NEAR EAST UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

GENETIC FUZZY SYSTEMS

COM 400 GRADUATION PROJECT

STUDENT : AZiZ KARACA (980131)

SUPERVISOR : ASSOC. PROF. DR. RAHIB ABIYEV

Nicosia 2002

TABLE OF CONTENTS

ACKNOWLEDGEMENT

First of all I would like to present my thanks my Graduation Project Supervisor Mr. Rahib Abiyev. He guided me with a keen interest and helped me by all means. Thanks for your continual support, Mr. Abiyev.

Also I would like to thank to all my teachers in Near East University, including Faculty of Engineering Dean Prof. Dr. Fahreddin Mamedov, Deparment of Computer Engineering Chairman Assist. Prof. Dr. Adnan Khasman, Student Advisors Mr. Faiq Ridvan and Mr. Tayseer Alshanableh and all the Staff of the Faculty of Engineering, and special thanks to the Vice-President of Near East University Assoc. Prof. Dr. Şenol Bektaş for given me the opportunity to experience this remerkable institute. They explained my questions patiently, they always help me a lot either in my study or my life. I asked many questions a lot of subjects and they always answered my questions and in detail.

And, I also want to thanks my friends Dilek Öztürk, Aysenur Ortaç and Bülent Korkmaz for providing both moral and financial support that make my four years TRNC full of fun.

Finally, I want to thank to my family without their endless support and love for me.

ABSTRACT

The automatic definition of a fuzzy system can be considered in many cases as an optimization or search process. Genetic Algorithms (GAs) are the best known and most widely used global search technique with an ability to explore and exploit a given operating space using available performance measures. GAs are known to be capable of finding near optimal solutions in complex search spaces. A priori knowledge of a fuzzy system may come in the form of known linguistic variables, fuzzy membership functions parameters, fuzzy rules, number of rules, etc. The generic code structure and independent performance features of GAs make them suitable candidates for incorporating a priori knowledge.

The search capabilities and ability for incorporating a priori knowledge have extended the use of GAs in the development of a wide range of methods for designing fuzzy systems over the last few years. Systems applying these design approaches have received the general name of Genetic Fuzzy Systems (GFSs).

In this project the different Genetic Algorithms approaches, for learning Fuzzy rule-based systems are considered. The synthesis processes of Fuzzy logic controller by using Genetic Algorithm are described.

INTRODUCTION

In a very broad sense, a Fuzzy System (FS) is any Fuzzy Logic-Based System, where Fuzzy Logic can be used either as the basis for the representation of different forms of system knowledge, or to model the interactions and relationships among the system variables. FSs have proven to be an important tool for modeling complex systems, in which, due to the complexity or the imprecision, classical tools are unsuccessful.

Genetic Algorithms (GAs) are search algorithms that use operations found in natural genetics to guide the trek through a search space. GAs are theoretically and empirically provide robust search capabilities in complex spaces, offering a valid approach to problems requiring efficient and effective searching.

The automatic definition of an FS can be considered in man cases as an optimization or search process. GAs are the best known and most widely used global search technique with an ability to explore and exploit a given operating space using available performance measures. GAs are known to be capable of finding near optimal solutions ın complex search spaces. A priori knowledge may be in the form of linguistic variables, fuzzy membership function parameters, fuzzy rules, number of rules, etc. The generic code structure and independent performance features of GAs make them suitable candidates for incorporating a priori knowledge. These advantages have extended the use of Gas in the development of a wide range of approaches for designing fuzzy systems over the last few years. Figure 1 shows this idea.

Fuzzy Rule Based Systems (FRBSs), the most extended FS model to which the most successful application of FSs belong, the fuzzy logic controllers (FLCs), which have been and are used in many real-world control problems.

Fuzzy Logic Controllers (FLCs) are being widely and successfully applied in different areas. Fuzzy Logic Controllers can be considered as knowledge-based systems, incorporating human knowledge into their Knowledge Base through Fuzzy Rules and Fuzzy Membership Functions (among other information elements). The definition of these Fuzzy Rules and Fuzzy Membership Functions is actually affected by subjective decisions, having a great influence over the performance of the Fuzzy Controller. From this point of view , FLCs can be interpreted as a particular type of real time expert systems. A second interpretation more adequate for the analysis of the control properties of the FLC is to think about FLCs as non-linear , time-invariant control laws. In addition, recent works have demonstrated the ability of Fuzzy Controllers to approximate continuous functions on a compact set with an arbitrary degree of precision; different kinds of FLCs are universal approximators. Combining ideas related to these different interpretations, some efforts have been made to obtain an improvement in system performance (a better approximation to an optimal controller, with a certain performance criterion) by incorporating learning mechanisms to modify predefined rules and/or membership functions, represented as parameterized expressions. The main goal will be to combine the ability to incorporate experts' knowledge with a knowledge-based point of view (Knowledge engineering), with the possibility of tuning by applying learning (Machine learning) or adaptation (Adaptive control) techniques through the analytical representation of the FLC. Ideas arising out of two main areas have been applied with this aim: ideas coming from Artificial Neural Networks (ANNs) and from Genetic Algorithms (GAs).

When applying ideas coming from ANNs, the learning techniques use basically the topological properties of IRv (e.g., the properties of gradient), where IRn represents the space of parameters of the controller. On the other hand, GAs are probabilistic search and optimization procedures based on natural genetics, working with finite strings of bits that represent the set of parameters of the problem, and with a fitness function to evaluate each one of these strings. The finite strings used by GAs may be considered as a representation of elements of IRn , but usually, the learning mechanisms make no use of the topological properties of this space of parameters.

