A Hybrid Genetic Algorithm-Evolution Strategy Process for Learning Fuzzy Logic Controller Knowledge Bases *

Oscar Cordón and Francisco Herrera

Dept. of Computer Science and Artificial Intelligence.
E.T.S. de Ingeniería Informática.
University of Granada, 18071 - Granada, Spain

Abstract. 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. The performance of the method proposed is shown by measuring the accuracy of the Fuzzy Logic Controllers designed in the modeling of two three-dimensional control surfaces derived from two mathematical functions presenting different characteristics. The results obtained by a method based on the Wang and Mendel's Knowledge Base generation process are also shown, allowing to compare both processes.

Keywords. Fuzzy logic controllers, Knowledge Bases, learning, genetic algorithms, evolution strategies

1 Introduction

Fuzzy logic controllers (FLCs), initiated by Mamdani and Assilian in the work [17], 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 a knowledge base (KB), to control it (for more information about FLCs see [8, 16]). The advantage of this approach with respect to classical *Control Theory* is that it has no necessity of expressing 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 presented by the human process operators to express their knowledge in form of control rules has made the researches to develop automatic

* This research has been supported by DGICYT PB92-0933

techniques for performing this task. In the last few years, many different approaches have been presented taking the Genetic Algorithms as a base, obtaining the so called Genetic Fuzzy Systems (GFSs). In this paper we present a hybrid genetic algorithm-evolution strategy process for designing GFSs by learning the KB from examples. The process performance is shown by using it to develop a fuzzy modeling of two three-dimensional control surfaces derived from two mathematical functions.

The method proposed consists of the following three steps, maintaining the generic structure used in [13]:

1. A *genetic generation process* for generating fuzzy control rules, with a generating method based on Evolutionary Algorithms, and a covering method of the system behaviour example set. This process allows us to obtain a set of rules covering the training set in an adequate form.
2. A *genetic simplification process* for simplifying rules, based on a binary coded Genetic Algorithm and a measure of the FLC performance in the control of the system being identified. It will save the overlearning that the previous component may cause.
3. A *genetic tuning process*, based on a Real Coded Genetic Algorithm 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.

In order to do this, we arrange the paper as follows. Section 2 presents a short introduction to Evolutionary Algorithms and GFSs. Some preliminaries about the FLC KB type considered are discussed in Section 3. The three stages of the proposed learning process are introduced respectively in Sections 4, 5 and 6; while Section 7 shows its application to the fuzzy modeling of the commented functions. Finally, some conclusions are pointed out in Section 8.

2 Evolutionary Algorithms and Genetic Fuzzy Systems

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 models that have been proposed and studied which are referred 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: *Genetic Algorithms* (GAs), *Evolution Strategies* (ESs), and *Evolutionary Programming* (EP).

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. But there are also important differences between them. 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.

Each individual in the population receives a measure of its fitness in the environment. Selection focuses attention on high fitness individuals, thus making use of the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for exploration. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

Natural evolution is a most robust yet efficient problem-solving technique. Evolutionary Computation can likewise made robust. The same procedures may be applied to diverse problems with relatively little reprogramming [9].

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

2.1 Genetic Algorithms

GAs are theoretically and empirically proven to provide robust search in complex spaces, thereby offering a valid approach to problems requiring efficient and effective search. The basic principles of the GAs were first laid down rigorously by Holland [15], and are well described in many texts such as [10, 18].

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.

Although there are many possible variants of the basic GA, the fundamental underlying mechanism operates on a population of chromosomes or individuals (representing possible solutions to the problem) and consists of three operations:

1. evaluation of individual fitness,
2. formation of a gene pool (intermediate population), and
3. recombination and mutation.

It is generally accepted that a GA must take into account the five following components for solving a problem:

1. *A genetic representation of the problem solutions,*
2. *a way to create an initial population of solutions,*
3. *an evaluation function which gives the fitness of each individual,*
4. *genetic operators that alter the genetic composition of children during reproduction, and*
5. *values for the parameters that the GA uses (population size, probabilities of applying genetic operators, etc.).*

2.2 Evolution Strategies

ESs were developed with a strong focus on building systems capable of solving difficult read-valued parameter optimization problems. The natural representation was a vector or real-valued genes which were manipulated primarily by mutation operators designed to perturb the real-valued parameters in useful ways.

ESs were initially developed by Rechenberg and Schwefel in 1964 as experimental optimization techniques. The first ES algorithm, the so-called *(1+1)-ES*, was based on working with only two individuals per generation, one parent and one descendent. Other more complex variants, based on considering a high number of parents ($\mu > 1$) and descendents ($\lambda > 1$) have appeared in later years, constituting the so called $(\mu + \lambda) - ES$. In the last few years, several new generalized ESs have been succesfully developed [1, 19].

Without a lack of generality, in this paper we work with the *(1+1)-ES*, the most simple ES model. In the following we briefly describe this scheme [1, 19]:

(1+1)-ES is based on encoding the possible optimization problem solution into a real coded string. This parent string is evolved by applying a mutation operator over each one of its components. The mutation strength is determined by a value σ, a 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, the offspring obtained by mutation is better adapted than its parent, then the descendent substitutes it 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.

As may be observed, the main component of the model is the mutation operator, **mut**. It is composed of two components, $\mathbf{mu}_\sigma$, which updates the value of the parameter σ, and $\mathbf{mu}_x$, which evolves the real coded string. The first one is based on Rechenberg's 1/5-success rule, which evolves the standard deviation according to the current value of the relative frequency p of succesful mutations, in the following way:

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

The second one mutates each component of the real coded string by adding normally distributed variations with standard deviation σ' to it:

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

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

The final algorithm process is the following:

Procedure Evolution Strategy (1+1)
begin (1)

$t = 0$;
initialize $P(t) \leftarrow (x, \sigma)$;
evaluate $f(x)$;
While (**Not** *termination-condition*) **do**
begin (2)
$t = t + 1$;
$(x^{'}, \sigma^{'}) \leftarrow \mathbf{mut}(\mathbf{x}, \sigma)$;
evaluate $f(x^{'})$;
If Better $(f(x^{'}), f(x))$
then $P(t+1) \leftarrow (x^{'}, \sigma^{'})$
else $P(t+1) \leftarrow P(t)$.
end (2)
end (1)

2.3 Genetic Fuzzy Systems

The KB is the FLC component comprising the expert knowledge known about the controlled system. So it is the only component of the FLC depending on the concrete application and it makes the accuracy of the FLC depends directly on its composition. It is comprised of two components, a *Data Base* (DB), containing the definitions of the fuzzy control rules linguistic labels, and a *Rule Base* (RB), constituted by the collection of fuzzy control rules representing the expert knowledge.

The KB derivation is the only task that have to be performed in order to design an FLC directly depending on the controlled system and it presents a significative importance in the design process [5, 8, 16]. It is known that the more used method for performing this task is based directly on extracting the expert experience from the 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, researches have been investigating automatic learning methods for designing FLCs by deriving automatically an appropiate KB for the controlled system without necessity of its human operator.

