy sets in the universes of discourse $U_1, \ldots, U_n$, $V$. These fuzzy sets are characterized by their membership functions

$$A_{ij}(B) : U_j(V) \to [0,1], \qquad j = 1, \ldots, n.$$

We consider every fuzzy set associated with a normalized triangular membership function. A computational way to characterize it is by using a parametric representation achieved by means of the 3-tuples $(a_{ij}, b_{ij}, c_{ij})$, $(a_i, b_i, c_i)$, $j = 1, \ldots, n$.

The classical Mamdani model is a linguistic model based on collections of IF-THEN rules with fuzzy quantities associated with linguistic labels, and the fuzzy model is essentially a qualitative expression of the system. A KB in which the fuzzy sets giving meaning (semantic) to the linguistic labels are uniformly defined for all rules included in the RB constitutes a *descriptive* approach, since the linguistic labels take the same meaning for all the fuzzy control rules contained in the RB.

One can consider a KB for which each fuzzy control rule presents its own meaning, i.e., the linguistic variables involved in the rules do not take as value any linguistic label from a global term set. In this case, the approach is called *approximate* [8] and the linguistic variables become fuzzy variables. In this second approach we say that the rules present *free semantics*. The difference between the two approaches is shown in Figure 2.

We will treat both approaches in this paper. For the generation process we consider classical Mamdani-type fuzzy control rules, all of which present the same meaning for the linguistic terms involved, and rules with a free semantics, without any associated linguistic syntaxis, but based on an

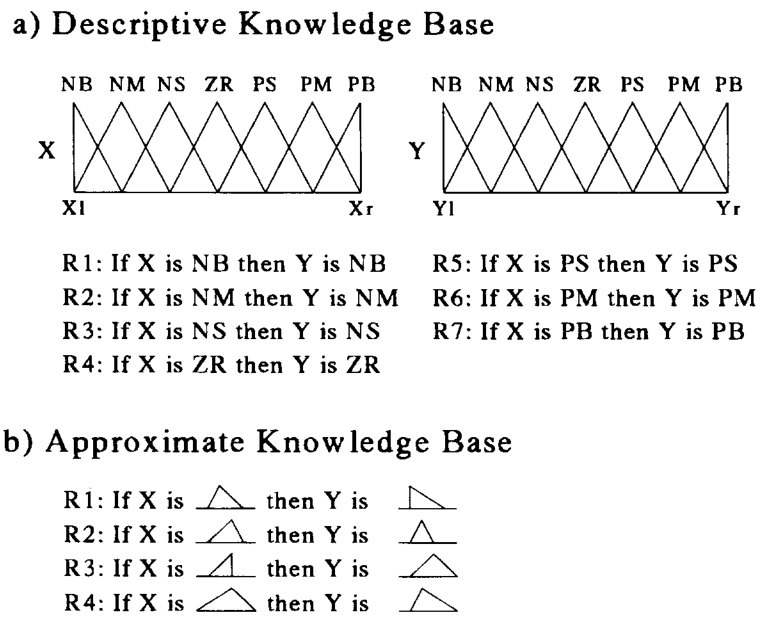
a) Descriptive Knowledge Base
NB NM NS ZR PS PM PB
X
Xl
Xr
NB NM NS ZR PS PM PB
Y
Yl
Yr
R1: If X is NB then Y is NB
R2: If X is NM then Y is NM
R3: If X is NS then Y is NS
R4: If X is ZR then Y is ZR
R5: If X is PS then Y is PS
R6: If X is PM then Y is PM
R7: If X is PB then Y is PB
b) Approximate Knowledge Base
R1: If X is then Y is
R2: If X is then Y is
R3: If X is then Y is
R4: If X is then Y is

**Figure 2.** Examples of descriptive and approximate knowledge bases.

initial domain fuzzy partition. Therefore, the GFS designer may choose the desired fuzzy model before running the whole process. This choice will affect the composition of the first and third process stages: the evolutionary fuzzy rule generation and genetic tuning respectively. If he decides to work with the approximate approach, then the first stage will include an ES for locally tuning the fuzzy control rule membership functions and an approximate tuning process will be used at the third stage. On the other hand, when the usual descriptive fuzzy model is chosen, no local tuning is performed in the first stage and a descriptive genetic tuning process is applied in the last stage.

We also consider an inductive approach for designing GFSs. Thus, there is a need to have a training data set, $E_p$, composed of $p$ numerical input-output (state-control) problem variable pairs, experimentally recorded, in order to perform the learning. These examples present the following structure:

$$e_l = (ex_1^l, \ldots, ex_n^l, ey^l), \qquad l = 1, \ldots, p.$$

### 3.2. Properties Required for the Generated Knowledge Base

Several important static properties have to be verified by the KB in order to obtain an FLC presenting good behavior. Below we are going to discuss the consideration two of them: *completeness* and *consistency* [16, 31].

A. COMPLETENESS OF A KNOWLEDGE BASE It is clear that an FLC should always be able to infer a proper control action for every state of the controlled system. This property is called *completeness*. The completeness of a KB relates to its two components, the DB and the RB, in the following way [31]:

- The *DB strategy* is concerned with the supports on which primary fuzzy sets are defined. The union of these supports should cover the related universe of discourse in relation with some level set $\sigma \in [0, 1]$. This property of an FLC is called $\sigma$-completeness. In general, when the DB is defined by means of a uniform fuzzy partition of the input spaces, we choose the level $\sigma$ at the crossover point, as shown in Figure 3, according to our belief about the positive sense of the fuzzy control rules forming the RB. In this way, with $s \in S$ being a system state and $S_i$ being the fuzzy sets giving meaning to the linguistic labels for the input space, the property of $\sigma$-completeness can be formally defined in the following way:

$$\forall s \in S, \qquad \bigcup S_i(s) > \sigma .$$

- The *RB strategy* refers to the fuzzy control rules themselves. The property of completeness is incorporated into the RB through design experience and engineering knowledge. An additional rule is added whenever a fuzzy condition is not included in the RB or whenever the degree of partial matching between some inputs and the predefined fuzzy conditions is lower than some desired level. An RB will be $\sigma$-complete when the following condition is verified [16]:

$$\forall s \in S, \qquad \text{hgt}(\text{OUT}(s)) > \sigma$$

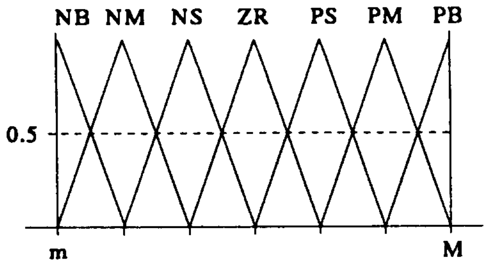

**Figure 3.** Graphical representation of a possible fuzzy partition.

where hgt(·) represents the height of a fuzzy set and OUT represents the fuzzy control action obtained from the input $s$ by means of the FLC inference process using the concrete KB.

As was mentioned in [16], it must be noticed that many usual practical applications controlled by FLCs do not present a complete KB. This is due to the fact that there are certain regions in the input domain that are not of interest for controlling the process. An example of this kind of systems is the well-known problem of the inverted pendulum.

Thus, we need a training data set that adequately represents the process control surface when performing inductive learning of the KB, because this set will determine the completeness of the generated KB, as we shall see later. Both these conditions will be incorporated into the learning process by means of the following expressions. The generic value $\sigma$ is called $\tau$ in the latter:

$$C_R(e_l) = \bigcup_{i=1,\dots,T} R_i(e_l) \geq \tau, \qquad l = 1,\dots,p,$$

$$R_i(e_l) = *\big(A_i(ex^l), B_i(ey^l)\big),$$

$$A_i(ex^l) = *\big(A_{i1}(ex_1^l),\dots, A_{in}(ex_n^l)\big),$$

where $*$ is a $t$-norm, and $R_i(e_l)$ is the *compatibility degree* between the rule $R_i$ and the example $e_l$.

Given a KB composed of $T$ fuzzy control rules $R_i$, the *covering value* of an example $e_l \in E_p$ is defined as

$$\mathrm{CV}_R(e_l) = \sum_{i=1}^{T} R_i(e_l),$$

and we require the following condition:

$$\mathrm{CV}_R(e_l) \geq \epsilon, \qquad l = 1,\dots,p$$

A good KB must satisfy both the conditions presented above, to verify the *completeness property* and to have an adequate final *covering value.*

B. CONSISTENCY OF A KNOWLEDGE BASE A generic set of IF-THEN rules is *consistent* if it does not contain contractions. This concept is clear in other knowledge-based systems, but it is difficult to translate it into the field of fuzzy logic control.

When the RB is generated via human operator experience, the rules obtained may be subject to different performance criteria. For example, high accuracy and low fuel consumption in a process are potentially contradictory. This may lead to an inconsistent RB in which the resulting fuzzy control action obtained from an input state may be multimodal, that is, two or more rules present the same antecedent and different consequents. Anyway, this multimodality disappears in the defuzzification, although in many cases the control obtained may not be effective on either of the two aforementioned criteria. Thus, a deeper study of the RB is required for eliminating or replacing the main inconsistent rules.

In [16] it is questioned whether an RB is inconsistent or not even when presenting rules with the same antecedent and different consequent. There is a need to relax the consistency requirement for considering it in fuzzy KBs. We do this by means of the concepts of *positive* and *negative examples* [21]. An example is considered positive for a fuzzy control rule when it matches with its antecedent and consequent, and is considered negative when it matches with its antecedent and not with its consequent. Hence, the fewer negative examples the fuzzy control rules have, the more consistent the KB can be considered. The existence of some negative examples for a rule is accepted when it presents a sufficiently large number of positive examples. We shall consider this property in the learning process. For its formulation refer to the next section.

## 3.3. Some Considerations on the Learning Approach

When the problem of designing a GFS is considered, there is a need to encode the possible solutions into a genetic representation for translating the FLC parameter space into a certain space in which the GA can operate. The FLC parameters considered in the learning process will condition the dimension and properties of this space, making problem solving faster or slower and even tractable in a lower or higher degree. As was noted in [38], research indicates that the string length and the problem complexity play a critical role in these factors.

This drawback may appear when designing GFSs belonging to the third family presented in the previous section, i.e., when the generation of the whole KB is considered in the genetic process. In this case, a large number of KB components must be included in the genetic representation, which becomes larger. This fact will be more pronounced if an approximate fuzzy model is considered. The use of different membership-function definitions for each rule makes the number of KB parameters increase, which makes the search space more complex, making the problem computationally intractable.