The application of Genetic Algorithms to FLCs with a learning purpose, has produced some interesting works. This chapter presents an overview of the area and a deeper analysis of two different works applying Genetic Algorithms to Fuzzy Logic Controllers whose Rule Base is defined through a set of Fuzzy Rules. The use of a set of Fuzzy Rules (and not a Fuzzy Relational Matrix or a Fuzzy Decision Table) is adapted to the application to complex control problems containing a large number of variables, since it reduces the dimensionality of the Knowledge Base for this kind of system. The first approach uses the Knowledge Base of the system as the population of the genetic system (a single rule containing the description of the corresponding Fuzzy Sets is an individual of the population). The second one uses the Knowledge Base (containing a set of Fuzzy Rules and a set of Membership Functions) as the individual of the genetic system, working with a population of Fuzzy Controllers. Each system has its own scheme to code the information evolved through the GA, and its evolution operators.

As is well known, the Knowledge Base (KB) of an FRBS is comprised of two components, a Data Base (DB), containing the definitions of the scaling factors and the membership functions of the fuzzy sets specifying the meaning of the linguistic terms, and a Rule Base (RB), constituted by the collection of fuzzy rules. Figure 2 shows the structure of a KB integrated in an FS with fuzzyfication modules, as used in fuzzy control.

GAs may be applied to adapting/learning the DB and/or RB of an FRBS. This tutorial will summarize and analyze the GFs, paying a special attention to FRBSs incorporating tuning/learning through GAs.