GAs have been demonstrated to be a powerful tool for automating the definition of the KB, since adaptative 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 have produced quite promising results. These approaches can receive the general name of *Genetic Fuzzy Systems* (GFSs) [4]. Figure 1 shows this idea.

Using the more general term *"evolutionary"* instead of *"genetic"* (when an EA is used instead of a GA) they may be called *Evolutionary Fuzzy Systems*.

GAs are applied to modify/learn the DB and/or the RB. It is possible to distinguish three different groups of *genetic FLC design processes* according to the KB component included in the learning process. These ones are the following:

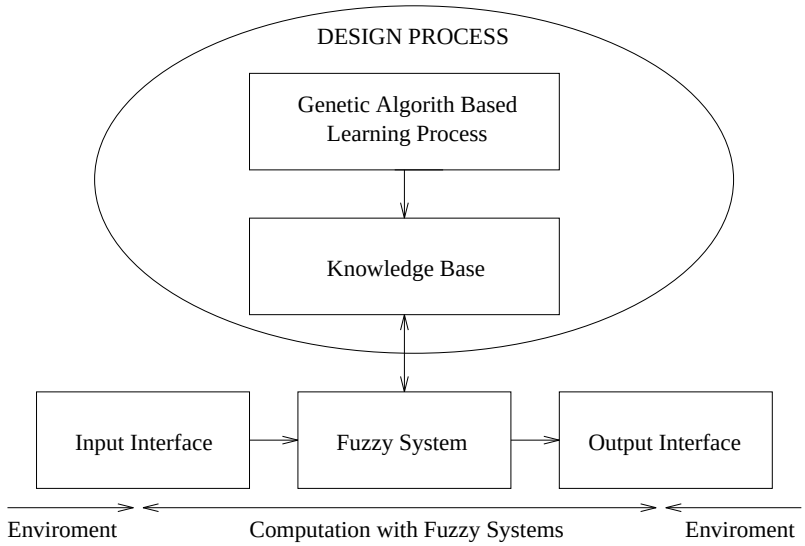


Fig. 1. Genetic fuzzy systems

1. *Genetic definition of the Fuzzy Logic Controller Data Base*
2. *Genetic derivation of the Fuzzy Logic Controller Rule Base*
3. *Genetic learning of the Fuzzy Logic Controller Knowledge Base*

For a wider description see [4] and for an extensive bibliography see [6] (section 3.13).

In this paper we present a GFS design method belonging to the third abovementioned family. In this way, making use of our process it will be possible to automatically generate a complete FLC KB when a training set formed by numerical input-output (state-control) problem variable pairs experimentally recorded is available.

3 Preliminaries

In this work, 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:

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

where $x_1, ..., x_n$ and y are the process state variables and the control variable respectively; and $A_{i1}, ..., A_{in}$, B are fuzzy sets in the universes of discourse $U_1, ..., U_n, V$.

These fuzzy sets are characterized by their membership functions

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

In our study 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-tuple (a_{ij}, b_{ij}, c_{ij}), (a_i, b_i, c_i), $j = 1, ..., n$.

We assume that we have a description of the control strategy in the form of an input-output data set without noise which we shall use as a base in our learning process for obtaining the KB.

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 represent a real world semantic.

It can be considered a KB for which fuzzy rules either present different meaning for the same linguistic terms or the fuzzy quantities have not any associated linguistic label. In this case, the KB and the FLC using it, present a different philosophy, the approach is *approximative* [4]. In this second approach we say that the rules present *free semantic*.

We will center on this second approach. For the generation process we consider rules with a free semantic, without any linguistic syntaxis associated to the rules, but based on an initial domain fuzzy partition. Using this approach we may say that the rules present *constrained free semantic.*

Each universe, U, contains a number of overlaping regions labeled with linguistic terms, forming a finite set of fuzzy sets on U. For instance, if X is a variable on U for temperature, then one may define A_1 as "*low temperature*", $A_i (1 < i < r)$ as "*medium temperature*" and A_r as "*high temperature*", etc.

These referential fuzzy sets are characterized by their membership functions $A_i(u) : U \rightarrow [0,1], i = 1, ..., r$. To ensure the performance of the fuzzy model and to provide to uniform basis for further study, it is essential that all the referential sets should be normal convex ones, and should satisfy the following completeness condition:

$$\forall u \in U\ \exists j,\ 1 \leq j \leq r, \text{ such that } A_j(u) \geq \delta$$

and δ is a fixed threshold, this being the *completeness degree* of the universes. Figure 2 shows an example of a fuzzy partition with $\delta = 0.5$.

The number of linguistic terms forming the fuzzy partition associated to each linguistic variable can be specified by the GFS designer in order to obtain the desired granularity level.

Making use of this previous fuzzy partition, an interval of performance, defined as follows, is associated to each one of the three points defining the membership functions $A_t(\cdot)$, (a_t, b_t, c_t)

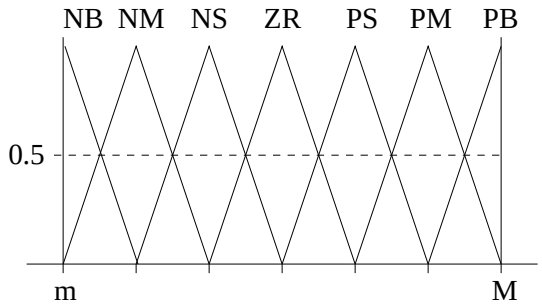


Fig. 2. Graphical representation of a possible fuzzy partition

$$[a_t^l, a_t^r] = [a_t - \frac{b_t - a_t}{2}, a_t + \frac{b_t - a_t}{2}]$$

$$[b_t^l, b_t^r] = [b_t - \frac{b_t - a_t}{2}, b_t + \frac{c_t - b_t}{2}]$$

$$[c_t^l, c_t^r] = [c_t - \frac{c_t - b_t}{2}, c_t + \frac{c_t - b_t}{2}]$$

for locally adjusting their parameters during the generating process. Figure 3 shows the intervals of performance associated to each one of the parameters.

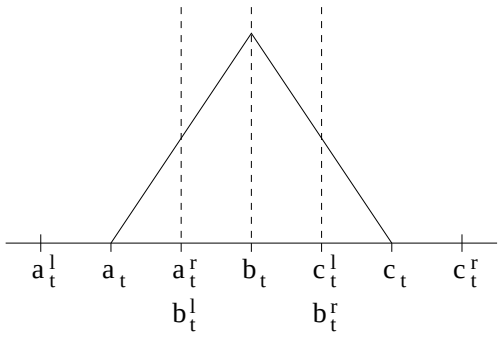


Fig. 3. Membership function and intervals of performance for the generating process

Therefore the fuzzy control rules generated will have their semantic within the performance interval established by the fuzzy partition membership functions.