Hence, the learning process must be carefully designed to avoid these problems. We will perform a global simplification on the whole learning process by dividing it into three different stages, making the search space simpler than in a single-stage design process. The proposed method will consist of the following three steps, maintaining the generic structure used in [9, 14, 21]:

1. An *evolutionary generation process* for generating fuzzy control rules, with two components: a fuzzy-rule generating method based on an inductive algorithm with an optional ES that locally tunes the rules, and an iterative covering method for the system behavior example set. As we are going to show in the following section, the use of the ES will determine the nature of the fuzzy model: approximate if it is applied, or descriptive if not. This process allows us to obtain a set of rules covering the training set in an adequate form.
2. A *genetic simplification process* for selecting rules, based on a binary-coded genetic algorithm and a measure of the FLC performance in the control of the system being identified. This will avoid the over-learning that the previous component may cause due to the existence of redundant rules in obtaining the final RB.
3. A *genetic tuning process*, based on a real coded GA and a measure of the FLC performance. It will give the final KB as output by tuning the membership functions for each fuzzy control rule or for the complete RB, according as the fuzzy model is approximate or descriptive.

Figure 4 presents a block diagram of the proposed learning process, showing the section of the paper in which each component is described.

## 4. THE EVOLUTIONARY GENERATION PROCESS

As commented earlier, the first stage consists of two processes, a *generating method* for obtaining desirable fuzzy rules from examples, and a *covering method* for the set of examples.

1. The *fuzzy-rule-generating method* finds the best rule in every run over the set of examples according to the features included in a fitness function. It may include an optional ES that locally tunes the fuzzy control rules obtained. Its application will determine if the final KB generated will be approximate or descriptive.
2. The *covering method* allows us to obtain a set of fuzzy rules covering the set of examples. This method is developed as an iterative process. In each iteration, it runs the generating method choosing the best fuzzy control rule, considers the relative covering value that this rule yields over the example set, and removes the examples with a covering value greater than a value $\epsilon$ provided by the controller designer.

The following two sections present both methods in depth.

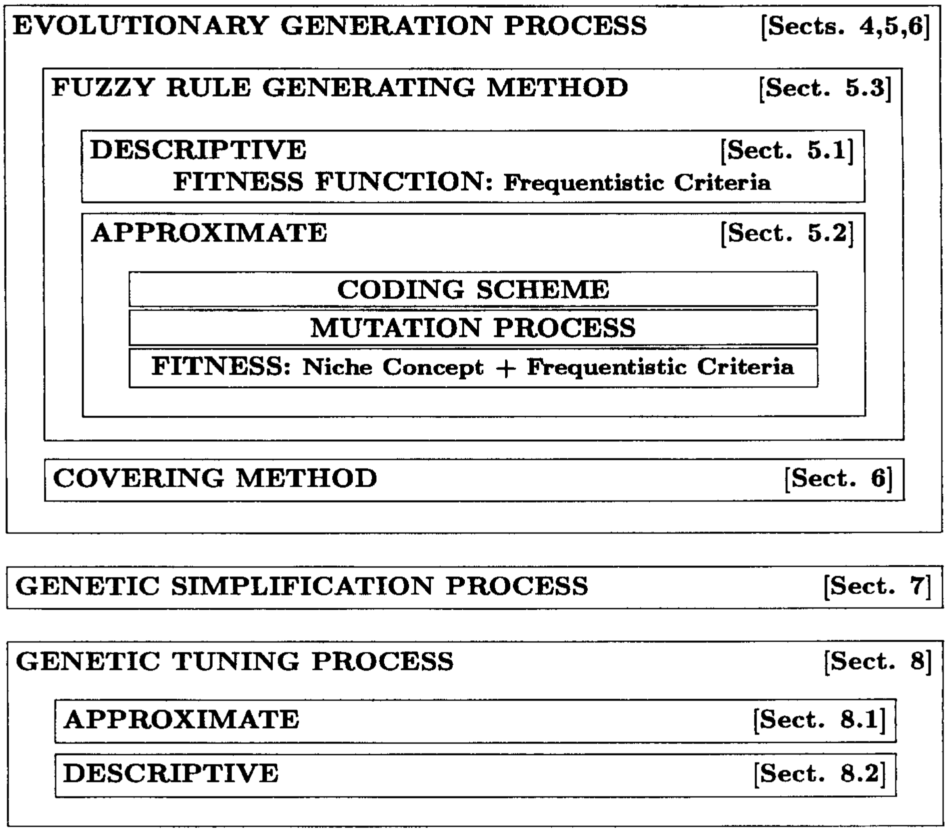
EVOLUTIONARY GENERATION PROCESS [Sects. 4,5,6]
FUZZY RULE GENERATING METHOD [Sect. 5.3]
DESCRIPTIVE [Sect. 5.1]
FITNESS FUNCTION: Frequentistic Criteria
APPROXIMATE [Sect. 5.2]
CODING SCHEME
MUTATION PROCESS
FITNESS: Niche Concept + Frequentistic Criteria
COVERING METHOD [Sect. 6]
GENETIC SIMPLIFICATION PROCESS [Sect. 7]
GENETIC TUNING PROCESS [Sect. 8]
APPROXIMATE [Sect. 8.1]
DESCRIPTIVE [Sect. 8.2]

**Figure 4.** Block-diagram architecture of the proposed learning process.

## 5. THE FUZZY RULE GENERATING METHOD

### 5.1. Generating Descriptive Fuzzy Control Rules

This generating method was introduced in [10]. A previously defined DB constituted by uniform fuzzy partitions with triangular membership functions crossing at height 0.5 is considered. The number of linguistic terms forming each one of them can be specified by the GFS designer in order to obtain the desired granularity level. Figure 3 shows the generic structure of a fuzzy partition with seven linguistic labels.

Each time the generating method is run, it produces a set of candidate fuzzy rules by generating the fuzzy rule best covering every example from the training set. These rules are obtained by taking the fuzzy-partition linguistic label that best matches the example component value for each variable. The accuracy of the candidates is measured by using a multicriterion fitness function, designed to take into account three different criteria. This allows us to ensure the completeness and consistency of the final set of generated rules. Finally, the best fuzzy control rule is selected from the set of candidates and given as process output. The criteria used by the fitness function are:

1. *High frequency value* [21]. The frequency of a fuzzy control rule $R_i$ through the set of examples $E_p$ is defined as

$$\Psi_{E_p}(R_i) = \frac{\sum_{l=1}^{p} R_i(e_l)}{p}.$$

2. *High average covering degree over positive examples* [21]. The set of positive examples to $R_i$ with a compatibility degree greater than or equal to $\omega$ is defined as

$$E_\omega^+(R_i) = \left\{ e_l \in E_p \middle| R_i(e_l) \geq \omega \right\}$$

with $n_\omega^+(R_i)$ being equal to $|E_\omega^+(R_i)|$. The *average covering degree* on $E_\omega^+(R_i)$ can be defined as

$$G_\omega(R_i) = \sum_{e_l \in E_\omega^+(R_i)} \frac{R_i(e_l)}{n_\omega^+(R_i)}.$$

3. *Small negative-example set* [11]. The set of the negative examples for $R_i$ is defined as

$$E^-(R_i) = \left\{ e_l \in E_p \middle| R_i(e_l) = 0 \text{ and } A_i(ex^l) > 0 \right\}.$$

An example is considered negative for a rule when it best matches some other rule that has the same antecedent but a different consequent. The negative examples are always considered over the complete training set. With $n_{R_i}^- = |E^-(R_i)|$ being the number of negative examples, the *penalty function on the negative examples set* will be

$$g_n(R_i^-) = \begin{cases} 1 & \text{if } n_{R_i}^- \leq kn_\omega^+(R_i), \\ \dfrac{1}{n_{R_i}^- - kn_\omega^+(R_i) + \exp(1)} & \text{otherwise}, \end{cases}$$

where we allow up to a certain fraction of the number of positive examples, $kn_\omega^+(R_i)$, of negative examples per rule without any penalty. This fraction is determined by the parameter $k \in [0, 1]$.

These three criteria are combined into a fitness function using any aggregation function increasing in the three variables. In this paper we work with the product in the following way:

$$F(R_i) = \Psi_{E_p}(R_i) G_\omega(R_i) g_n(R_i^-).$$

Rules obtaining higher values of this function will be more accurate.

## 5.2. Generating Approximate Fuzzy Control Rules

It should be noted that the above generating method produces fuzzy control rules of a descriptive nature. All the linguistic labels involved in the rules generated in the different generating method runs will present the same meaning, defined by the primary fuzzy partitions considered for each linguistic variable, and so the final rule set will present a descriptive behavior as well.

Nevertheless, when the ES is applied to optimize the best fuzzy control rule selected from the candidate rule set, it modifies the shapes of the concrete membership functions involved in the rule, without taking into account the meaning of the other rules previously generated. This causes the locally adjusted rule to present an approximate nature. This modification to the generating method, introduced in [11], is described below.

Among the different types of ESs developed until now, we have selected the well-known (1 + 1)-ES introduced in Section 2 for our purpose. This optimization technique was used for the same task in [9, 14] but with two differences. First, it acted as a GA genetic operator in combination with several crossovers and mutations. Secondly, since that approach was based on generating fuzzy rules with constrained free semantics, the modification that the ES developed over the membership functions was also constrained by a set of intervals of performance determined by a previous fuzzy partition. Since in this case we are considering fuzzy rules with unconstrained free semantics, the membership functions of the fuzzy rules obtained from the ES are only constrained to be meaningful.

Below we describe the three main aspects of the designed ES: *coding scheme*, *mutation process*, and *fitness function*. Then we finish this section by analyzing the global behavior of the approximate generating method.

A. CODING SCHEME In order to apply the ES, the fuzzy control rule is encoded into a real string by using the membership function parametric representation introduced in Section 3. Each triangular-shaped membership function involved in the rule is encoded into a 3-tuple of real numbers, and the aggregation of the partial codings forms the ES individual.