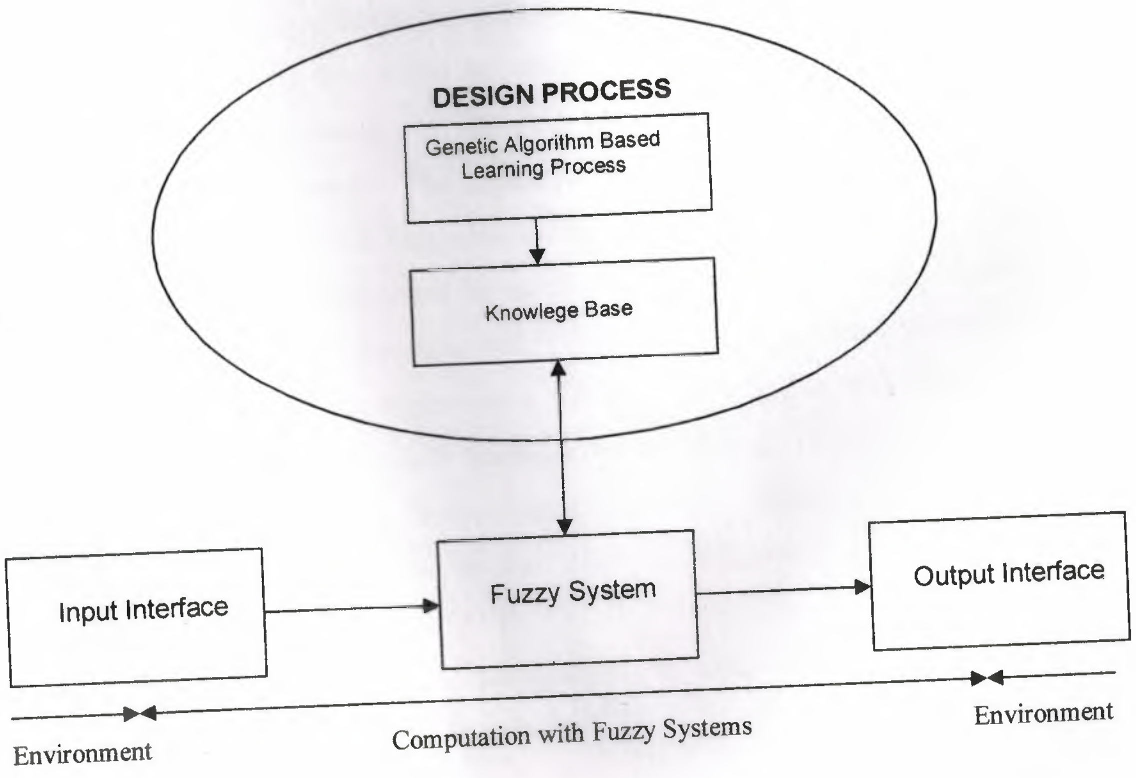


Figure 1: Genetic Fuzzy Systems

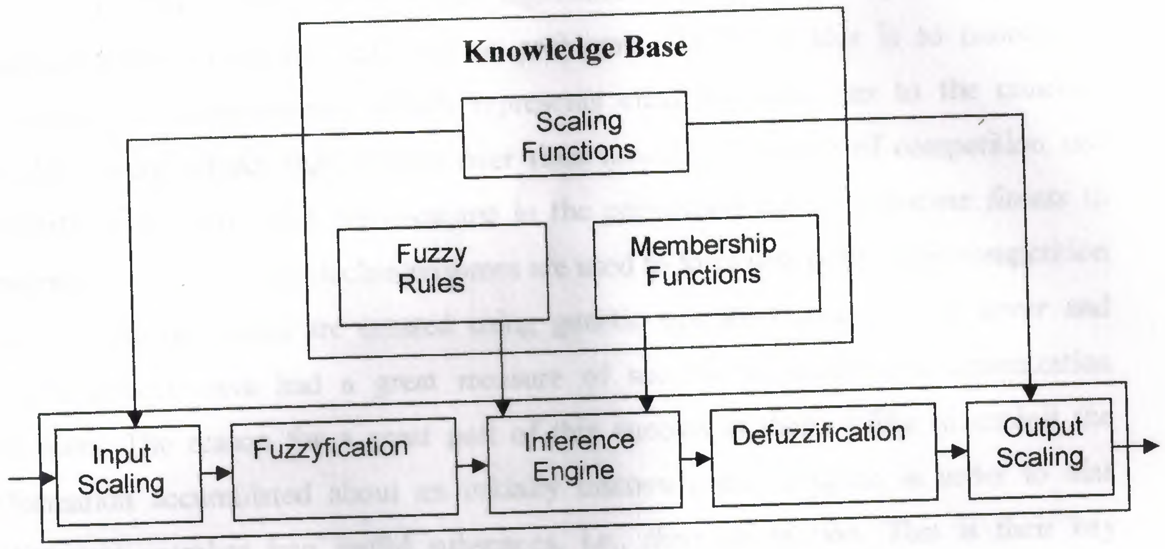


Figure 2: Structured Knowledge Base

1 GENETIC ALGORITHMS

GAs are general purpose search algorithms which use principles inspired by natural genetics to evolve solutions to problems. The basic idea is to maintain a population of chromosomes, which represents candidate solutions to the concrete problem being solved, that evolves over time through a process of competition and controlled variation. Each chromosome in the population has an associate *fitness* to determine *(selection)* which chromosomes are used to form new ones in the competition process. The new ones are created using genetic operators such as cros *sover* and *mutation.* GAs have had a great measure of success in search and optimization problems. The reason for a great part of this success is their ability to exploit the information accumulated about an initially unknown search space in order to bias subsequent searches into useful subspaces, i.e., *their adaptation.* This is their key feature, particularly in large, complex, and poorly understood search spaces, where classical search tools (enumerative, heuristic...) are inappropriate, offering a valid approach to problems requiring efficient and effective search techniques.

A GA starts off with a population of randomly generated *chromosomes,* and advances toward better *chromosomes* by applying genetic operators modeled on the genetic processes occurring in nature. The population undergoes evolution in a form of natural selection. During successive iterations, called *generations,* chromosomes in the population are rated for their adaptation as solutions, and on the the basis of these evaluations, a new population of chromosomes is formed using a selection mechanism and specific genetic operators such as crossover and mutation. An *evaluation* or *fitness function (f)* must be devised for each problem to be solved. Given a particular chromosome, a possible solution, the fitness function returns a single numerical fitness, which is supposed to be proportional to the utility or adaptation of the solution represented by that chromosome.

Although there are many possible variants of the basic GA, the fundamental underlying mechanism consists of three operations:

1. evaluation of individual fitness,

2. formation of a gene pool (intermediate population) through selection mechanism, and

3. recombination through crossover and mutation operators.

Figure 3 shows the structure of a basic GA, where *P(t)* denotes the population at generation *t*.

```
Procedure Genetic Algorithm
begin (1)
    t = 0;
    initialize P(t);
    evaluate P(t);
    While (Not termination-condition) do
    Begin (2)
        t = t + 1;
            select P(t) from P(t – 1);
            recombine P(t);
            evaluate P(t);
    end (2)
end (1)
```

Figure 3: Structure of a GA

1.1 Main characteristics of GAs

GAs may deal successfully with a wide range of problem areas. The main reasons for this success are: 1) GAs can solve hard problems quickly and reliably, 2) GAs are easy to interface to existing simulations and models, 3) GAs are extendible and 4) GAs are easy to hybridize. All these reasons may be summed up in only one: GAs are *robust*. GAs are more powerful in difficult environments where space is usually large, discontinuous, complex and poorly understood. They are not guaranteed to find the global optimum solution to a problem, but they are generally good at finding acceptably good solutions to problems acceptably quickly. These reasons have been behind the fact that, during the last few years, GA applications have grown enormously in many fields.

The basic principles of GAs were first laid down rigorously by Holland , and are well described in many books. It is generally accepted that the application of a GA to solve a problem must take into account the following five components:

1. *A genetic representation of solutions to the problem,*
2. *a way to create an initial population of solutions*
3. *an evaluation function which gives the fitness of each chromosome,*

4. *genetic operators that alter the genetic composition of offspring during reproduction, and*
5. *values for the parameters that the GA uses (population size, probabilities of applying genetic operators, etc.).*

Some of these components will be de scribed or analyzed below based on the ideas contained in previous paragraphs.

1.1.1 Representation and evaluation of solutions

Representation is a key issue when applying GAs because they directly manipulate a coded representation of the problem and, consequently, the representation schema can severely limit the window through which a genetic system observes its world.

Additionally, the search process involved when applying GAs to solve a problem is driven or biased by the concept of utility or adaptation of the individuals as solutions to that problem. A fitness function must be devised for each problem in such a way that given a particular chromosome, a solution, the fitness function returns a single numerical fitness, which is supposed to be proportional to (to evaluate) the utility or adaptation of the individual which that chromosome represents.

1.1.2 Selection Mechanism

Let's consider *p,* a population with chromosomes C1, ..., *CN.* The selection mechanism produces an intermediate population, *P',* with copies of chromosomes in *P* . The number of copies received from each chromosome depends on its fitness (the evaluation described in the previous paragraph), chromosomes with higher fitness usually have a greater chance of contributing copies to *P'* .There are a number of ways of making this selection. We might view the population as mapping onto a roulette wheel, where each chromosome is represented by a space that proportionally corresponds to its fitness. By repeatedly spinning the roulette wheel, chromosomes are chosen using "stochastic sampling with replacement" to fill the intermediate population. The selection proceure called *stochastic universal sampling* is one of the most efficient, where the number of offspring of any structure is bound, by the floor and ceiling of the expected number of offspring.

1.1.3 Recombination through Crossover and Mutation

After selection has been carried out the construction of the intermediate population is complete, then the genetic operators, crossover and mutation, can be applied.

Recombination through Crossover. The crossover operator is a method information between cromosomes; it combines the features of two parent cromosomes to form two offsprings, with the possibility that the offsprings generated through recombination are better adapted than their parents. The crossover operator is not usually applied to all pairs of chromosomes in the intermediate population. A random choice is made, where the likelihood of crossover being applied depends on probability defined by a crossover rate, the crossover probability *(Pc)*. The crossover operator plays a central role in GAs, in fact it may be considered to be one of the algorithm's defining characteristics, and it is one of the components to be borne in mind to improve the GA behavior. Definitions for this operator (and the next one) are highly dependent on the particular representation chosen.

Mutation. A mutation operator arbitrarily alters one or more components of a selected structure so as to increase the structural variability of the population. Each position of each solution vector in the population undergoes a random change according to a probability defined by a mutation rate, the mutation probability *(Pm)*.