4 The Genetic Generation Process

As it has been commented, the first stage consists of two processes, a *generating method* of desirable fuzzy rules from examples and a *covering method* of the set of examples.

1. The fuzzy rule generating method is developed by means of a GA encoding a single fuzzy rule in each chromosome. The GA finds the best rule in every run over the set of examples according to the features included in the GA fitness function. An ES is used for locally tuning the best fuzzy control rules obtained in the genetic search iterations.
2. The covering method is developed as an iterative process. It allows a set of fuzzy rules to be obtained covering the set of examples. In each iteration, it runs the generating method choosing the best chromosome (rule), considers the relative covering value that this rule provokes over the example set and removes the examples with a covering value greater than a value ϵ provided by the controller designer.

Next subsections present both methods in-depth.

4.1 The Fuzzy Rule Generating Process

The generating method for fuzzy rules is developed by means of a special GA, where a chromosome encodes a fuzzy rule and an ES that locally tunes the fuzzy rules. We describe the EA components below.

4.1.1 Representation

A chromosome C encoding a candidate rule is composed of two different parts, C_1 and C_2, each one corresponding to each one of the KB components. The first part of the chromosome encodes the linguistic rule (belonging to the RB), and the second one the meaning associated to each linguistic term involved in the rule (belonging to the DB).

In order to represent the first part there is a need to number the linguistic labels belonging to each one of the linguistic variable term sets. A variable x_i taking values in a term set $T(x_i) = \{L_1(x_i), \ldots, L_{n_i}(x_i)\}$ has associated the ordered set $T^{'}(x_i) = \{1, \ldots, n_i\}$.

On the other hand, the second part adopts the same representation employed in the unconstrained free semantic generating process presented in [13]. Each one of the triangular membership functions composing the rule, $L_i(x_j)$, is encoded by means of its associated 3-tuple $(a_{L_i(x_j)}, b_{L_i(x_j)}, c_{L_i(x_j)})$.

Hence, the following generic rule

$$\text{IF } x_1 \text{ is } L_{i_1}(x_1) \text{ ... and } x_n \text{ is } L_{i_n}(x_n) \text{ THEN } y \text{ is } L_{i_{n+1}}(y)$$

is encoded into a chromosome C with the following form:

$$C_1 = (i_1, \ldots, i_n, i_{n+1})$$
$$C_2 = (a_{L_{i_1}(x_1)}, b_{L_{i_1}(x_1)}, c_{L_{i_1}(x_1)}, \ldots, a_{L_{i_n}(x_n)}, b_{L_{i_n}(x_n)}, c_{L_{i_n}(x_n)},$$
$$a_{L_{i_{n+1}}(y)}, b_{L_{i_{n+1}}(y)}, c_{L_{i_{n+1}}(y)})$$
$$C = C_1 C_2$$

Now, the fundamental underlying mechanisms of a GA, formation of an initial gene pool, fitness function, and genetic operators are developed.

4.1.2 Initial Gene Pool

A third of the initial gene pool is created making use of the examples contained in the training set, E_p, and other third is initiated totally at random. The initialization of the individuals belonging to remainder third takes common characteristics with the other two. The first part of them is initiated from the examples, and the second one at random.

With M being the GA population size and $t = \min\{|E_p|, \frac{M}{3}\}$, let t examples be generated at random from E_p. Then, the initial population generation process is performed in three steps as follows:

1. Making use of the existing linguistic variable primary fuzzy partitions, generate t individuals by taking the rule best covering each one of the t randomly obtained examples. Initiate C_1 and C_2 by coding respectively the rule linguistic terms and their meaning in the way commented on above.
2. Generate another t individuals initiating C_1 in the same way followed in the previous step, and computing the values of C_2 at random, each gene varying in its respective interval.
3. Generate the remaining $M - 2 \cdot t$ individuals by computing at random the values of the first part, $C1$, and making use of these for randomly generating the C_2 part, each gene varying in its respective interval.

4.1.3 Evaluation of Individual Fitness

The fitness function measuring the adaptation of each rule of the population is a multiobjective function based on several criteria. We present some requeriments and the commented criteria below.

Requeriments: For generating a set of rules R describing the behaviour of a system, it is necessary to establish a condition for it. This is the requirement of covering all possible situation-action pairs, $e_l \in E_p$, the *completeness property* [8, 16]. This may be formalized for a constant $\tau \in [0, 1]$, it requires the non-zero union of fuzzy sets $A_i(\cdot)$, $B_i(\cdot)$, $i = 1, ..., T$, $T = |R|$, and is formulated by the following expressions:

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

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

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

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 set of rules R, the *covering value* of an example e_l is defined as

$$CV_R(e_l) = \sum_{i=1}^{T} R_i(e_l)$$

and we require the following condition

$$CV_R(e_l) \geq \epsilon \quad l = 1, ..., p.$$

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

High frequency value [13]: The frequency of a fuzzy control rule, R_i, through the set of examples, E_p, is defined as:

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

with $R_i(e_l)$ being the *compatibility degree* between the rule R_i and the example e_l.

High average covering degree over positive examples [13]: The set of positive examples to R_i with compatibility degree greater than or equal to ω is defined as:

$$E_\omega^+(R_i) = \{e_l \in E_p / R_i(e_l) \geq \omega\}$$

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:

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

Small negative example set [11]: The set of the negative examples for R_i is defined as:

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

An example is considered negative for a rule when it better 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:

$$g_n(R_i{}^-) = \begin{cases} 1 & \text{if } n^-_{R_i} \leq k \cdot n^+_\omega(R_i) \\ \frac{1}{n^-_{R_i} - k n^+_\omega(R_i) + exp(1)} & \text{otherwise} \end{cases}$$

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

Low niche interaction rate: As it was commented in [4], there are different approaches for designing GFSs by learning the complete FLC KB. In many cases, the difference among them is the genetic representation employed. As it have been presented in section 4.1.1, we work with chromosomes encoding a single fuzzy control rule in the genetic generation process. This representation makes the solution space to be *strongly multimodal* because each possible fuzzy control rule determines a peak in it.

The problem is that when dealing with multimodal functions with peaks of unequal value, simple GAs are characterized by converging to the best peak of the space (or to a space zone containing several of the best peaks) and to lose an adequate individual sampling over other peaks in other space zones. This phenomena is called *genetic drift* and is not a correct behavior for several kinds of problems in which one may be interested in knowing the location of other function optima. In our case, this fact will provoke the generation of a non accurated final KB not satisfying the completeness property due to the absence of fuzzy control rules in space zones with a small reward in the fitness function.