B. MUTATION PROCESS Two changes to the generic ES mutation scheme have to be performed in order to apply this technique to the problem considered: the *definition of multiple step sizes* and the *incremental optimization of the individual parameters*. We analyze them below:

- *Definition of multiple step sizes*. As the mutation strength depends directly on the value $\sigma$ of the standard deviation of the normally distributed random variable $z_i$, the step size cannot be a single value. In our case the membership functions encoded are defined over different universes and require different order mutations. Therefore, a step size $\sigma_i = \sigma s_i$ for each component was used in the (1 + 1)-ES. In any case the relations of all $\sigma_i$ were fixed by the values $s_i$, and only the common factor $\sigma$ was adapted, following the assumptions presented in [1].
- *Incremental optimization of the individual parameters*. Usually, the different parent components are not related and the ES is used in its usual working mode in which all of them are adapted at the same time. Unfortunately, in our problem each three correlative parameters, $(x_0, x_1, x_2)$, define a triangular-shaped membership function, and the property $x_0 \leq x_1 \leq x_2$ must be verified in order to obtain meaningful fuzzy sets. Therefore, there is a need to develop an incremental optimization of the individual parameters because the intervals of performance for each one of them will depend on the value of any of the others.

  As we are considering an unconstrained free semantics, a global interval of performance (in which the three parameters defining the membership function may vary freely) is defined for each fuzzy set involved in the fuzzy rule being optimized. With $C_i = (x_0, x_1, x_2)$ being the membership function currently adapted, the associated interval of performance is $[C_i^l, C_i^r] = [x_0 - (x_1 - x_0)/2,\ x_2 + (x_2 - x_1)/2]$. The incremental adaptation is based on generating the mutated fuzzy set $C_i' = (x_0', x_1', x_2')$ by first adapting the modal point $x_1$, obtaining the mutated value $x_1'$ defined in the interval $[x_0, x_2]$, and then adapting the left and right points $x_0$ and $x_2$, obtaining the values $x_0'$ and $x_2'$ defined respectively in the intervals $[C_i^l, x_1']$ and $[x_1', C_i^r]$. It may be clearly observed that the progressive application of this process allows us to obtain fuzzy sets freely defined in the said interval of performance.

The value of the parameter $s(x_i)$ determining the particular step sizes, $\sigma_i = \sigma s(x_i)$, is computed each time the component $x_i$ is going to be mutated. When $i = 1$, i.e., the modal point is being adapted, $s(x_1)$ is equal to $\min(x_1 - x_0, x_2 - x_1)/2$. In the other two cases, $i = 0$ and $i = 2$, we have $s(x_0) = \min(x_0 - C_i^l, x_1' - x_0)/2$ and $s(x_2) = \min(x_2 - x_1', C_i^r - x_2)/2$. Hence when $\sigma$ takes value 1 at the first ES generation, the obtaining of a large number of $z_i$ normal values performing a successful $x_i$ mutation [i.e., the corresponding $x_i' = x_i + z_i$ with $z_i \sim N_i(0, \sigma_i'^2)$ lying in the expected interval for $x_i$] is ensured. If the mutated value lies outside of it, it is assigned the value of the interval extent closer to $x_i + z_i$.

The next algorithm summarizes the application of the adaptation process on a membership function encoded in the parent. With $C_i = (x_0, x_1, x_2)$ being the fuzzy set currently adapted, the steps to follow are:

1. Compute the step size of the central point, $s(x_1) \leftarrow \min\{x_1 - x_0, x_2 - x_1\}/2$.
2. Generate $z_1 \sim N(0, \sigma_1^2)$ and compute $x_1'$ in the following way:

$$x_1' \leftarrow \begin{cases} x_1 + z_1 & \text{if} \quad x_1 + z_1 \in [x_0, x_2], \\ x_0 & \text{if} \quad x_1 + z_1 < x_0, \\ x_2 & \text{if} \quad x_1 + z_1 > x_2. \end{cases}$$

3. Adapt the remaining two points:
   (a) $s(x_0) \leftarrow \min\{x_0 - C_i^l, x_1' - x_0\}/2$.
   Generate $z_0 \sim N(0, \sigma_0^2)$.

$$x_0' \leftarrow \begin{cases} x_0 + z_0 & \text{if} \quad x_0 + z_0 \in \left[C_i^l, x_1'\right], \\ C_i^l & \text{if} \quad x_0 + z_0 < C_i^l, \\ x_1' & \text{if} \quad x_0 + z_0 > x_1'. \end{cases}$$

   (b) $s(x_2) \leftarrow \min\{x_2 - x_1', C_i^r - x_2\}/2$.
   Generate $z_2 \sim N(0, \sigma_2^2)$.

$$x_2' \leftarrow \begin{cases} x_2 + z_2 & \text{if} \quad x_2 + z_2 \in [x_1', C_i^r], \\ x_1' & \text{if} \quad x_2 + z_2 < x_1', \\ C_i^r & \text{if} \quad x_2 + z_2 > C_i^r. \end{cases}$$

C. FITNESS FUNCTION The fitness function is based on the same criteria used in the descriptive gnerating process with the addition of a new one related to the fuzzy-control-rule interaction level. While the interaction level among the neighboring rules in an descriptive rule set is fixed and equal to the union of the primary-fuzzy-set supports $\sigma$ (see the discussion of DB strategy in Section 3.2), this does not occur in an approximate rule set. In the case in which an ES is considered in the generation process, we have selected the approximate approach, and the membership functions differ between some generated fuzzy control rules and others. This causes variable interaction levels among the neighboring rules composing the final rule set generated. Therefore, it is necessary to consider this aspect in the generation process to obtain the best possible rule set representing the knowledge contained in the training data set. To put this into effect, we shall make use of the *GA niche* concept [15, 19]. Below we introduce this concept, analyze some previous applications of it in GFS design, describe its use in our approximate generating method, and present the final expression of the fitness function.

- *The niche concept.* The *niche* and *species* concepts were introduced in GAs in order to improve their behavior when dealing with multimodal functions with peaks of unequal value. As in nature, the formation of stable subpopulations of organisms surrounding separate niches by forcing similar individuals to share the available resources is induced. One of the most usually employed methods for introducing niches and species in GAs is based on *individual fitness sharing* [15, 19]. In this scheme, the population is divided into different subpopulations (species) according to the similarity of the individuals, forming niches in two possible solution spaces: the gene and the decoded parameter ones, *genotypic* and *phenotypic sharing* respectively. The individuals belonging to each niche share the associated payoff amongst them. A *sharing function* is defined to determine the neighborhood and degree of sharing for each string in the population.
- *Previous applications of niching in GFS design.* The idea of using this concept for designing GFSs is not new. Satyadas and Krishnakumar analyze it in [38] and propose a genetic design process using it in [29]. Another process making use of the concept is proposed in [9, 12]. Both schemes are based on phenotypic sharing, which seems to be the most suitable approach for the GFS design problem. However, in the two schemes the purpose of niching is quite different. While the first method employs it to generate different optimal fuzzy models for a concrete control problem, the second one uses it to improve the FLC design, generating a KB formed by fuzzy control rules with a suitable interaction between them.
- *Applying niching to our approximative generating method.* We are going to use the second kind of niche in our generation process, translating it from the GA field to the ES one, by means of the *low niche interaction rate* [9] criterion.

  Since this criterion is based on GA niching, some considerations must be taken into account to design it by means of a sharing function. We noted that the concept of niching is always associated with the GAs and is not usually applied in the ES field. Although the type of ES considered is based on the existence of a single individual parent in the optimization process, the role of the sharing function continues to be the sharing of the global payoff between the individuals located in the same niche. In this case, the only change is that the individuals that share their fitness do not belong to a genetic population. The sharing is developed between the individual currently being optimized and the fuzzy control rules already generated. Therefore, the payoff associated to this individual will be lower when it is closer to a niche center determined by the previously generated rules.

We should also note that one of the most important drawbacks of the classical sharing scheme is the need to know *where* each niche is and *how big* it is in order to allow the fitness sharing. Typically, this requirement is addressed by the assumption that if two individuals are close together, within a distance known as the *niche radius*, then their fitness must be shared. Although several methods have been proposed to determine this radius (see [15]), its calculation is a very difficult task in a large number of cases.

Fortunately, in our case it is easy to determine the location and size of the different existing niches. As we are working in the phenotypic space, each individual represents a fuzzy control rule formed by $n$ input linguistic variables and an output one. Each variable takes as its value a triangular-shaped fuzzy number encoded in the string. Therefore, the center of the niche in the solution space will be an $(n+1)$-dimensional point, whose coordinates corresponds to the modal points of the triangular membership functions. Two individuals will share their payoff if there is any interaction between the different fuzzy numbers giving values to the linguistic variables, i.e., if the fuzzy sets associated to the same variable in the two chromosomes overlap each other. Hence the algorithm does not present a fixed niche radius value as in the classical sharing scheme, but rather the size of the niche depends on the membership-function shapes encoded in the different individuals.

With $N_i = (N_i x, N_i y)$ being the centers of the rules (niches) determined until now ($i = 1, \dots, d$, where $d$ is the number of generating process runs developed), and $C$ being the individual encoding the fuzzy control rule being adapted, the *low niche interaction rate* penalizes the fitness associated to $C$ in the following way:

$$\begin{aligned}
\text{LNIR}(C) &= 1 - \text{NIR}(C),\\
\text{NIR}(C) &= \max_i\{h_i\},\\
h_i &= *(A(N_i x), B(n_i y)), \qquad i = 1, \dots, d,\\
A(N_i x) &= *(A_1(N_i x_1), \dots, A_n(N_i x_n)),\\
C &\sim \text{IF } x_1 \text{ is } A_1 \text{ and} \dots \text{and } x_n \text{ is } A_n \text{ THEN } y \text{ is } B
\end{aligned}$$

Hence LNIR$(C)$ is defined in $[0,1]$. It gives the maximum value (no penalization) when the rule encoded in $C$ does not interact with any of the rules generated until now. The minimum value (maximum penalization) is obtained when this rule is equal to one of those generated previously.