We should point out that after crossover and mutation, an additional selection strategy, called *elitist strategy,* may be adopted to make sure that the best performing chromosome always survives intact from one generation to the next. This is necessary since it is possible that the best chromosome disappears thanks to crossover or mutation.

1.2 Learning with GAs

Although GAs are not learning algorithms, they may offer a powerful and domain independent search method for a variety of learning tasks. In fact, there has been a good deal of interest in using GAs for machine learning problems.

Three alternative approaches, in which GAs have been applied to learning processes, have been proposed, the Michigan, the Pittsburgh, and the Iterative Rule Learning (IRL) approaches. In the first one, the chromosomes correspond to classifier rules which are evolved as a whole, whereas in the Pittsburgh approach, each chromosome encodes a complete set of classifiers. In the IRL approach each chromosome represents only one rule learning, but contrary to the first, only the best individual is considered as the solution, discarding the remaining chromosomes in the population.

1.2.1 The Michigan approach

The chromosomes are individual rules and a rule set is represented by the entire population. The collection of rules are modified over time via interaction with the environment. This model maintains the population of classifiers with credit assignment, rule discovery and genetic operations applied at the level of the individual rule.

There is a considerable variety in the structural and functional details of this model. The prototype organization is composed of three parts:

1. the *performance system* that interacts with the environment and contains the rule base and the production system,
2. the *credit assignment system* or credit apportionment system, developing learning by the modification and adjustment of conflictesolution parameters of the classifier (rule) set, their strengths; Holland's Bucket Brigade is one example of this, and
3. the *classifier discovery system* that generates new classifiers, rules, from a classifier set by means of GAs.

A genetic learning process based on the Michigan approach receives the name of Classifier System (CS). The prototypical organization of a CS is illustrated on Figure 4.

1.2.2 The Pittsburgh approach

Each chromosome encodes a whole RB or KB. Crossover serves to provide a new combination of rules and mutation provides new rules. In some cases, variable-length rule bases are used, employing modified genetic operators for dealing with these variable-length and position independent genomes.

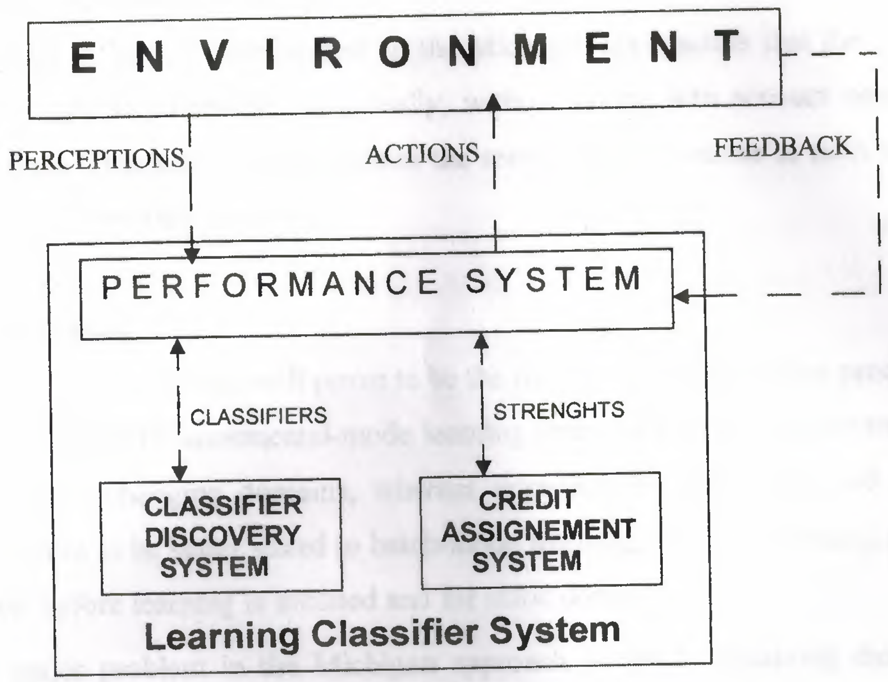


Figure 4: Organization of a classifier system

1.2.3 Iterative Rule Learning approach

In this latter model, as in the Michigan one, each chromosome in the population represents a single rule, but contrary to the Michigan one, only the best individual is considered to form part of the solution, discarding the remaining chromosomes in the population. Therefore, in the iterative model, the GA provides a partial solution to the problem of learning. In order to obtain a set of rules, which will be a true solution to the problem, the GA (with a structure similar to the one described in Figure 3) has to be placed within an iterative scheme similar to the following:

1. Use a GA to obtain a rule for the system.
2. Incorporate the rule into the final set of rules.
3. Penalize this rule.
4. If the set of rules obtained till now is adequate to be a solution to the problem, the system ends up returning the set of rules as the solution. Otherwise return to step 1.

A very easy way to penalize the rules already obtained, and thus be able to learn new rules when performing inductive learning, consists of eliminating from the training set all those examples that are covered by the set of rules obtained previously.

This way of learning is to allow "niches" and " species" formation. Species formation seems particularly appealing for concept learning, considering the process as the learning of multimodal concepts.

The main difference with respect to the Michigan approach is that the fitness of each chromosome is computed individually, without taking into account cooperation with other ones. This substantially reduces the search space, because in each sequence of iterations only one rule is searched.

1.2.4 Conclusions

The Michigan approach will prove to be the most useful in an on-line process. It is more flexible to handle incremental-mode learning (training instances arrive over time) and dynamically changing domains, whereas approach the Pittsburgh and the IRL approaches seem to be better suited to batch-mode learning, where all training instances are available before learning is initiated and for static domains.

The major problem in the Michigan approach is that of resolving the conflict between the individual and collective interests of classifiers within the system. The ultimate aim of a learning classifier system is to evolve a set of co-adapted rules which act together in solving some problems. In a Michigan style system, with selection and replacement at the level of the individual rule, rules which cooperate to effect good actions and receive payoff also compete with each other under the action of the GA.