The *niche* and *species* concepts were introduced in order to overcome this behavior [7, 10]. As the great majority of the GA concepts, they are based on traslating natural notions to the field of GAs. In nature, a niche is viewed as an organism's task in the enviroment and a species is a collection of individuals with similar features. In this way, 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 niche and species in GAs is based on the *individual fitness sharing* [7, 10]. In this scheme, the population is divided in different subpopulations (species) according to the similarity of the individuals. These subpopulations form niches in two possible solution spaces: the gene and the decoded parameter ones, *genotypic* and *phenotypic*

sharing respectively. Acting as in nature, the individuals belonging to each niche share the associated payoff among them. A *sharing function* is defined to determine the neighbourhood and degree of sharing for each string in the population.

In [3] it is presented a multimodal problem optimization method for obtaining the desired number of optima of multimodal functions that makes use of the commented niche concept. The so called *Sequential Niche Technique* is based on iterate an unimodal function optimization process (concretelly, a GA) that gives a multimodal function optimum at each run. Each one of these optima constitute a niche center because they are respectively the best solutions found in different space zones. With the purpose of moving the search focus further away from the zones in which optima have been yet located for finding new ones, a derating function modifying the fitness landscape according to the distance between the individuals and the previously located niches is used.

In order to develop and adequate search in the fuzzy rule multimodal space, the generation process structure presented in [13] and used in this paper makes the algorithm work in a similar way. A basic GA (fuzzy rule generating method) is iterated for a number of times, obtaining the best fuzzy control rule with respect to the current training set state from each run performed. Then the influence of this rule over this set is considered by running the covering method. This process modyfies the fitness landscape due to it removes the examples yet covered in a desired degree from the training set, guiding the search focus to another space zone.

This work mode allows to carry over the knowledge learned in one run to each subsequent one. After each run of the algorithm, the location of a new niche is known and can be taken into account for the remaining runs. In this way, the proposed method works in an increasing way.

The performance of this generation method have been demostrated at the sight of the results obtained in [13]. Anyway, the process may be improved making use newly of the niche concept and of the FLC working basis. It is known that the FLC accuracy is due to the interpolative reasoning they develop. A concrete process state usually fires more than one fuzzy control rule and the interaction among these rules is what allows the FLC to obtain the best control action for this state. In this way, the FLC performance and smoothness depend on the existence of an adequate interaction rate between the KB fuzzy control rules at each problem space zone. Rules too close in the problem space cause an undesirable *overlearning* due to their excessive interaction makes the inferred control action move from the optimal one, while remote rules make the FLC lose their interpolation capability, performing equally badly.

The commented generation process allows to verify the completeness property, i. e., to obtain fuzzy rules in all the problem space zones in which there exist examples but does not get a suitable interaction among them. It will be desirable that neighbour fuzzy rules interact adequatelly in order to obtain a high performance KB. The adition of a new criterion to the generating method fitness function will alow to obtain the desired behavior. Individuals encoding fuzzy control rules which are very close to one of the rules previously generated (i. e., individuals located in the same niche) will be penalized (they are very near

to the niche center and have to share their fitness with it). This will encourage an adequate exploitation of the space zones in which a niche have been yet located.

A niche scheme is an adequate GA component in order to design the desired criterium. In this case, the most recommended sharing scheme seems to be a phenotypic one because using it we work directly with the fuzzy control rules.

One of the most important drawbacks associated to the classical sharing scheme is that there exists need of knowning *where* each niche is and *how big* is it in order to allow the fitness sharing. This fact is approached assuming that if two individuals are close together, within a distance known as *niche radius*, then their fitness must be shared. The problem is that in a big quantity of cases, although several methods have been proposed in order to determine its value (see [7]), the calculation of this radius is a very difficult task.

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 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, each one of its components being the corresponding triangular membership function modal point. Two individuals will share their payoff if there is any interaction among the different fuzzy numbers giving value to the linguistic variables, i. e., if the fuzzy sets associated to the same variable in both chromosomes overlap each one. Hence the algorithm does not present a fixed niche radius value as in the classical sharing scheme but 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 ($i = 1, \ldots, d$, where d is the number of generating process runs developed), and C is an individual from the current population, the *low niche interaction rate* penalizes the fitness associated to C in the following way:

$$LNIR(C) = 1 - NIR(C)$$

$$NIR(C) = Max_i\{h_i\}$$

$$h_i = *(A(N_i x), B(N_i y)), i = 1, \ldots, d$$

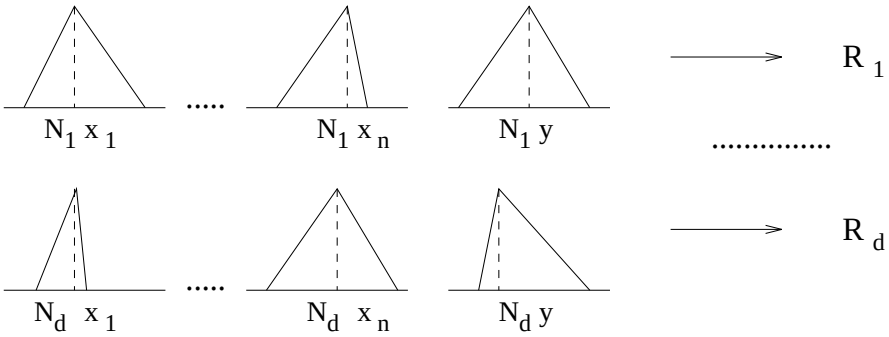
$$A(N_i x) = *(A_1(N_i x_1), \ldots, A_n(N_i y))$$

$$C \sim R_i : \text{IF } x_1 \text{ is } A_1 \text{ and } \ldots \text{and } x_n \text{ is } A_n \text{ THEN } y \text{ is } B$$

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 the rule encoded in C is equal to one of those generated previosly.

The Figure 4 shows graphically a situation where there is interaction between the rule encoded by C an any of the rules generated until now:

The adition of the $LNIR$ to the previous generation process makes two modifications over the fitness landscape to be applied at each algorithm step. The purpose of these are to change the fitness payoff associated to the individuals in



There is interaction between the rule encoded in C and one of the previously generated in the following case:

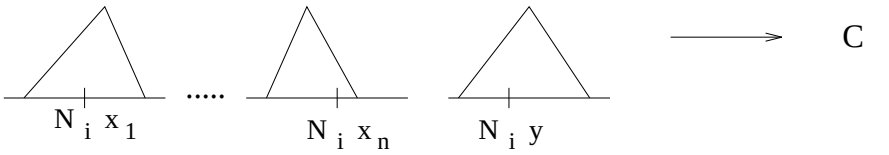


Fig. 4. Interaction between the current rule and the predetermined ones

subsequent runs, encouraging the exploration of new space zones and penalizing it on others in which a niche have been located. These modifications operate in two different levels:

- The covering method removes examples from the training data set, eliminating the payoff associated to the space zones where these examples were located. This constitutes a *high level modification* due to it translate the search focus to other space zone. In this way, it encourage an adequate space exploration.
- When a niche has been located in a space zone but it continue being the most promising one, the new fuzzy rules that are going to be generated in the same zone will interact with the ones generated until now. An adequate interaction rate will be desirable in order to make best use of the advantages of the FLC interpolative reasoning. It is put into effect by using a *niche penalizing function* working in the aforecommented way: when closer is the new rule to the previously generated ones, more penalized it is.
  In this case, this modification constitute a *low level* one because the algorithm continues working in the same space zone but penalyzing the excesive closeness to the niches located in it. It encourage an adequate space exploitation.