Figure 5 graphically shows a situation where there is interaction between the rule encoded by $C$ and any of the rules generated until now:

- *Final fitness-function expression.* Finally, the fitness function for this approximate generating method is the following:

$$F(R_i) = \Psi_{E_p}(R_i) \cdot G_\omega(R_i) \cdot g_n(R_i^-) \cdot \text{LNIR}(R_i)$$

D. BEHAVIOR AND ADVANTAGES OF THE APPROXIMATE GENERATING METHOD The combined action of the low-niche-interaction criterion and the covering method modifies the fitness landscape on two different levels at each algorithm step. The purpose of these changes in the individual fitness payoff is to encourage the generation of individuals exploring new zones of the solution space in the subsequent runs while penalizing the ones located in existing niches. The two different modification levels are discussed below:

- The covering method removes examples from the training data set, eliminating the payoff associated to the space zones where these examples were located. This is a *high-level modification* in that it

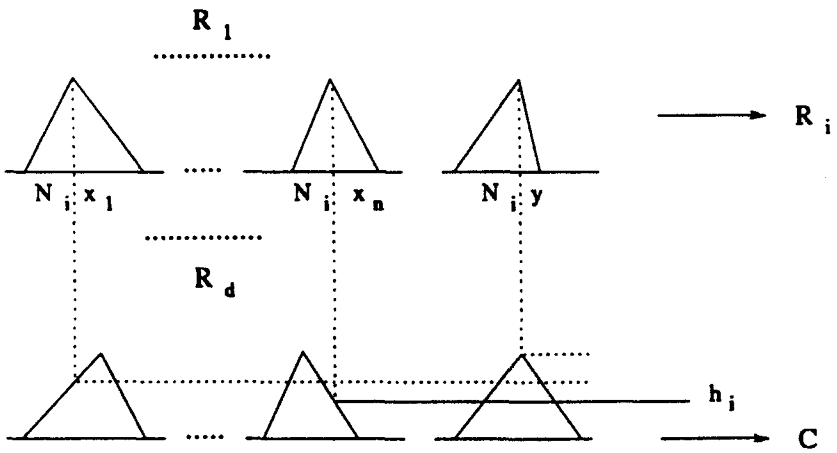

**Figure 5.** Interaction between the current rule and the predetermined ones.

translates the search focus to another space zone. This modification encourages adequate space exploration.

- When a niche has been located in a space zone and it continues to be the most promising one (i.e., the examples located in it have not been yet covered and they have a big associated payoff), new fuzzy rules will be generated in the same zone and they will interact with the ones generated so far. An adequate interaction rate is desirable to make the best use of the FLC interpolative reasoning capabilities. This is obtained by using a *niche penalizing function* that penalizes excessive proximity of the new rule to the previously generated ones.
  The three frequentistic criteria for the fitness function try to widen the supports of the generated fuzzy control rules to extend their applicability and cover more examples. The niching criterion tries to narrow their support by penalizing excessive proximity. A combination of these four criteria enables us to obtain a suitable interaction level among neighboring rules.
  This is a *low-level modification* in that the algorithm continues working in the same space zone but penalizes excessive proximity to the niches located therein. This modification encourages adequate space exploitation.

Therefore, the approximate generation process will allow us to verify the two following fundamental aspects:

- The process will ensure that fuzzy control rules are obtained in each space zone in which the control problem is defined, i.e., in each zone in which any example exists. The KB completeness is verified in this way.
- In the same way, it will maintain an adequate rule distribution in each one of the niches present in the solution space. A suitable interaction between the KB fuzzy control rules is thus obtained.

## 5.3. The Generating Algorithm

Finally, the generating method may be summarized in the following algorithm:

- Initialize the candidate fuzzy rule set $B^c$ to empty.
- For every $e_l \in E_p$, generate the fuzzy rule $R_c$ best covering it by taking the linguistic label of the fuzzy partition best matching with the component value of $e_l$ for each variable. If $R_c \notin B^c$, add it to $B^c$.
- Evaluate all the fuzzy rules contained in $B^c$, and select the one yielding the highest value of the fitness function, $R_r$.
- If desired, optimize $R_r$ by using the proposed ES for locally tuning the membership functions involved in it, obtaining a new fuzzy control rule of approximate nature.

## 6. THE COVERING METHOD

The covering method is developed as an iterative process that allows us to obtain a set of fuzzy rules covering the example set. In each iteration, it runs the generating method, obtaining the best fuzzy control rule according to the current state of the training set, considers the relative covering value this rule imposes on it, and removes from it the examples with a covering value greater than $\epsilon$. The covering method is developed as follows:

```
1. Initialization:
   (a) Introduce $k$, $\omega$, and $\epsilon$.
   (b) Set the example covering degree $CV[l] \leftarrow 0$, $l = 1, \ldots, p$.
   (c) Initialize the final set of rules $B^g$ to empty.
2. Over the set of examples $E_p$, apply the generating method, obtaining
   as output the best fuzzy control rule $R_r$ according to the current state
   of $E_p$.
3. Introduce $R_r$ in $B^g$.
4. For every $e_l \in F_p$ do
   (a) $CV[l] \leftarrow CV[l] + R_r(e_l)$,
   (b) If $CV[l] \geq \epsilon$, then remove it from $E_p$.
5. If $E_p = \emptyset$, then STOP, else return to step 2.
```

Since two similar rules may be obtained, i.e., $B^g$ may present redundant rules, it is necessary to simplify the complete rule set obtained from this process for deriving the final KB, thereby allowing the system to be controlled.

## 7. THE GENETIC SIMPLIFICATION PROCESS

It is possible that the iterative nature of the generation process may cause overlearning. This occurs when some examples are covered to a higher degree than the desired one, and it makes the obtained RB perform worse due to the existence of redundant rules. In order to solve this problem and improve its accuracy, it is necessary to simplify the rule set obtained from the previous process, removing the redundant rules for deriving the final RB allowing the system to be controlled.

The simplification process used was proposed in [21]. It is based on a binary-coded GA, in which the selection of the individuals is developed using the stochastic universal sampling procedure together with an elitist selection scheme, and the generation of the offspring population is put into effect by using the classical binary multipoint crossover (performed at two points) and uniform mutation operators.

The encoding scheme generates fixed-length chromosomes. Considering the rules contained in the rule set $B^g$ derived from the previous step counted from 1 to $m$, an $m$-bit string $C = (c_1, \ldots, c_m)$ represents a subset of candidate rules to form the RB finally obtained as this stage's output, $B^s$, such that

$$\text{IF} \quad c_i = 1 \quad \text{THEN} \quad R_i \in B^s \quad \text{ELSE} \quad R_i \notin B^s.$$

The initial population is generated by introducing a chromosome representing the complete previously obtained rule set $B^g$, that is, with all $c_i = 1$. The remaining chromosomes are selected at random.

As regards the fitness function, $E(\cdot)$, it is based on an application-specific measure usually employed in the design of GFSs, the mean squared error (SE) over a training data set, $E_{\text{TDS}}$, which is represented by the following expression:

$$E(C_j) = \frac{1}{2|E_{\text{TDS}}|} \sum_{e_l \in E_{\text{TDS}}} [ey^l - S(ex^l)]^2,$$

where $S(ex^l)$ is the output value obtained from the FLC using the RB coded in $C_j$, $R(C_j)$, when the state variable values are $ex^l$, and $ey^l$ is the known desired value.

There is a need to keep the *completeness property* considered in the previous stage. We will ensure this condition by forcing every example contained in the training set to be covered by the encoded RB to a degree greater than or equal to $\tau$,

$$C_{R(C_j)}(e_l) = \bigcup_{j=1,\ldots,T} R_j(e_l) \geq \tau \qquad \forall e_l \in E_{\text{TDS}} \text{ and } R_j \in R(C_j),$$

where $\tau$ is the minimal training-set completeness degree accepted in the simplification process. Usually, $\tau$ is less than or equal to $\omega$, the compatibility degree used in the generation process.

Therefore, we define a *training-set completeness degree* of $R(C_j)$ over the set of examples $E_{\text{TDS}}$ as

$$\text{TSCD}\big(R(C_j), E_{\text{TDS}}\big) = \bigcap_{e_l \in E_{\text{TDS}}} C_{R(C_j)}(e_l),$$

and the final fitness function penalizing the lack of the completeness property is

$$F(C_j) = \begin{cases} E(C_j) & \text{if} \quad \text{TSCD}\big(R(C_j), E_{\text{TDS}}\big) \geq \tau, \\ \frac{1}{2}\sum_{e_l \in E_{\text{TDS}}} (ey^l)^2 & \text{otherwise}. \end{cases}$$

## 8. THE GENETIC TUNING PROCESS

The genetic tuning process is based on two variants, depending on whether the fuzzy model is approximate [20] or descriptive [10]. Both processes are based on the previous existence of a complete KB, that is, an initial DB definition and a RB constituted by $m$ fuzzy control rules. As we are going to see below, the only difference between the two processes is the coding scheme. While the first of them encodes the whole KB into the chromosomes for adjusting each one of the membership functions involved in the fuzzy control rules independently, the chromosomes considered in the second one only encode the primary fuzzy partitions constituting the DB in order to adjust the linguistic labels' membership functions for all the fuzzy control rules contained in the RB.

The GA designed for both tuning processes presents a real coding issue and uses the stochastic universal sampling as selection procedure together with an elitist scheme. The operators employed for performing the individual recombination and mutation are Michalewicz's nonuniform mutation [36] and the max-min arithmetical crossover [20]. A short description thereof is presented below:

- *Nonuniform mutation.* If $C_v^t = (c_1, \dots, c_k, \dots, c_H)$ is a chromosome and the element $c_k$ was selected for this mutation (the domain of $c_k$ is $[c_{kl}, c_{kr}]$), the result is a vector $C_v^{t+1} = (c_1, \dots, c'_k, \dots, c_H)$, with $k \in 1, \dots, H$ and

$$c'_k = \begin{cases} c_k + \Delta(t, c_{kr} - c_k) & \text{if} \quad a = 0, \\ c_k - \Delta(t, c_k - c_{kl}) & \text{if} \quad a = 1, \end{cases}$$

where $a$ is a random number that may have a value of zero or one, and the function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to 0 increases as $t$ increases:

$$\Delta(t, y) = y(1 - r^{(1-t/T)^b}),$$

where $r$ is a random number in the interval $[0, 1]$, $T$ is the maximum number of generations, and $b$ is a parameter chosen by the user, which determines the degree of dependence on the number of iterations. This property causes this operator to make a uniform search in the initial space when $t$ is small, and a very local one in later stages.

- *Max-min arithmetical crossover.* If $C_v^t = (c_1, \dots, c_k, \dots, c_H)$ and $C_w^t = (c_1', \dots, c_k', \dots, c_H')$ are to be crossed, the following four offspring are generated:

$$\begin{aligned} &C_1^{t+1} = aC_w^t + (1-a)C_v^t, \\ &C_2^{t+1} = aC_v^t + (1-a)C_w^t, \\ &C_3^{t+1} \quad \text{with} \quad c_{3k}^{t+1} = \min\{c_k, c_k'\}, \\ &C_4^{t+1} \quad \text{with} \quad c_{4k}^{t+1} = \max\{c_i, c_k'\}. \end{aligned}$$

  This operator can use a parameter $a$ which is either a constant, or a variable whose value depends on the age of the population. The resulting descendents are the two best of the four aforesaid offspring.