This conflict between individual and collective interests of individual classifiers does not arise with Pittsburgh-style classifier systems, since reproductive competition occurs between complete rule sets rather than individual rules. However , maintenance and evaluation of a population of complete rule-sets in Pittsburgh-style systems can often lead to a much greater computational burden (in terms of both memory an processing time). Therefore, problems with the Pittsburgh approach have proven to be, at least, equally as challenging. Although the approach avoids the problem of explicit competition between classifiers, large amounts of computing resources are required to evaluate a complete population of rule-sets.

When compared with the latter , the advantage of the IRL approach is that, in the first stage space it considerably reduces the search because it looks for only one rule in each sequence of iterations, although this approach also implies a great computational burden.

On the other hand, GAs are also used for refining parameters in other learning approaches, as is done using GAs for determining weights in a neural network.

2 GENETIC ALGORITHMS IN FUZZY CONTROL SYSTEMS

The application of Genetic Algorithms to FLCs with a learning purpose, has produced some interesting works. This chapter presents an overview of the area and a deeper analysis of two different works applying Genetic Algorithms to Fuzzy Logic Controllers whose Rule Base is defined through a set of Fuzzy Rules. The use of a set of Fuzzy Rules (and not a Fuzzy Relational Matrix or a Fuzzy Decision Table) is adapted to the application to complex control problems containing a large number of variables, since it reduces the dimensionality of the Knowledge Base for this kind of system. The first approach uses the Knowledge Base of the system as the population of the genetic system (a single rule containing the description of the corresponding Fuzzy Sets is an individual of the population). The second one uses the Knowledge Base (containing a set of Fuzzy Rules and a set of Membership Functions) as the individual of the genetic system, working with a population of Fuzzy Controllers. Each system has its own scheme to code the information evolved through the GA, and its evolution operators.

2.1 Fuzzy Logic Controllers

It will be first presented the concept of a fuzzy set and other related ideas as a previous knowledge to introduce Fuzzy Logic Controllers.

A set may be defined using different methods: enumerating their elements, defining a condition that separates the elements belonging to the set, from the remaining elements, using a characteristic function that takes value 1 for all the elements belonging to the set, and value 0 otherwise, etc. When using a characteristic function, a set might be defined as a function from the universe *(U)* to the set $\{0,1\}^2$. A generalization of this definition based on a characteristic function, obtained by allowing values from the whole interval [0, 1], will produce a new type of set that will be called *fuzzy set*.

A *fuzzy set F* in a universe of discourse *X* is characterized by a membership function μ_F that takes values within [0 , 1] .

$$\mu_F : X \rightarrow [0,1]$$

A point-wise fuzzy set or a singleton, is a fuzzy set that is reduced to a single element with membership function 1. The support of a fuzzy set F is the crisp set containing those elements of X, with a membership function μ_F greater than 0. The α-cut of a fuzzy set F is the crisp set containing those elements of X, with a membership function μ_F greater than or equal to α. A fuzzy set A is convex if and only if,

$$\forall x,y \in X,\ \forall \lambda \in [0, 1]: \mu_a(\lambda x + (1-\lambda) y) \geq \min(\mu_a(x), \mu_a(y)).$$

The use of fuzzy sets provides a basis for a systematic way of manipulating vague and imprecise concepts.

A fuzzy set may be defined in a discrete universe of discourse or in a continuous universe of discourse, and in both cases there are different ways of representing the fuzzy set. Fuzzy sets defined in a discrete universe of discourse are usually described by pairs composed of an element $(x \in X)$ and its corresponding membership function $(\mu_F(x))$, this method may be considered as the equivalent to an enumeration of a discrete crisp set. When the universe of discourse is continuous, different representations of a fuzzy set are applied. The first method is the use of parameterized membership functions (triangular, trapezoidal, Gaussian, sigmoidal and other types of parameterized fuzzy sets are used), using the parameters as representation. The second method is the use of the representation of discrete fuzzy sets, after discretizing the continuous fuzzy set. The third method is the use of several α-cuts (occasionally referred to as h-levels or horizontal levels) of the fuzzy set, this method is a sort of discretization but on the values of the function not on the variables.

The basic operations defined for crisp sets (intersection, union and complementation) may be generalized for fuzzy sets. The intersection of two fuzzy sets is defined through any *t-norm*,

$$\mu_{a\cap B}(x) = \mu_a(x)\ T\ \mu_B(x).$$

A t-norm is a function of two arguments, nondecreasing in each argument, commutative, associative, and satisfying the boundary conditions x T 0 = 0 and x T 1 = x. The union of fuzzy sets is defined by any *s-norm,* called *t-conorm* too,

$$\mu_{aUB}(x) = \mu_a(x)\ S\ \mu_b(x).$$

An s-norm is a function of two arguments, nondecreasing in each argument, commutative, associative, and satisfying the boundary conditions x S 0 = x and x S 1 = 1. The complement of a fuzzy set is usually obtained as $\mu_a(x) = 1 - \mu_a(x)$ but may be defined using different expressions. A complement operation c should satisfy at least two conditions: $c(O) = 1$, $c(c(a)) = a$, and $a < b$ *implies* $c(a) > c(b)$. Other mathematical concepts may be generalized for fuzzy sets through the extension principle, that in a simple expression may be defined as follows: if U and V are two universes of discourse and f is a mapping from U to V, for a certain fuzzy set A in U the extension principle defines a fuzzy set $B(f(A))$ in V by

$$\mu_b(v) = \sup_{u \in f^{-1}(v)} \{\mu_a(U)\}.$$

This expression may be applied to generalize fuzzy sets, arithmetical operations or any other mathematical concept defined as a mapping.

A fuzzy rule based system is characterized by a set of rules and by the definitions of some concepts related to information processing: the aggregation operators, the fuzzy connectives and the inference method. Rules are defined by their antecedents and consequents. Antecedents, and frequently consequents, are associated with fuzzy concepts:

R1 : if $x1$ is A11 and ... and xm is $Am1$ then y is $B1$,

. . .