Therefore, the final generation process will allow to verify the two following

fundamental aspects:

- The process will ensure fuzzy control rules to be obtained in each space zone in which the control problem is defined, that is, 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 into each one of the niches existing in the solution space. A suitable interaction among the KB fuzzy control rules is so obtained.

An *evaluation function* to the rule R_i, and therefore a fitness function to the associated chromosome C_i is defined as follows:

$$Z(R_i) = \Psi_{E_p}(R_i) \cdot G_w(R_i) \cdot g_n(R_i^-) \cdot LNIR(R_i)$$

with the objective of maximizing the fitness function.

4.1.4 Genetic Operators

Due to the special nature of the chromosomes involved in this generation process, the design of the genetic operators become a main task. There is need of remembering that the individuals encode two diferent information levels, a first part containing the rule linguistic terms (RB information), and a second one coding the meaning associated to these labels (DB information). As there exists a strong relationship between both parts, operators working cooperatively in C_1 and C_2 are required in order to make best use of the representation used.

It can be clearly observed that the existing relationship will present several problems if not handled adequately. For example, modifications in the first chromosome part have to be automatically reflected in the second one. It makes no sense to modify the linguistic term and continue working with the previous label meaning. On the other hand, there is need of developing recombination in a correct way in order to obtain meaningful offsprings.

Taking into account these aspects, several operators belonging to three genetic operator classes are going to be used: *mutation*, *crossover*, and *evolution strategy*.

Mutation: Two different operators are used, each one of them acting on a different chromosome part. A short description of them is given below:

- As C_2 corresponds to the individual representation employed in the free structure generation process, the same mutation developed in this process is performed on it. In this way, Michalewicz's non-uniform mutation operator is employed [18].
  If $C_v^t = (c_1, ..., c_k, ..., c_H)$ 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 $C_v^{t+1} = (c_1, ..., c'_k, ..., c_H)$, with $k \in 1, ..., H$, and

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

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

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

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 dependency with the number of iterations. This property causes this operator to make an uniform search in the initial space when t is small, and a very local one in later stages.

- The mutation operator selected for C_1 is similar to the one proposed by Thrift in [21]. When a mutation on a gene belonging to the first part of the chromosome is going to be performed, a local modification is developed by changing the current linguistic term to the inmediately preceding or subsequent one (the decision is made at random). When the label to be changed is the first or last one in the term set, the only possible change is developed. As it have been commented, a mutation in C_1 provokes a change in C_2. When a linguistic variable changes its value from one term to another, the meaning associated to it is automatically updated in the second chromosome part to the default values in the corresponding primary fuzzy partition.

Crossover: As regards to the recombination process, two different crossover operators are employed depending on the two parents' scope:

- If the rule encoded by both individuals is the same, then the genetic search has located a promising space zone that have to be adequatelly exploitated. This task is developed by applying the max-min-arithmetical crossover operator in C_2 and obviously by maintaining the parent C_1 values in the offspring. This crossover operator is proposed in [12] and works in the way shown above.
  If $C_v^t = (c_1, ..., c_k, ..., c_H)$ and $C_w^t = (c'_1, ..., c'_k, ..., c'_H)$ are to be crossed, the following four offspring are generated

$$C_1^{t+1} = aC_w^t + (1 - a)C_v^t$$
$$C_2^{t+1} = aC_v^t + (1 - a)C_w^t$$
$$C_3^{t+1} \text{ with } c_{3k}^{t+1} = \min\{c_k, c'_k\}$$
$$C_4^{t+1} \text{ with } c_{4k}^{t+1} = \max\{c_k, c'_k\}$$

  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.

– When the parents encode different rules, it makes no sense to apply the previous operator because it will provoke the obtaining of disrupted descendents. This fact is due to the combination of two membership functions associated to different linguistic labels makes the obtaining of two new fuzzy sets not belonging to the intervals of performance determined by the initial fuzzy partition. This second case highly recommend the use of the information encoded by the parents for explorating the search space in order to discover new promising zones. In this way, an standard crossover operator is applied over both parts of the chromosomes. This operator performs as follows: a crossover point *cp* is randomly generated in C_1 and the two parents are crossed at the *cp*-th and $n+1+3\cdot cp$ genes. The crossover is developed this way in both chromosome parts, C_1 and C_2, thereby producing two meaningful descendents.
Let us look at an example in order to clarify the standard crossover application. Since $C_t = (c_1, \ldots, c_{cp}, c_{cp+1}, \ldots, c_{n+1}, a_{c_1}, b_{c_1}, c_{c_1}, \ldots, a_{c_{cp}}, b_{c_{cp}}, c_{c_{cp}}, a_{c_{cp+1}}, b_{c_{cp+1}}, c_{c_{cp+1}}, \ldots, a_{c_{n+1}}, b_{c_{n+1}}, c_{c_{n+1}})$ and $C_t^{'} = (c_1^{'}, \ldots, c_{cp}^{'}, c_{cp+1}^{'}, \ldots, c_{n+1}^{'}, a_{c_1^{'}}, b_{c_1^{'}}, c_{c_1^{'}}, \ldots, a_{c_{cp}^{'}}, b_{c_{cp}^{'}}, c_{c_{cp}^{'}}, a_{c_{cp+1}^{'}}, b_{c_{cp+1}^{'}}, c_{c_{cp+1}^{'}}, \ldots, a_{c_{n+1}^{'}}, b_{c_{n+1}^{'}}, c_{c_{n+1}^{'}})$ the individuals to be crossed at point *cp*, the two resulting offspring are:

$$C_{t+1} = (c_1, \ldots, c_{cp}, c_{cp+1}^{'}, \ldots, c_{n+1}^{'}, a_{c_1}, b_{c_1}, c_{c_1}, \ldots,$$

$$a_{c_{cp}}, b_{c_{cp}}, c_{c_{cp}}, a_{c_{cp+1}^{'}}, b_{c_{cp+1}^{'}}, c_{c_{cp+1}^{'}}, \ldots, a_{c_{n+1}^{'}}, b_{c_{n+1}^{'}}, c_{c_{n+1}^{'}})$$

$$C_{t+1}^{'} = (c_1^{'}, \ldots, c_{cp}^{'}, c_{cp+1}, \ldots, c_{n+1}, a_{c_1^{'}, b_{c_1^{'}}}, c_{c_1^{'}}, \ldots,$$

$$a_{c_{cp}^{'}}, b_{c_{cp}^{'}}, c_{c_{cp}^{'}}, a_{c_{cp}}, b_{c_{cp}}, c_{c_{cp}}, \ldots, a_{c_{n+1}}, b_{c_{n+1}}, c_{c_{n+1}})$$

Hence the complete recombination process will allow GA to follow an adequate exploration-exploitation rate in the genetic search. The expected behavior consists of an initial phase where a high number of standard crossovers and a very small of max-min-arithmetical ones (equal to zero in the great majority of the cases) are developed. The genetic search will perform a wide exploration in this first stage, locating the promising zones and sampling the population individuals at them in several runs. In this moment a new phase begin, characterized by the increasing of the exploitation of these zones and the decreasing of the space exploration. Therefore the number of max-min-arithmetical crossovers rises a lot and the application of the standard crossover decreases. An example of this behavior is shown in section 7 (Figure 8).