The preliminary definition of the fitness function was presented in [20] and was based on using a training input-output data set, $E_{\mathrm{TDS}}$, and a concrete error measure, the mean squared error. In this way, the adaptation value associated to an individual $C_j$ was obtained by computing the error between the outputs given by the FLC using the KB encoded in the chromosome and those contained in the training data set. The fitness function was represented by the following expression:

$$E(C_j) = \frac{1}{2|E_{\mathrm{TDS}}|} \sum_{e_l \in E_{\mathrm{TDS}}} [ey^l - S(ex^l)]^2.$$

In this paper we are going to consider an extension of this fitness function in order to keep the completeness property considered in the two previous stages. Hence, the fitness function defined for the genetic simplification process, $F(C_j)$, is going to be used in the tuning one as well.

Once the aspects common to both genetic tuning processes have been introduced, there is a need to present the particular ones for each of them. The only differences between the two processes are the coding scheme and the generation of the initial population. These particular aspects of both variants are commented on in the following subsections.

## 8.1. The Approximative Genetic Tuning Process

As mentioned earlier, each chromosome forming the genetic population will encode a complete KB. More concretely, all of them encode the derived RB, $R^s$, obtained as output from the simplification process, and the differences between them are the fuzzy-control-rule membership functions.

Taking into account the parametric representation of the triangular-shaped membership functions based on a 3-tuple of real values introduced in Section 3, each one of the rules is encoded in pieces of the chromosome $C_{ji}$, $i = 1, \dots, m = n + 1$, in the following way:

$$C_{ji} = (a_{i1}, b_{i1}, c_{i1}, \dots, a_{in}, b_{in}, c_{in}, a_i, b_i, c_i).$$

Therefore the complete RB with an associated DB is represented by a complete chromosome $C_j$.

$$C_j = C_{j1} C_{j2} \dots C_{jm}.$$

The initial gene pool is created from the initial KB. This KB is encoded directly into a chromosome, denoted as $C_1$. The remaining individuals are generated by associating an interval of performance, $[c_h^l, c_h^r]$ to every gene $c_h$ in $C_1$, $h = 1, \dots, (n + 1) \cdot m \cdot 3$. Each interval of performance will be the interval of adjustment for the corresponding variable, $c_h \in [c_h^l, c_h^r]$.

If $t \bmod 3 = 1$, then $c_t$ is the left value of the support of a fuzzy number. The fuzzy number is defined by the three parameters $(c_t, t_{t+1}, c_{t+2})$, and the intervals of performance are the following:

$$c_t \in \left[c_t^l, c_t^r\right] = \left[c_t - \frac{c_{t+1} - c_t}{2}, c_t + \frac{c_{t+1} - c_t}{2}\right],$$

$$c_{t+1} \in \left[c_{t+1}^l, c_{t+1}^r\right] = \left[c_{t+1} - \frac{c_{t+1} - c_t}{2}, c_{t+1} + \frac{c_{t+2} - c_{t+1}}{2}\right],$$

$$c_{t+2} \in \left[c_{t+2}^l, c_{t+2}^r\right] = \left[c_{t+2} - \frac{c_{t+2} - c_{t+1}}{2}, c_{t+2} + \frac{c_{t+3} - c_{t+2}}{2}\right].$$

Figure 6 shows these intervals.

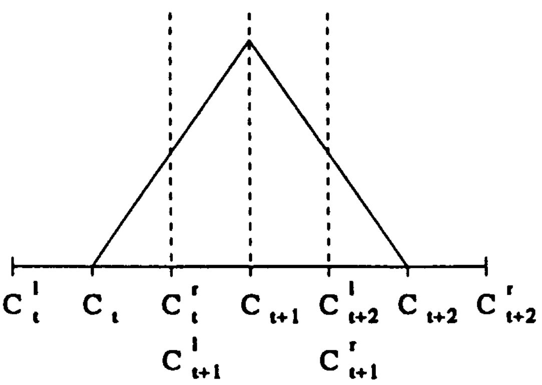

**Figure 6.** Membership function and intervals of performance for the tuning process.

Therefore, we create a population of chromosomes which presents $C_1$ as its first individual and the remaining ones initiated randomly, with each gene being in its respective interval of performance.

### 8.2. The Descriptive Genetic Tuning Process

This second genetic tuning process is a modified version of the approximate one. In this case each chromosome encodes a different DB definition. A primary fuzzy partition is represented as an array composed of $3N$ real values, with $N$ being the number of terms forming the linguistic-variable term set. The complete DB for a problem, in which $m$ linguistic variables are involved, is encoded into a fixed-length real coded chromosome $C_j$ built by joining the partial representations of each one of the variable fuzzy partitions as is shown in the following:

$$c_{ji} = (a_{i1}, b_{i1}, c_{i1}, \ldots, a_{iN_i}, b_{iN_i}, c_{iN_i}),$$

$$C_j = C_{j1} C_{j2} \ldots C_{jm}.$$

The initial gene pool is created making use of the initial DB definition. It is encoded directly into a chromosome, denoted as $C_1$. The remaining individuals are generated in the interval of performance associated to each membership function. As in the approximate approach, the interval of performance associated to every gene $c_h$ in $C_1$, $[c_h^l, c_h^r]$, $h = 1, \ldots, \Sigma_{i=1}^{m} 3N_i$, will be the interval of adjustment for the corresponding gene, $c_h \in [c_h^l, c_h^r]$.

## 9. APPLICATION OF THE LEARNING PROCESS TO THE FUZZY MODELING OF THREE THREE-DIMENSIONAL MATHEMATICAL FUNCTIONS

To analyze the accuracy of the GFS proposed, we have selected three $n$-dimensional mathematical functions to derive theoretical three-dimensional control surfaces. The mathematical functions and the variable universes of discourse considered are shown below. The *spherical model*, $F_1$, is a unimodal function; the generalized *Rastrigin function*, $F_2$, is a strongly multimodal one; and the third function, $F_3$, is a smooth one presenting discontinuities at (0, 0) and (1, 1)—as may be observed in their graphical representations (Figure 7):

$$F_1(x_1, x_2) = x_1^2 + x_2^2,$$

$$x_1, x_2 \in [-5, 5], \qquad F_1(x_1, x_2) \in [0, 50];$$

$$F_2(x_1, x_2) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2,$$

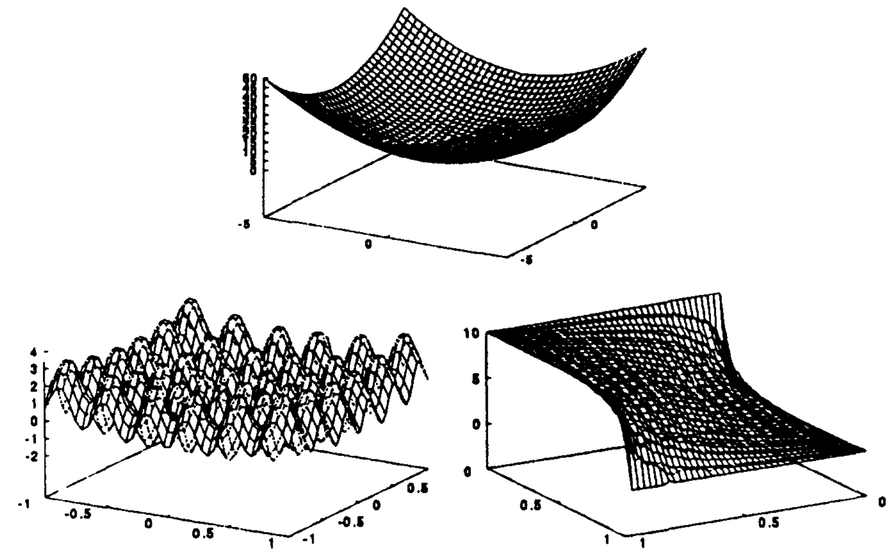

**Figure 7.** Graphical representations of $F_1$ (at the top), $F_2$ (lower left), and $F_3$ (lower right).

$$x_1, x_2 \in [-1, 1], \qquad F_2(x_1, x_2) \in [2, 3.5231];$$

$$F_3(x_1, x_2) = 10 \frac{x_1 - x_1 x_2}{x_1 - 2x_1 x_2 + x_2},$$

$$x_1, x_2 \in [0, 1], \qquad F_3(x_1, x_2) \in [0, 10].$$

These surfaces will be approximated by different fuzzy models derived from several design methods. In the descriptive fuzzy model, the three following processes are considered:

**D1.** the widely employed Wang-Mendel (WM) method [43],

**D2.** a two-stage GFS based on obtaining a complete KB by deriving the RB by means of the WM method and defining the DB by means of the descriptive genetic tuning method (see Section 8) constituting the third stage of the descriptive-fuzzy-model design process proposed, and

**D3.** the three-stage descriptive-GFS proposed in this paper.

In the approximative fuzzy model, the following two processes are employed:

**A1.** a two-stage GFS based on obtaining a complete KB by deriving the RB by means of the WM method and defining the DB by means of the approximate genetic tuning method (see Section 6), and

**A2.** the three-stage approximate GFS proposed in this paper.

For each of the functions, a training data set uniformly distributed in the three-dimensional definition space has been obtained experimentally. In this way, two sets with 1681 values have been generated for functions $F_1$ and $F_2$ by taking 41 values for each one of the two state variables considered to be uniformly distributed in their respective intervals. As the intervals of the two input variables are shorter for $F_3$, the training set has been generated in the same way, but considering only 26 values. The final data set for this function is composed of 674 values (instead of 676), because it is not defined at two space points.

Three other data sets have been generated for use as test sets for evaluating the performance of the learning method, avoiding any possible bias related to the data in the training set. The size of these data sets is a percentage of the size of the corresponding training set, 10 percent to be precise. The data are obtained by generating at random the state-variable values in the concrete universes of discourse for each one of them, and computing the associated output-variable value. Hence two test sets formed by 168 data and one by 67 are used to measure the accuracy of the FLCs designed by computing the mean squared error for them.

The initial DB used in the generating process is constituted by three primary fuzzy partitions (two corresponding to the state variables and one to the control variable) formed by seven *linguistic terms* with triangular-shaped fuzzy sets giving meaning to them (as shown in Figure 3), and the appropriate scaling factors to translate the generic universe of discourse into the one associated with each problem variable.