Rn : if $x1$ is $A1n$ and ... and xm is Amn then y is $B1$.

Occasionally consequents are analytical functions of the input variables:

R1 : if $x1$ is A11 and ... and xm is $Am1$ then $y = f1(x1, \dots, xm)$,

. . .

Rn : if $x1$ is $A1n$ and ... and xm is Amn then $y = fn(x1, \dots, xm)$.

In both cases xi are input variables, Aij are fuzzy sets related to input variables, y is the output variable, Bk are fuzzy sets related to the output variable, and fl are functions of the input variables

$$fl(x_1, \dots, xm) = \alpha O1 + \alpha 1 1x1 + \dots + \alpha m1xm.$$

The fuzzy connective *and*, between fuzzy concepts, is usually implemented through the product or the minimum operators (any t-norm may be used as the *and* connective). Systems using the first type of rule are usually called Mamdani type controllers, while those that use the second type of rule are usually named TSK (Takagi, Sugeno and Kang) fuzzy systems.

A fuzzy relationship over the collections of fuzzy sets *A1, ..., Aml B,* is a fuzzy subset over their Cartesian product, *A1* x ...x *Am* x *B.* In a fuzzy relationship, each element of the Cartesian product *(A1i, ..., Amj , Bk),* has a grade of membership to the Relation, represented by μR *(A1i, ..., Amj , Bk)* and taking values within the interval [0, 1]. Table 1 shows the tabular representation of a fuzzy relation for an FLC with one input *(X)* and one output *(Y)* variable, with three fuzzy sets (A*11* , *A12, A13)* related to the input variable and four fuzzy sets related to the output variable *(B1, B2, B3, B4).*

Table 1 A fuzzy relation R.

R	B1	B2	B3	B4
A11	0.5	0.8	0.2	0.0
A12	0.0	0.3	1.0	0.1
A13	0.0	0.0	0.3	1.0

The complete behavior of a fuzzy system may be characterized by a fuzzy relation that is the combination of the fuzzy relations defined by each element of the rule set.

This combination may be represented through the connective *also,*

$$R = \text{also}\,(R1, \ldots, Rn),$$

where also is usually implemented with the maximum operator (any t-conorm or snorm may be used as the *also* connective), generating a fuzzy output. This fuzzy output is a fuzzy subset of Y , from which a non-fuzzy output is usually obtained (deffuzification). Two deffuzification methods often used are the Center of Area (COA) method and the Mean of Maxima (MOM) method. The non-fuzzy output obtained with these two methods has the following expressions:

$$y = \frac{\int_Y \mu_o(y)\, y\, dy}{\int_Y \mu_o(y)\, y\, dy}\ \text{(COA)}, \quad y = \frac{\int_{Y^*} y\, dy}{\int_{Y^*} dy} \qquad (MOM)$$

Where Y is the universe of discourse of the output variable, μ_o *(y)* is the membership value of the output value *y* to the fuzzy output, and $Y_\bullet$ is the α-cut of the fuzzy output, with α equal to the maximum of $\mu_o(y)$.

When using consequences that are functions of the input variables, the *also* connective is implemented as a weighted sum of rules output, producing, then, a numerical output,

$$y = \frac{\sum_n \mu_i y_i}{\sum_n \mu_i},$$

where μ_i is the truth value of the antecedent of rule *i,* and *yi* is the output of rule *i.* In addition to the representation of the behavior of the fuzzy system through a fuzzy relation or with a set of fuzzy rules, a third possibility is the representation using a fuzzy decision table. A fuzzy decision table represents a special case of a crisp relation (the ordinary type of relations we are familiar with) defined over the collections of fuzzy sets corresponding to the input and output variables. Table 2 shows a fuzzy decision table for an FLC with two input *(X1,X2)* and one output *(Y)* variable, with three fuzzy sets (A11 , A12, A13; A21, *A22, A23)* related to each input variable and four fuzzy sets related to the output variable *(B1, B2, B3, B4).*

Table 2 A fuzzy decision table.

	A21	A22	A23
A11		B1	B2
A12	B1	B2	B3
A13	B1	B3	B4

The modular structure of an FLC is:

1. A *Fuzzification interface* that transfers the values of input variables into fuzzy information, assigning grades of membership to each fuzzy set defined for that variable.

2. A *Knowledge Base* that comprises a Data Base, and a fuzzy control Rule Base. The Data Base is used in fuzzification and defuzzification processes.

3. An *Inference Engine* that inrers fuzzy control actions employing fuzzy implications and the rules of inference of fuzzy logic.

5. A *Defuzzification interface* that yields a non fuzzy control action from an inferred fuzzy control action.

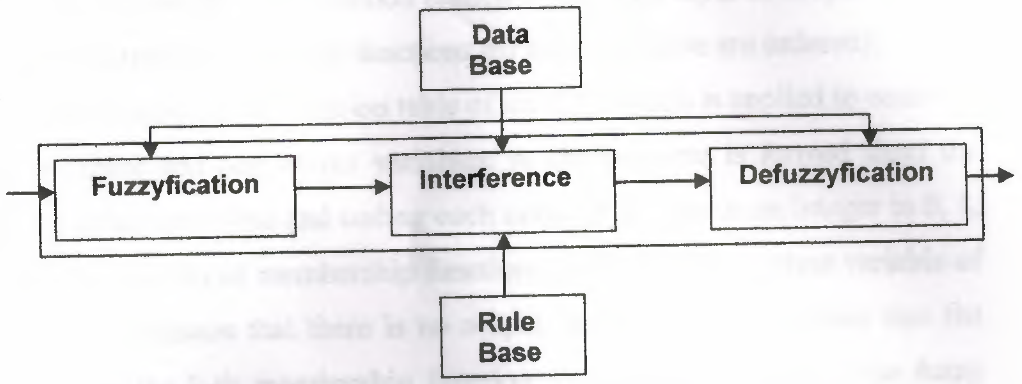


Figure 1 shows the defined structure. The fuzzy systems that will be described in the following have this structure.