Evolution Strategy: The last genetic operator to be applied consists of an *(1+1)-ES*. This optimization technique has been selected and integrated into the genetic recombination process in order to perform a local tuning of the best population individuals (rules) in each run. Each time a GA generation is performed, the ES will be applied over a percentage α of the best different population individuals existing in the current genetic population. In this way, it allows to develop again a strong exploitation over the promising space zones

found in each generation by adjusting the C_2 part values of the chromosomes located at them.

The basis of the ES employed were briefly presented in Section 2.2. Now we are going to describe the adaptation of this algorithm to our problem. As it has been commented previously, the mutation strenght depends directly on the value of the parameter σ, which determines the standard deviation of the normally distributed random variable z_i. In our case, the step size σ can not be a single value because each one of the membership functions encoded in the second part of the chromosome is defined over different universes and so require different order mutations. Therefore, an step size $\sigma_i = \sigma \cdot s_i$ for each component has already been used in the *(1+1)-ES*. Anyway the relations of all σ_i were fixed by the values s_i and only the common factor σ is adapted following the assumptions presented in [1].

Each parent component x_i varying in the interval of performance $[x_i^l, x_i^r]$ will have its own associated step size σ_i with $s_i = \frac{x_i^r - x_i^l}{4}$. Hence when σ takes value 1 at the first ES generation, the obtaining of a big quantity of z_i normal values in the interval $[-\frac{x_i^r - x_i^l}{4}, \frac{x_i^r - x_i^l}{4}]$ is ensured. All these values, as the ones remaining in the intervals $[x_i^l, -\frac{x_i^r - x_i^l}{4}]$ and $[\frac{x_i^r - x_i^l}{4}, x_i^r]$, perform a succesful x_i mutation (that is, the corresponding $x_i + z_i$ lies in the x_i interval of performance). When this value does not belong to the commented interval, the mutated value $x_i^{'}$ is equal to the interval extent, x_i^l or x_i^r, closer to $x_i + z_i$.

The Figure 5 summarizes the application scope of the genetic operators proposed:

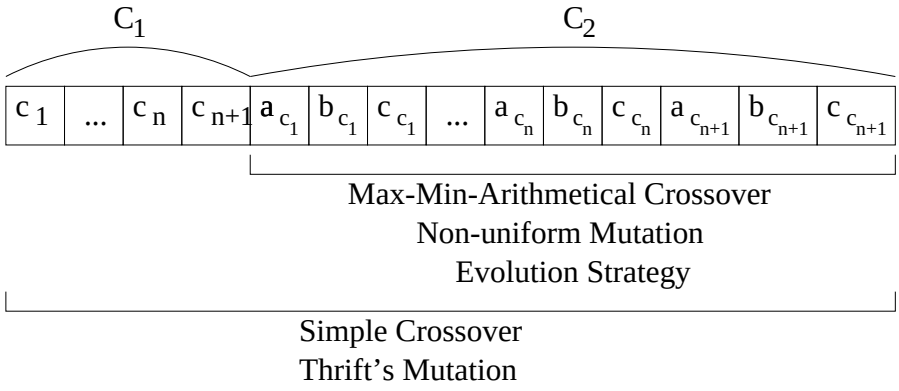


Fig. 5. Generating process genetic representation and operators'application scope

Finally, the following algorithm summarizes the whole process:

1. *Compute the value n_c of individuals belonging to $P(t)$ couples to be crossed taking as base the value of the crossover rate P_c.*
2. **While** *($n_c > 0$)* **do**

(a) *Select at random the parents to be crossed: .*
(b) **If** *($C_1(father) = C_1(mother)$)*
then *maintain C_1 and perform max-min-arithmetical crossover on C_2 for obtaining the two descendents*
else *perform standard crossover on C_1 and C_2.*
(c) $n_c \leftarrow n_c - 1$.

3. *Compute the value n_m of genes to be mutateed taking as base the value of the mutation rate P_m.*

4. **While** *($n_m > 0$)* **do**
(a) *Select at random the parent and gene to be muted.*
(b) **If** *(the gene belongs to C_1)*
then *perform Thrift's mutation on it at C_1 and update the corresponding C_2 piece of chromosome to represent adequatelly the initial mutated gene meaning*
else *perform non-uniform mutation on the C_2 gene.*
(c) $n_m \leftarrow n_m - 1$.

5. *Compute the value $n_{es} \leftarrow \alpha \cdot N$ of individuals to be applied the ES and sort descendentlty the current population, taking into account only the different chromosomes.*

6. **While** *($n_{es} > 0$)* **do**
(a) *Select the next parent to be mutated, begining at the population head.*
(b) *Perform the ES on C_2.*
(c) $n_{es} \leftarrow n_{es} - 1$.

With regards to the *selection procedure*, it is Baker's stochastic universal sampling [2], in which the number of any structure offspring is limited by the floor and ceiling of the expected number of offspring, together with the elitist selection.

4.2 The Covering Method

The covering method was presented in [13]. It is developed as an iterative process that allows to obtain a set of fuzzy rules covering the example set. In each iteration, it runs the generating method, chooses the best chromosome (rule), considers the relative covering value this rule provokes over the training set, and removes from it the examples with a covering value greater than ϵ. The covering method is developed as follows:

1. *Initialization:*
(a) *To introduce k, ω and ϵ.*
(b) *To set the example covering degree $CV[l] \leftarrow 0$, $l = 1, ..., p$.*
(c) *To initialize the final set of rules B^g to empty.*

2. *Over the set of examples* E_p, *to apply the generating method.*
3. *To select the best chromosome* C_r *encoding the fuzzy rule* R_r.
4. *To introduce* R_r *in* R^g.
5. *For every* $e_l \in E_p$ *do*
   (a) $CV[l] \leftarrow CV[l] + R_r(e_l)$,
   (b) *If* $CV[l] \geq \epsilon$ *then remove it from* E_l.
6. *If* $E_p = \emptyset$ *then Stop else return to Step 2.*

Since two similar rules may be obtained, it is necessary to simplify the complete KB obtained from the previous process for deriving the final KB, thereby allowing the system to be controlled.