The following parameter values, corresponding to the first two stages, are combined for determining the different runs of the method to be carried out: $\epsilon = \{1, 1.5\}$, $\omega = 0.05$, $k = 0.1$, and $\tau = \{0.1, 0.25, 0.5\}$. This leads to an overall total of 12 runs per function when joined to the two different fitness functions used in the tuning process, $E(\cdot)$ and $F(\cdot)$. With respect to the remaining parameters, the $t$-norm $*$ used in the rule generation process is the minimum, the ES in the approximate generation process is applied until there is no improvement in 100 generations (the parameter $c$ of the $\frac{1}{5}$-success rule is equal to 0.9), and the genetic simplification and tuning processes run over 500 and 1000 generations, respectively. In all cases, the population is formed by 61 individuals, the value of the nonuniform mutation parameter $b$ is 5.0, and the crossover and mutation rates are, respectively, $P_c = 0.6$ and $P_m = 0.1$ (this last one per individual). The max-min arithmetical crossover parameter $a$ takes the value 0.35.

Finally, as regards to the FLC reasoning method used, we have selected the *minimum t-norm* to play the role of the implication and conjunctive

**Table 1.** Fuzzy Modeling of $F_1$ Using Design Methods D1, D2, and D3

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 118 | 4.3968 | 4.7109 | 62 | 2.0651 | 1.9786 | 0.3358 | 0.2625 |
| 1.0 | 0.1 | $F(\cdot)$ | 118 | 4.3968 | 4.7109 | 62 | 2.0651 | 1.9786 | 0.4138 | 0.2895 |
| 1.0 | 0.25 | $E(\cdot)$ | 118 | 4.3968 | 4.7109 | 65 | 2.1996 | 2.0667 | 0.4081 | 0.3233 |
| 1.0 | 0.25 | $F(\cdot)$ | 118 | 4.3968 | 4.7109 | 65 | 2.1996 | 2.0667 | 0.4142 | 0.3358 |
| 1.0 | 0.5 | $E(\cdot)$ | 118 | 4.3968 | 4.7109 | 74 | 2.5674 | 2.4023 | 0.5036 | 0.3974 |
| 1.0 | 0.5 | $F(\cdot)$ | 118 | 4.3968 | 4.7109 | 74 | 2.5674 | 2.4023 | 0.4900 | 0.4385 |
| 1.5 | 0.1 | $E(\cdot)$ | 167 | 3.5187 | 3.4986 | 90 | 1.4530 | 1.5083 | 0.2854 | 0.3049 |
| 1.5 | 0.1 | $F(\cdot)$ | 167 | 3.5187 | 3.4986 | 90 | 1.4530 | 1.5083 | 0.2931 | 0.3142 |
| 1.5 | 0.25 | $E(\cdot)$ | 167 | 3.5187 | 3.4986 | 91 | 1.6709 | 1.6225 | 0.3459 | 0.3174 |
| 1.5 | 0.25 | $F(\cdot)$ | 167 | 3.5187 | 3.4986 | 91 | 1.6709 | 1.6225 | 0.3389 | 0.3122 |
| 1.5 | 0.5 | $E(\cdot)$ | 167 | 3.5187 | 3.4986 | 97 | 1.6791 | 1.7355 | 0.3279 | 0.3138 |
| 1.5 | 0.5 | $F(\cdot)$ | 167 | 3.5187 | 3.4986 | 97 | 1.6791 | 1.7355 | 0.3335 | 0.3132 |
| | | D1 | 49 | 2.0481 | 2.2559 | | | D2 | 0.3585 | 0.3771 |

operators, and the *center of gravity weighted by the matching strategy* as the defuzzification operator [12].

The results obtained in the different experiments are collected in Tables 1–6, where #R stands for the number of rules of the corresponding KB, FT for the fitness function used in the genetic tuning process, and $SE_{tra}$ and $SE_{tst}$ for the values obtained by the concrete FLC in the SE measure computed over the training and test data sets, respectively. The last table

**Table 2.** Fuzzy Modeling of $F_1$ Using Design Methods A1 and A2

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 102 | 4.7940 | 3.6363 | 66 | 3.2559 | 2.6212 | 1.6892 | 1.1688 |
| 1.0 | 0.1 | $F(\cdot)$ | 102 | 4.7940 | 3.6363 | 66 | 3.2559 | 2.6212 | 1.6679 | 1.1591 |
| 1.0 | 0.25 | $E(\cdot)$ | 102 | 4.7940 | 3.6363 | 73 | 3.4921 | 2.8006 | 1.8755 | 1.3989 |
| 1.0 | 0.25 | $F(\cdot)$ | 102 | 4.7940 | 3.6363 | 73 | 3.4921 | 2.8006 | 1.9751 | 1.5777 |
| 1.0 | 0.5 | $E(\cdot)$ | 102 | 4.7940 | 3.6363 | 79 | 3.6715 | 2.8745 | 2.0005 | 1.5199 |
| 1.0 | 0.5 | $F(\cdot)$ | 102 | 4.7940 | 3.6363 | 79 | 3.6715 | 2.8745 | 2.1323 | 1.5180 |
| 1.5 | 0.1 | $E(\cdot)$ | 128 | 4.1043 | 3.5518 | 76 | 2.5293 | 1.9513 | 1.4829 | 0.9607 |
| 1.5 | 0.1 | $F(\cdot)$ | 128 | 4.1043 | 3.5518 | 76 | 2.5293 | 1.9513 | 1.4629 | 0.9518 |
| 1.5 | 0.25 | $E(\cdot)$ | 128 | 4.1043 | 3.5518 | 80 | 2.6445 | 2.1978 | 1.4588 | 1.1293 |
| 1.5 | 0.25 | $F(\cdot)$ | 128 | 4.1043 | 3.5518 | 80 | 2.6445 | 2.1978 | 1.4716 | 1.2680 |
| 1.5 | 0.5 | $E(\cdot)$ | 128 | 4.1043 | 3.5518 | 86 | 2.8155 | 2.3975 | 1.6923 | 1.2228 |
| 1.5 | 0.5 | $F(\cdot)$ | 128 | 4.1043 | 3.5518 | 86 | 2.8155 | 2.3975 | 1.8189 | 1.3936 |
| | | | 49 | 2.0481 | 2.2559 | | | A1 | 0.4530 | 0.4543 |

**Table 3.** Fuzzy Modeling of $F_2$ Using Design Methods D1, D2, and D3

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 281 | 0.5563 | 0.6390 | 195 | 0.4758 | 0.5195 | 0.3692 | 0.4091 |
| 1.0 | 0.1 | $F(\cdot)$ | 281 | 0.5563 | 0.6390 | 195 | 0.4758 | 0.5195 | 0.3766 | 0.4236 |
| 1.0 | 0.25 | $E(\cdot)$ | 281 | 0.5563 | 0.6390 | 198 | 0.4882 | 0.5411 | 0.3824 | 0.4234 |
| 1.0 | 0.25 | $F(\cdot)$ | 281 | 0.5563 | 0.6390 | 198 | 0.4882 | 0.5411 | 0.3991 | 0.4500 |
| 1.0 | 0.5 | $E(\cdot)$ | 281 | 0.5563 | 0.6390 | 214 | 0.4967 | 0.5539 | 0.3732 | 0.4152 |
| 1.0 | 0.5 | $F(\cdot)$ | 281 | 0.5563 | 0.6390 | 214 | 0.4967 | 0.5539 | 0.4123 | 0.4574 |
| 1.5 | 0.1 | $E(\cdot)$ | 373 | 0.5805 | 0.6721 | 249 | 0.4699 | 0.4997 | 0.3676 | 0.3795 |
| 1.5 | 0.1 | $F(\cdot)$ | 373 | 0.5805 | 0.6721 | 249 | 0.4699 | 0.4997 | 0.3723 | 0.3877 |
| 1.5 | 0.25 | $E(\cdot)$ | 373 | 0.5805 | 0.6721 | 243 | 0.4841 | 0.5189 | 0.3780 | 0.3932 |
| 1.5 | 0.25 | $F(\cdot)$ | 373 | 0.5805 | 0.6721 | 243 | 0.4841 | 0.5189 | 0.3918 | 0.4093 |
| 1.5 | 0.5 | $E(\cdot)$ | 373 | 0.5805 | 0.6721 | 263 | 0.4827 | 0.5184 | 0.3708 | 0.3828 |
| 1.5 | 0.5 | $F(\cdot)$ | 373 | 0.5805 | 0.6721 | 263 | 0.4827 | 0.5184 | 0.4019 | 0.4264 |
| | | D1 | 49 | 1.7783 | 2.0490 | | | D2 | 0.8141 | 0.9641 |

row shows the results corresponding to both design processes based on the WM KB generation method (the "Generation" columns are related to the single WM method, D1, and the "Tuning" column to the two-stage GFS, D2 or A1).

Analyzing these results, the good behavior presented by the proposed GFS may be observed. All the FLCs designed using it are much more accurate than the ones based on the WM RB generation method in the

**Table 4.** Fuzzy Modeling of $F_2$ Using Design Methods A1 and A2

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 279 | 0.2903 | 0.3017 | 193 | 0.2031 | 0.2141 | 0.1878 | 0.1997 |
| 1.0 | 0.1 | $F(\cdot)$ | 279 | 0.2903 | 0.3017 | 193 | 0.2031 | 0.2141 | 0.1868 | 0.2050 |
| 1.0 | 0.25 | $E(\cdot)$ | 279 | 0.2903 | 0.3017 | 202 | 0.2242 | 0.2374 | 0.2046 | 0.2127 |
| 1.0 | 0.25 | $F(\cdot)$ | 279 | 0.2903 | 0.3017 | 202 | 0.2242 | 0.2374 | 0.2045 | 0.2203 |
| 1.0 | 0.5 | $E(\cdot)$ | 279 | 0.2903 | 0.3017 | 223 | 0.2361 | 0.2573 | 0.2167 | 0.2340 |
| 1.0 | 0.5 | $F(\cdot)$ | 279 | 0.2903 | 0.3017 | 223 | 0.2361 | 0.2573 | 0.2180 | 0.2392 |
| 1.5 | 0.1 | $E(\cdot)$ | 368 | 0.2870 | 0.3421 | 245 | 0.1810 | 0.2142 | 0.1674 | 0.1988 |
| 1.5 | 0.1 | $F(\cdot)$ | 368 | 0.2870 | 0.3421 | 245 | 0.1810 | 0.2142 | 0.1680 | 0.2025 |
| 1.5 | 0.25 | $E(\cdot)$ | 368 | 0.2870 | 0.3421 | 240 | 0.1961 | 0.2495 | 0.1820 | 0.2268 |
| 1.5 | 0.25 | $F(\cdot)$ | 368 | 0.2870 | 0.3421 | 240 | 0.1961 | 0.2495 | 0.1823 | 0.2293 |
| 1.5 | 0.5 | $E(\cdot)$ | 368 | 0.2870 | 0.3421 | 267 | 0.2133 | 0.2623 | 0.1988 | 0.2459 |
| 1.5 | 0.5 | $F(\cdot)$ | 368 | 0.2870 | 0.3421 | 267 | 0.2133 | 0.2623 | 0.1997 | 0.2434 |
| | | | 49 | 1.7783 | 2.0490 | | | A1 | 1.0188 | 1.2032 |