2.2 Genetic Algorithms and Fuzzy Logic Controllers

The systems enumerated in this Section apply the general ideas of GAs to FLCs, using their own coding scheme. GAs are applied to modify the membership functions and/or the Rule Base. Some of these works use strings of numbers instead of strings of bits. The evaluation function, the composition of the first generation (initial population) and the termination condition are related to the task for which each FLC was designed.

Different methods are defined to apply GAs to the Rule Base, depending on its represntation: a set of rules, a decision table or a relational matrix.

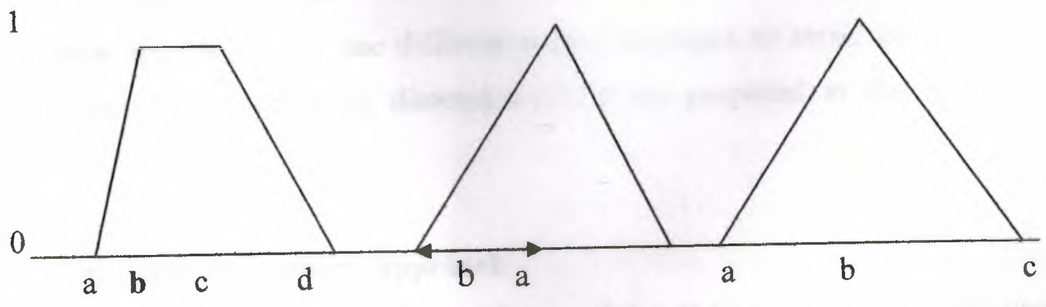


Figure 2 Some parameterized fuzzy membership functions.

Liska and Melsheimer use a rule base defined as a set with a fixed number of rules. The membership functions are labeled, coding each rule with integer numbers (labels) that define the membership function related to a certain input or output variable that is applied by the rule (membership functions for every variable are ordered).

GAs are used to modify the decision table of an FLC, which is applied to control a system with two input and one output variables. A chromosome is formed from the decision table by going row-wise and coding each output fuzzy set as an integer in 0, 1, ..., n, where n is the number of membership functions defined for the output variable of the FLC. Value "0" indicates that there is no output, and value "k" indicates that the output fuzzy set has the k-th membership function. Applying this code to the fuzzy decision table represented in table 2, the obtained string is (0,1,2,1,2,3,1,3,4). The mutation operator changes a fuzzy code either up a level or down a level, or to zero (if it is already zero, then it chooses a non-zero code at random).

Occasionally GAs are used to modify the fuzzy relational matrix (R) of an FLC with one input and one output. The chromosome is obtained by concatenating the m x *n* elements of R, where m and *n* are the number of fuzzy sets associated with the input and output variable respectively. The elements of R that will make up the genes may be represented by binary codes or real numbers, e.g., (0.5,0.8,0.2,0.0,0.0,0.3,1.0,0.1,0.0,0.0,0.3,1.0) for the relation defined by table 1.

All these works propose really interesting ways of adding learning capabilities to an FLC by means of GAs, obtaining results that show the ability of GAs to improve the performance of predefined FLCs. The application examples of these works present FLCs with two to five variables (input and output) and up to twenty membership functions (adding those from each variable). The question is how do they work when the problem dimensions increase.

Two new approaches that use different coding strategies to avoid the problems that will probably produce larger dimension FLCs are proposed in the following sections.

2.3 The Fuzzy Rule-Based Approach

As pointed out by different authors, when applying GAs to FLCs, there are two basic decisions to be made: how to code the possible solutions to the problem as a finite bit string, and how to evaluate the merit of each string.

The way we code a possible solution as a bit string constitutes the way we translate the problem defined on the space of parameters of the controller (represented by R^n) to a certain space where GAs will operate. But this is not the first question to be answered. Previously we have to define the general characteristics of the FLC to be applied, characteristics that will affect the dimension and the properties of the space of parameters where the adaptation or learning process will occur. As a result of the dimension and properties of this space, the learning process will become faster or slower and even treatable or untreatable. The obvious conclusion is that an adequate selection of the characteristics of the FLC is crucial for the whole learning process. The following Sections will present the coding scheme used by each learning system, but previously some common characteristics to both FLCs will be described.

The general structure of the FLC will be composed of a normalizer , a fuzzifier , an inference engine, a defuzzifier and a denormalizer; the scheme of Figure 1 with the addition of a normalization and a denormalization elements. The input variables will be linearly normalized from its real interval to [-1,1] interval. The fuzzy sets will have trapezoidal membership functions defined through four parameters (Figure 2, left). The rules applied by the inference engine will have a fuzzy antecedent and a fuzzy consequent (as in expression 6), and will be described as a set of fuzzy rules. The output variables will be linearly denormalized from [-1,1] interval to its real interval.

The use of normalized variables in fuzzification and defuzzification processes may be interpreted with two different meanings. If the normalization limits are fixed a priori, the only effect is conceptual, producing an FLC that works with variables defined in a normalized universe of discourse. If the normalization limits are considered as adaptable parameters throughout the learning process, the obtained effect may be interpreted as that of a controller with a parameterized gain with respect to each input and output variable. The parameterized gain for each variable will be directly related to the corresponding normalization limits, and may be tuned through the learning process.

The use of trapezoidal membership functions is one of the possible selections for parameterized membership functions. The most broadly used parameterized membership functions are: triangular , trapezoidal, Gaussian, bell and sigmoidal. These parameterized functions may be classified into two main groups: the piece-wise linear functions (triangular and trapezoidal) and the differentiable functions (Gaussian, bell and sigmoidal). While the first group produces a reduced computational complexity, the second one is particularly interesting when derivatives are involved in the learning process (most of the fuzzy-neural systems). In the applications that will be presented in the following, the learning process will not involve any kind of derivative, and the computational complexity leads us to select piece-wise linear functions. From piece-wise linear functions, trapezoidal membership functions will be selected since triangular membership functions may be considered as a particular case of a trapezoidal function.