5 The Genetic Simplification Process

Due to the iterative nature of the genetic generation process, an overlearning phenomenon may appear. This occurs when some examples are covered at a higher degree than the desired one and it makes the obtained RB perform worse. In order to solve this problem and improve its accuracy, it is necessary to simplify the rule set obtained from the previous process for deriving the final RB allowing the system to be controlled.

The simplification process used was proposed in [13]. It is based on a binary coded GA, in which the selection of the individuals is developed using the aforementioned stochastic universal sampling procedure together with an elitist selection scheme, and the recombination is put into effect by using the classical binary multipoint crossover (performed at two points) and uniform mutation operators.

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

$$If\ c_i = 1\ then\ R_i \in B^s\ \ else\ R_i \notin B^s$$

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

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

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

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 variables values are ex^l, and ey^l is the known desired value.

Anyway, there is a need to keep the *control rule completeness* property considered in the previous stage. An FLC must always be able to infer a proper control action for every process state. We will ensure this condition by forcing every example contained in the training set to be covered by the encoded RB at a degree greater than or equal to τ,

$$C_{R(C_j)}(e_l) = \bigcup_{j=1..T} R_j(e_l) \geq \tau, \quad \forall e^l \in E_{TDS} \text{ and } R_j \in R(C_j) \ ,$$

where τ is the minimal training set completeness degree acepted in the simplification process. Usually, τ is less than or equal to ω, 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_{TDS} as

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

The final fitness function penalizing the lack of the completeness property is:

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

6 The Genetic Tuning Process

The genetic tuning process was presented in-depth in [12]. The process is based on the existence of a previous complete KB, that is, an initial DB definition and a RB constituted by m fuzzy control rules.

Each chromosome forming the genetic population will encode a complete KB. Each one of them contains the RB R^s with a different DB associated with it.

The GA designed for the tuning process presents a real coding issue, uses the stochastic universal sampling as selection procedure and Michaelewicz's non-uniform mutation operator. As regards to the crossover operator, the max-min-arithmetical is employed again.

As we commented before, the membership functions are triangular-shaped. Thus, each one of them has an associated parametric representation based on a 3-tuple of real values. Each one of the rules will be encoded in pieces of the chromosome C_{ri}, $i = 1, \ldots, m$, in the following way:

$$C_{ri} = (a_{i1}, b_{i1}, c_{i1}, \ldots, a_{in}, b_{in}, c_{in}, a_i, b_i, c_i)$$

Therefore the complete RB with an associated DB is represented by a complete chromosome C_r:

$$C_r = C_{r1} \ C_{r2} \ \ldots \ C_{rm}$$

As may be seen, each individual in the population represents a complete KB. More concretely, all of them encode the derived system RB R^s and the difference

between them are the fuzzy control rule membership functions, that is, the DB definition.

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, $h = 1 \ldots (n+1) \cdot m \cdot 3$. Each interval of performance will be the interval of adjustment for the correspondent 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, c_{t+1}, c_{t+2}) and the intervals of performance are the following:

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

Figure 6 shows these intervals.

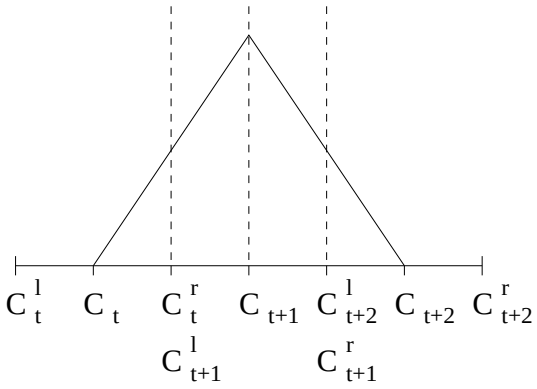


Fig. 6. Membership function and intervals of performance for the tuning process

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

The fitness function of a chromosome is defined by using a training input-output data set, E_{TDS}, and a concrete error measure, the medium square error in our proposal. In this way, the adaptation value associated to an individual is 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 is represented by the following expression:

$$E(C) = \frac{1}{2|E_{TDS}|} \sum_{e_l \in E_{TDS}} (ey^l - S(ex^l))^2$$

7 Application of the learning process to the fuzzy modeling of two three-dimensional mathematical functions

In order to analyze the accuracy of the method proposed, we have selected two n-dimensional mathematical functions for using them 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 an unimodal function while the *generalized Rastrigin function*, F_2, is a strongly multimodal one, as may be observed in their graphical representations (Figure 7).

$$F_1(x_1, x_2) = x_1^2 + x_2^2,$$
$$x_1, x_2 \in [-5, 5], F_1(x_1, x_2) \in [0, 50]$$

$$F_2(x_1, x_2) = x_1^2 + x_2^2 - cos(18x_1) - cos(18x_2),$$
$$x_1, x_2 \in [-1, 1], F_2(x_1, x_2) \in [2, 3.5231]$$

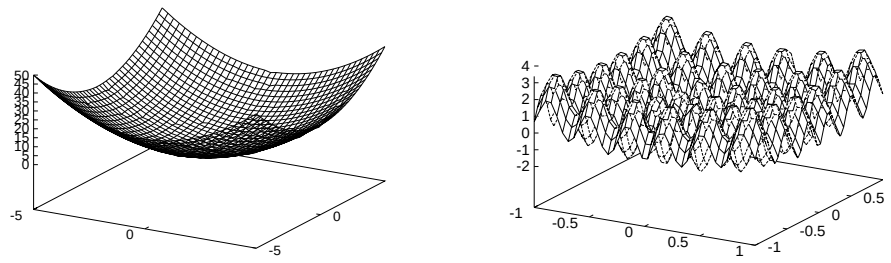


Fig. 7. Graphical representations of F_1 and F_2

Two different ways of fuzzy modeling of these surfaces are going to be compared by using the following both design methods:

1. a two-stage method based on obtaining a complete KB by deriving the RB by means of the widely employed Wang and Mendel's (WM) method [22] and defining the DB by means of the genetic tuning method constituting the third stage of the method proposed, and
2. the GFS design method proposed in this paper.

For each one 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 by taking 41 values for each one of the two state variables considered to be uniformly distributed in their respective intervals.

Two other data sets have been generated for their 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 corresponding training set one, a ten percent to be precise. The data are obtained by generating at random the state variable values into 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 are used to measure the accuracy of the FLCs designed by computing the medium square 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 associated to the control one) formed by *seven linguistic terms* with triangular-shaped fuzzy sets giving meaning to them (as shown in Figure 2), and the adequate scaling factors to traslate 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 $\tau = \{0.25, 0.5\}$. It leads to an overall of 4 runs per function. With respect to the remaining parameters, the t-norm $*$ used in the rule generation process is the Minimum, the generating process GAs runs over 50 generations, the ES is applied until there is no improvement in 25 generations over a percentage $\alpha = 20\%$ of the population individuals (the parameter c of the 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 non-uniform 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 indiviual). The max-min-aritmethical crossover parameter a takes the value 0.35.