**Table 5.** Fuzzy Modeling of $F_3$ Using Design Methods D1, D2, and D3

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 106 | 0.1818 | 0.1037 | 67 | 0.0381 | 0.0467 | 0.0185 | 0.0210 |
| 1.0 | 0.1 | $F(\cdot)$ | 106 | 0.1818 | 0.1037 | 67 | 0.0381 | 0.0467 | 0.0194 | 0.0219 |
| 1.0 | 0.25 | $E(\cdot)$ | 106 | 0.1818 | 0.1037 | 67 | 0.0559 | 0.0446 | 0.0292 | 0.0191 |
| 1.0 | 0.25 | $F(\cdot)$ | 106 | 0.1818 | 0.1037 | 67 | 0.0559 | 0.0446 | 0.0291 | 0.0180 |
| 1.0 | 0.5 | $E(\cdot)$ | 106 | 0.1818 | 0.1037 | 81 | 0.1347 | 0.0687 | 0.0604 | 0.0246 |
| 1.0 | 0.5 | $F(\cdot)$ | 106 | 0.1818 | 0.1037 | 81 | 0.1347 | 0.0687 | 0.0715 | 0.0338 |
| 1.5 | 0.1 | $E(\cdot)$ | 156 | 0.2207 | 0.1043 | 97 | 0.0370 | 0.0348 | 0.0185 | 0.0181 |
| 1.5 | 0.1 | $F(\cdot)$ | 156 | 0.2207 | 0.1043 | 97 | 0.0370 | 0.0348 | 0.0182 | 0.0208 |
| 1.5 | 0.25 | $E(\cdot)$ | 156 | 0.2207 | 0.1043 | 98 | 0.0549 | 0.0382 | 0.0273 | 0.0167 |
| 1.5 | 0.25 | $F(\cdot)$ | 156 | 0.2207 | 0.1043 | 98 | 0.0549 | 0.0382 | 0.0325 | 0.0192 |
| 1.5 | 0.5 | $E(\cdot)$ | 156 | 0.2207 | 0.1043 | 110 | 0.1269 | 0.0631 | 0.0599 | 0.0315 |
| 1.5 | 0.5 | $F(\cdot)$ | 156 | 0.2207 | 0.1043 | 110 | 0.1269 | 0.0631 | 0.0728 | 0.0457 |
| | | D1 | 49 | 0.1943 | 0.0444 | | | D2 | 0.0602 | 0.0286 |

fuzzy modeling of the three functions. The two process variants make it stronger because they allow it to tackle many different kinds of control surfaces; smooth ones, like those generated by means of the functions $F_1$ and $F_3$, are best modeled by using descriptive KBs, whilst the approximate approach seems to work better with complex surfaces with strong changes, such as the one generated from $F_2$.

**Table 6.** Fuzzy Modeling of $F_3$ Using Design Methods A1 and A2

| Parameters | | | Generation | | | Simplification | | | Tuning | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | $\tau$ | FT | #R | $SE_{tra}$ | $SE_{tst}$ | #R | $SE_{tra}$ | $SE_{tst}$ | $SE_{tra}$ | $SE_{tst}$ |
| 1.0 | 0.1 | $E(\cdot)$ | 101 | 0.2255 | 0.1608 | 65 | 0.1457 | 0.1129 | 0.0863 | 0.0766 |
| 1.0 | 0.1 | $F(\cdot)$ | 101 | 0.2255 | 0.1608 | 65 | 0.1457 | 0.1129 | 0.0965 | 0.0930 |
| 1.0 | 0.25 | $E(\cdot)$ | 101 | 0.2255 | 0.1608 | 65 | 0.1507 | 0.3056 | 0.0929 | 0.3171 |
| 1.0 | 0.25 | $F(\cdot)$ | 101 | 0.2255 | 0.1608 | 65 | 0.1507 | 0.3056 | 0.0986 | 0.1778 |
| 1.0 | 0.5 | $E(\cdot)$ | 101 | 0.2255 | 0.1608 | 70 | 0.1777 | 0.3622 | 0.1153 | 0.3918 |
| 1.0 | 0.5 | $F(\cdot)$ | 101 | 0.2255 | 0.1608 | 70 | 0.1777 | 0.3622 | 0.1230 | 0.3275 |
| 1.5 | 0.1 | $E(\cdot)$ | 132 | 0.1912 | 0.1057 | 87 | 0.1043 | 0.0902 | 0.0795 | 0.0597 |
| 1.5 | 0.1 | $F(\cdot)$ | 132 | 0.1912 | 0.1057 | 87 | 0.1043 | 0.0902 | 0.0695 | 0.0648 |
| 1.5 | 0.25 | $E(\cdot)$ | 132 | 0.1912 | 0.1057 | 93 | 0.1053 | 0.2930 | 0.0714 | 0.0531 |
| 1.5 | 0.25 | $F(\cdot)$ | 132 | 0.1912 | 0.1057 | 93 | 0.1053 | 0.2930 | 0.0721 | 0.0522 |
| 1.5 | 0.5 | $E(\cdot)$ | 132 | 0.1912 | 0.1057 | 90 | 0.1113 | 0.3274 | 0.0786 | 0.0537 |
| 1.5 | 0.5 | $F(\cdot)$ | 132 | 0.1912 | 0.1057 | 90 | 0.1113 | 0.3274 | 0.0774 | 0.0664 |
| | | | 49 | 0.1943 | 0.0444 | | | A1 | 0.0562 | 0.0210 |

Two drawbacks may be associated to the proposed process, the KBs generated always present more fuzzy control rules than the WM-based methods, and the approximate fuzzy model makes the KB lose its readability:

- A larger number of fuzzy rules does not constitute a drawback when working with real problems in which the main requirement is the accuracy of the control and not the speed of the controller response. Furthermore, large numbers of fuzzy control rules can always be compiled for increased run-time performance [5]. The proposed learning process allows the GFS designer to obtain the desired value in the relationship between accuracy and number of rules in the KB by controlling the value of the parameter $\epsilon$ in the evolutionary learning process. The higher the value of this parameter is, the more rules the generated KB presents, but more accurate is the FLC in the great majority of the cases. In the same way, the lower it is, the fewer rules are obtained, but usually less accurate the FLC is.
- The loss of KB readability may be justified by the benefit obtained by the improved FLC accuracy, which is a main concern in many real control problems.

To illustrate the behavior of the proposed process, some graphical representations of the fuzzy modeling obtained on the selected functions are shown in figures 8 and 9. Each figure is drawn using the designed FLCs

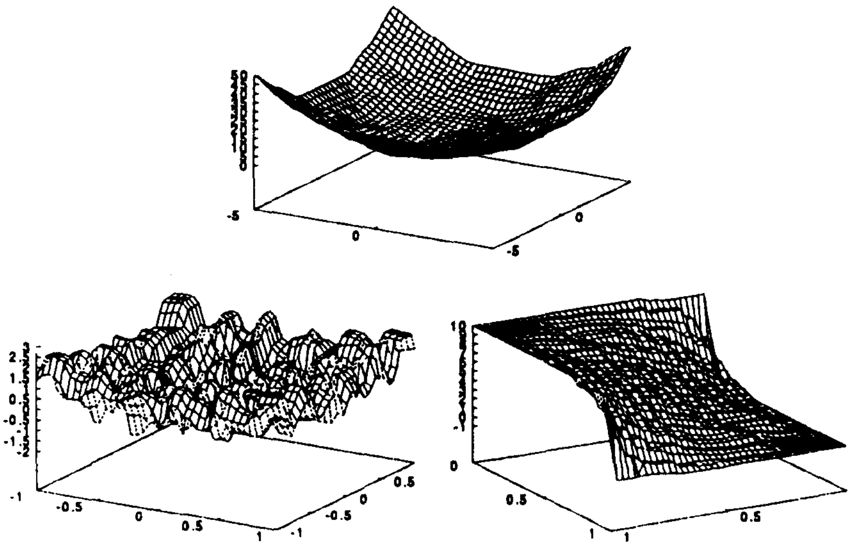

**Figure 8.** Graphical representations of the fuzzy modeling obtained for $F_1$ (at the top), $F_2$ (lower left), and $F_3$ (lower right).

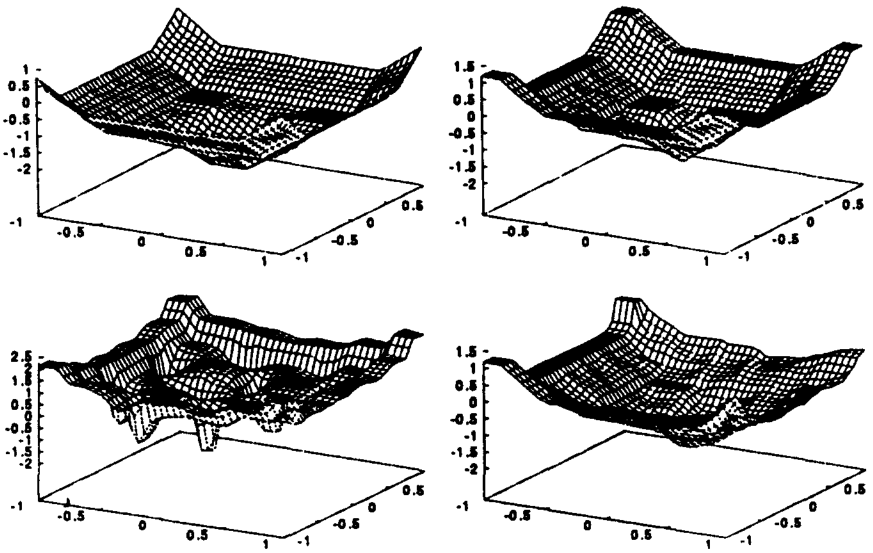

**Figure 9.** Graphical representations of the fuzzy modeling obtained for $F_2$ using methods D1 (top left), D2 (top right), D3 (lower left), and A2 (lower right).

best approaching each one of the functions. These FLCs have been selected making use of an index weighting the final value obtained by the controller in the SE measure over both data sets by the number of data they contain in the following way:

$$M = \frac{|E_{\text{tra}}| \cdot \text{SE}_{\text{tra}} + |E_{\text{tst}}| \cdot \text{SE}_{\text{tst}}}{|E_{\text{tra}}| + |E_{\text{tst}}|}.$$

Figure 8 shows the obtained fuzzy modeling for the functions $F_1$, $F_2$, and $F_3$ by means of the FLCs presenting lower values in this measure. The first and third plots correspond to the runs of the design method D3, the descriptive GFS proposed, using the parameter values $\epsilon = 1.5$ and $\tau = 0.1$, while the second one corresponds to A2, our approximate GFS, using the same parameter values.

Figure 9 presents the remaining fuzzy modelings obtained for the second function with the purpose of highlighting the difference between the descriptive and approximate approaches when dealing with a complex function. In view of these representations and the ones shown in Figure 8, we can conclude that the approximative approach works better for this kind of functions, though presenting a similar number of fuzzy control rules in the KB (249 in the KB derived using method D3, and 245 in the one obtained from A2).

## 10. CONCLUDING REMARKS

A GFS has been presented for designing FLCs by learning the KB from examples, combining an iterative and ES-based generation and two GA-based simplification and tuning processes. Its application to the generation of KBs presenting descriptive and approximate behavior for different kinds of problems has been considered. Its performance in the fuzzy modeling of three three-dimensional control surfaces has been shown and compared with other methods based on the WM process. The proposed GFS has obtained good results.

It has been observed that each one of the KB approaches has performed better in a given kind of problem. While the smooth control surfaces, like the ones generated by means of $F_1$ and $F_3$, are better modeled by using descriptive KBs, the approximate approach works better when applied to complex and rough surfaces with strong changes, such as the one generated from $F_2$. This fact makes the proposed process more robust, because it is able to work with control surfaces of different natures by means of its two variants.

## References

1. Bäck, T., and Schwefel, H.-P., Evolution strategies I: Variants and their computational implementation, in *Genetic Algorithms in Engineering and Computer Science* (J. Periaux, G. Winter, M. Galán, P. Cuesta, Eds.), Wiley, 111–126, 1995.

2. Beasly, D., Bull, D. R., and Martin, R. R., A sequential niche technique for multimodal function optimization, *Evolutionary Comput.* 1(2), 101–125 (1993).

3. Bolata, F., and Nowé, A., From fuzzy linguistic specifications to fuzzy controllers using evolution strategies, *Proceedings of the Fourth IEEE International Conference on Fuzzy Systems* (*FUZZ-IEEE'95*), Yokohama, Japan, 1089–1094, 1995.

4. Bonarini, A., ELF: Learning incomplete fuzzy rule sets for an autonomous robot, *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies* (*EUFIT'93*), Aachen, Germany, 69–75, Sept. 1993.

5. Bonissone, P. P., A compiler for fuzzy logic controllers, *Proceedings of the International Fuzzy Engineering Symposium* (*IFES'91*), 706–717, Nov. 1991.

6. Cooper, M. G., and Vidal, J. J., Genetic design of fuzzy logic controllers, *Proceedings of the Second International Conference on Fuzzy Theory and Technology (FTT'93)*, Durham, 1993.

7. Carse, B., Fogarty, T. C., and Munro, A., Evolving fuzzy rule based controllers using genetic algorithms, *Fuzzy Sets and Systems* 80, 273–293, 1996.

8. Cordón, O., and Herrera, F., A general study on genetic fuzzy systems, in *Genetic Algorithms in Engineering and Computer Science* (J. Periaux, G. Winter, M. Galán, P. Cuesta, Eds.), Wiley, 33–57, 1995.

9. Cordón, O., and Herrera, F., A hybrid genetic algorithm-evolution strategy process for learning fuzzy logic controller knowledge bases, in *Fuzzy Logic and Soft Computing* (F. Herrera, and J. L. Verdegay, Eds.), Physica-Verlag, 251–278, 1996.

10. Cordón, O., Herrera, F., and Lozano, M., A three-stage method for designing genetic fuzzy systems by learning from examples, in *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV)* (H. M. Voight, W. Ebeling, E. Rechemberg, H. P. Schwefel, Eds.), Lecture Notes in Comput. Sci. 1141, Berlin, 720–729, 1996.

11. Cordón, O., and Herrera, F., Generating and selecting fuzzy control rules using evolution strategies and genetic algorithms, *Proceedings of the Information Processing and Management of Uncertainly in Knowledge-Based Systems (IPMU'96)*, Granada, Spain, 733–738, 1996.

12. Cordón, O., Herrera, F., and Peregrín, A., Applicability of the fuzzy operators in the design of fuzzy logic controllers, *Fuzzy Sets and Systems*, 86, 15–41, 1997.

13. Cordón, O., Herrera, F., and Lozano, M., A classified review on the combination fuzzy logic–genetic algorithms bibliography: 1989–1995, in *Genetic Algorithms and Fuzzy Logic Systems. Soft Computing Perspectives* (E. Sanchez, T. Shibata, L. Zadeh, Eds.), World Scientific, 1997.

14. Cordón, O., and Herrera, F., Identification of linguistic fuzzy models by means of genetic algorithms, in Fuzzy Identification: A User's Handbook (D. Driankov, H. Hellendoorn, R. Pabu, Eds.), Springer Verlag, 1997.

15. Deb, K., and Goldberg, D. E., An investigation of niche and species formation in genetic function optimization, *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum, Hillsdale, N.J., 42–50, 1989.

16. Driankov, D., Hellendoorn, H., and Reinfrank, M., *An Introduction to Fuzzy Control*, Springer-Verlag, 1993.

17. Fogel, L. J., Owens, A. J., and Walsh, M. J., *Artificial Intelligence through Simulated Evolution*, Wiley, New York, 1966.

18. Fogel, D. B., *System Identification Through Simulated Evolution. A Machine Learning Approach*, Ginn Press, USA, 1991.

19. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.

20. Herrera, F., Lozano, M., and Verdegay, J. L., Tuning fuzzy controllers by genetic algorithms, *Internat. J. Approx. Reason.* 12, 1995, 299–315.

21. Herrera, F., Lozano, M., and Verdegay, J. L., A learning process for fuzzy control rules using genetic algorithms, *Fuzzy Sets and Systems*, to appear.

22. Herrera, F., Lozano, M., and Verdegay, J. L., Generating fuzzy rules from examples using genetic algorithms, in *Fuzzy Logic and Soft Computing* (B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, Eds.), World Scientific, 11–20, 1995.

23. Herrera, F., and Verdegay, J. L. (Eds.), *Genetic Algorithms and Soft Computing*, Physica-Verlag, 1996.

24. Hoffman, F., and Pfister, G., A new learning method for the design of hierarchical fuzzy controllers using messy genetic algorithms, *Proceedings of the Sixth International Fuzzy Systems Association World Congress (IFSA'95)*, Sao Paulo, Brazil, 249–252, July 1995.

25. Holland, J. H., Adaptation in natural and artificial systems, Univ. of Michigan Press, Ann Arbor, 1975; MIT Press, London, 1992.

26. Karr, C. L., Genetic algorithms for fuzzy controllers, *AI Expert*, 26–33, 1991.

27. Karr, C. L., Applying genetics to fuzzy logic, *AI Expert*, 38–43, 1991.

28. Kinzel, J., Klawonn, F., and Kruse, R., Modifications of genetic algorithms for designing and optimizing fuzzy controllers, *Proceedings of the First IEEE Conference on Evolutionary Computation (EC-IEEE'94)*, Orlando, 28–33, June 1994.

29. Krishnakumar, K., and Satyadas, A., Evolving multiple fuzzy models and its application to an aircraft control problem, in *Genetic Algorithm in Engineering and Computer Science* (J. Periaux, G. Winter, M. Galán, P. Cuesta, Eds.), Wiley, 305–320, 1995.

30. Kropp, K., and Baitinger, U. G., Optimization of fuzzy logic controller inference rules using a genetic algorithm, *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, Aachen, Germany, 1090–1096, Sept. 1993.

31. Lee, C. C., Fuzzy logic in control systems: fuzzy logic controller—parts I, II, *IEEE Trans. Systems Man Cybernet.* 20, 404–435, 1990.

32. Lee, M. A., and Takagi, H., Embedding apriori knowledge into an integrated fuzzy system design method based on genetic algorithms, *Proceedings of the Fifth International Fuzzy Systems Association World Congress (IFSA'93)*, Seoul, 1293–1296, July 1993.

33. Leitch, D., and Probert, P., Context depending coding in genetic algorithms for the design of fuzzy systems, *Proceedings of the IEEE/Nagoya University WWW on Fuzzy Logic and Neural Networks/Genetic Algorithms*, Nagoya, 1994.

34. Magdalena, L., and Velasco, J. R., Fuzzy rules-based controllers that learn by evolving its knowledge base, in *Fuzzy Logic and Soft Computing* (F. Herrera and J. L. Verdegay, Eds.), Physica-Verlag, 172–201, 1996.

35. Mamdani, E. H., and Assilian, S., An experiment in linguistic synthesis with a fuzzy logic controller, *Internat. J. Man-Machine Stud.* 7, 1–13, 1975.

36. Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

37. Ng, K. C., and Li, Y., Design of sophisticated fuzzy logic controllers using genetic algorithms, *Proceedings of the Third IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'94)*, Orlando, 1708–1712, June 1994.

38. Satyadas, A., and Krishnakumar, K., Evolutionary learning techniques for fuzzy controller synthesis, *Proceedings of the First Industry/University Symposium on High Speed Civil Transport Vehicles*, N.C., Dec. 1994.

39. Satyadas, A., and Krishnakumar, K., EFM-based controllers for space station attitude control: Application and analysis, in *Fuzzy Logic and Soft Computing* (F. Herrera and J. L. Verdegay, Eds.), Physica-Verlag, 152–171, 1996.

40. Schwefel, H.-P., *Evolution and Optimum Seeking*, Sixth-Generation Comput. Technol. Ser., Wiley, 1995.

41. Surmann, H., Kanstein, A., and Goser, K., Self-organizing and genetic algorithms for an automatic design of fuzzy control and decision systems, *Proceedings of the First European Congress on Fuzzy and Intelligent Technologies (EUFIT'93)*, Aachen, Germany, 1097–1104, Sept. 1993.

42. Thrift, P., Fuzzy logic synthesis with genetic algorithms, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, San Diego, 509–513, July 1991.

43. Wang, L. X., and Mendel, J. M., Generating fuzzy rules by learning from examples, *IEEE Trans. Systems Man Cybernet.* 22, 1414–1427, 1992.