Finally, as regards to the FLC reasoning method employed, we have selected the *Minimum t-norm* playing the role of the implication and conjunctive operators, and the *Center of Gravity weighted by the matching* strategy acting as the defuzzification operator [5].

The results obtained in the different experiments are collected in the following tables where $\#R$ stands for the number of rules of the corresponding KB. Each table last row shows the results obtained by the WM-based method in the corresponding function. Analyzing these results, the good behavior presented by the proposed method can be observed. All the GFSs designed using it are more accurated in a high degree than the ones based on the WM RB generation method in the fuzzy modeling of both functions.

Finally, a graphical representation of the behavior of the crossover operators used is shown in Figure 8. As it was commented in section 4.1.4, the expected behavior consists of an initial phase where a high number of standard crossovers and a very small of max-min-arithmetical ones are developed, and a second phase where the number of max-min-arithmetical crossovers rises a lot and the application of the standard crossover decreases. The figure, drawn making use of the data collected in the run 56 of the experiment developed with parameters $\epsilon = 1.5$ and $\tau = 0.5$ (the FLC obtained presenting best behavior in the fuzzy

Table 1. Results obtained in the fuzzy modeling of function F_1

Parameters			Generation		Simplification		Tuning
ϵ	ω	τ	#R	SE	#R	SE	SE
1.0	0.05	0.25	70	1.992471	58	1.630508	0.698604
1.0	0.05	0.5	70	1.992471	63	1.770926	0.663183
1.5	0.05	0.25	98	2.411402	67	1.779137	0.696869
1.5	0.05	0.5	98	2.411402	73	2.130197	1.118251
W	M		49	4.651811			0.950740

Table 2. Results obtained in the fuzzy modeling of function F_2

Parameters			Generation		Simplification		Tuning
ϵ	ω	τ	#R	SE	#R	SE	SE
1.0	0.05	0.25	251	0.288433	190	0.244648	0.214298
1.0	0.05	0.5	251	0.288433	206	0.266778	0.230763
1.5	0.05	0.25	346	0.268026	232	0.213960	0.195233
1.5	0.05	0.5	346	0.268026	253	0.232196	0.210177
W	M		49	2.094091			1.218088

modeling of function F_2), shows clearly this behavior.

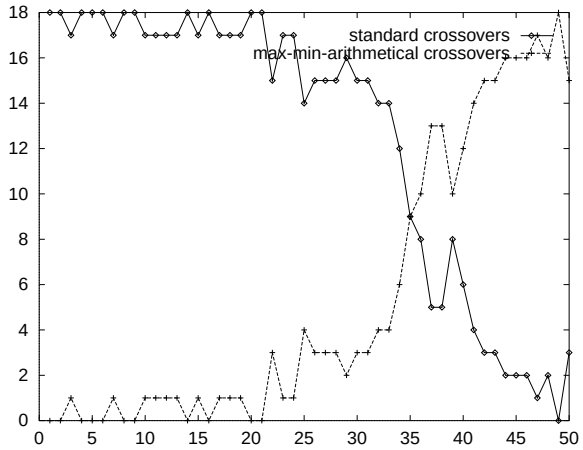


Fig. 8. Number of crossovers per generation in a run of one experiment developed

8 Concluding Remarks

A method for designing GFSs by learning the KB from examples combining a hybrid GA-ES generation and two GA-based simplification and tuning processes has been presented. Its performance have been shown by applying it to the fuzzy modeling of two three-dimensional control surfaces and compared with another method based on the WM process obtaining good results.

References

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

[2] J.E. Baker, Reducing bias and inefficiency in the selection algorithm, Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum, Hillsdale, NJ (1987) 14-21.

[3] D. Beasly, D.R. Bull, R.R. Martin, A sequential niche technique for multimodal function optimization, Evolutionary Computation 1(2) (1993) 101-125.

[4] O. Cordón, F. Herrera, A general study on genetic fuzzy systems, in: J. Periaux, G. Winter, M. Galán, P. Cuesta, Eds., Genetic Algorithms in Engineering and Computer Science (John Wiley and Sons, 1995) 33-57.

[5] O. Cordón, F. Herrera, A. Peregrín, Applicability of the fuzzy operators in the design of fuzzy logic controllers, Fuzzy Sets and Systems (1996) to appear.

[6] O. Cordón, F. Herrera, M. Lozano, A classified review on the combination fuzzy logic-genetic algorithms bibliography: 1989-1995, in: E. Sanchez, T. Shibata, L. Zadeh, Eds., Genetic Algorithms and Fuzzy Logic Systems. Soft Computing Perspectives (World Scientific, 1996).

[7] K. Deb, D.E. Goldberg, An investigation of niche and species formation in genetic function optimization. Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum, Hillsdale, NJ, (1989) 42-50.

[8] D. Driankov, H. Hellendoorn, M. Reinfrank, An introduction to fuzzy control (Springer-Verlag, 1993).

[9] D.B. Fogel, Evolutionary computation. Toward a new philosophy of machine intelligence (IEEE Press, 1995).

[10] D.E. Goldberg, Genetic algorithms in search, optimization, and machine learning (Addison-Wesley, New York, 1989).

[11] A. González, R. Pérez, Completeness and consistency conditions for learning fuzzy rules. Technical Report DECSAI-95103, Dept. of Computer Science and Artificial Intelligence, University of Granada, 18071 Granada, Spain (November 1995).

[12] F. Herrera, M. Lozano, J.L. Verdegay, Tuning fuzzy controllers by genetic algorithms, International Journal of Approximate Reasoning 12 (1995) 299-315.

[13] F. Herrera, M. Lozano, J.L. Verdegay, A learning process for fuzzy control rules using genetic algorithms. Technical Report DECSAI–95108, Dept. of Computer Science and A.I., University of Granada, Spain (February 1995).

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

[15] J.H. Holland, Adaptation in natural and artificial systems (Ann arbor: The University of Michigan Press, 1975) (The MIT Press, London, 1992).
[16] C.C. Lee, Fuzzy logic in control systems: fuzzy logic controller - parts I and II, IEEE Transactions on Systems, Man, and Cybernetics 20 (1990) 404-435.
[17] E.H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, International Journal of Man-Machine Studies 7 (1975) 1-13.
[18] Z. Michalewicz, Genetic algorithms + data structures = evolution programs (Springer-Verlag, 1992).
[19] H.-P. Schwefel, Evolution and optimum seeking. Sixth-Generation Computer Technology Series (John Wiley and Sons, 1995).
[20] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Transations on Systems, Man, and Cybernetics 15 (1985) 116-132.
[21] P. Thrift, Fuzzy logic synthesis with genetic algorithms. Proceedings of Fourth International Conference on Genetic Algorithms (1991) 509-513.
[22] L.X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Transactions on Systems, Man, and Cybernetics 22 (1992) 1414-1427.