When applying GAs to FLCs, rules are generally represented as a fuzzy decision table or in some cases (systems with one input and one output) as a fuzzy relational matrix. The use of a decision table has some advantages from the point of view of an easier analysis of completeness and consistency of the knowledge base.

2.4 Working with a Population of Rules

This first approach starts from Artificial Intelligence theory, with a conventional expert system architecture (with fuzzy knowledge), adding some new modules to create rules, evaluate their performance, etc. Nevertheless, the work is oriented towards the control of complex processes. Figure 3 shows the main blocks of the system, that will be described below.

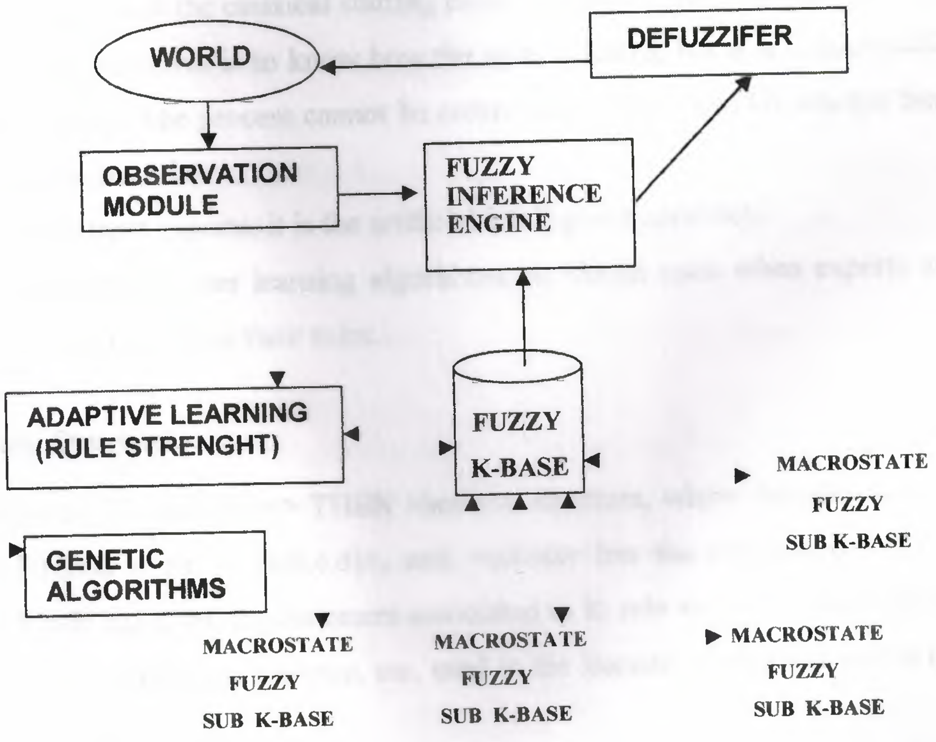

WORLD
DEFUZZIFER
OBSERVATION MODULE
FUZZY INFERENCE ENGINE
ADAPTIVE LEARNING (RULE STRENGHT)
FUZZY K-BASE
GENETIC ALGORITHMS
MACROSTATE FUZZY SUB K-BASE
MACROSTATE FUZZY SUB K-BASE
MACROSTATE FUZZY SUB K-BASE
MACROSTATE FUZZY SUB K-BASE

Figure 3 Rule System Architecture

Observation module: The facts base. This module must receive, analyze and filter the set of signals that the control system gets as inputs from the process. From this set of signals, the observation module generates the real input vector for the system.

Bad quality data are marked up, in order to avoid making inferences from them.

Fuzzy inference engine: It has been adapted to manage rules with explicit fuzzy sets and rule strength.

Defuzzifier: The output of the system must be a real number .The process doesn't understand terms such as "high" or "small" .

Adaptive Learning: Every rule has a strength (credibility) value that is modified by this module after every inference whether the rule has been applied or not. Genetic Algorithms: To create new rules.

Fuzzy Knowledge bases: Split into several different KBs, each one adapted to a particular big state of the world. These *macrostates* may be obtained from experts or from a clustering process. The reason for splitting the knowledge base into several small bases is that it will be easier to learn if the process is working in a small area of data than if it is working in the whole space. Initial knowledge bases can be obtained from three different ways:

1. Random base: it is the classical starting point for genetic learning algorithms. It may be useful if the objective is to know how the system learns, but it is unacceptable for real process control. The process cannot be controlled by a random knowledge base until it has learned some rules.

2. Knowledge from experts: it is the artificial intelligence approach.

3. Knowledge from other learning algorithms: to obtain rules when experts are unaffordables or to complement their rules.

2.4.1 *Rule Structure*.

Rules have an IF <condition> THEN <action> structure, where <condition> is a conjunction of terms < *Vx,* Є [a,b,c,d]>, and <action> has the expression < *Vy* Є [e,f,g,h]>. Each rule has a set of parameters associated to it: rule strength (credibility of the rule), rule strength average, life span, etc, used in the learning algorithms and in the fuzzy inference engine.

Each term must be read as *"Vx,* belongs to the set defined by [a,b,c,d]", where [a,b,c,d] defines a trapezoidal fuzzy set (Figure 2, left). Conditions may have an indeterminate number of terms. On the other hand, actions are unique. This is because rule strength (rule credibility) is related to how good or bad the action proposed for the given conditions is: different actions should have different strengths.

The following expression shows a rule sample:

If *V1* € [0.1, 0.2, 0.4, 0.5] and *V3* € [0.3, 0.3, 0.4, 0.6]

then *V7* € [0.2, 0.3, 0.3, 0.4] .

This rule definition gives flexibility to the system: facts are not limited to using predefined fuzzy sets. Rule generation algorithms will allow the system to find the best adapted rules for the control system.

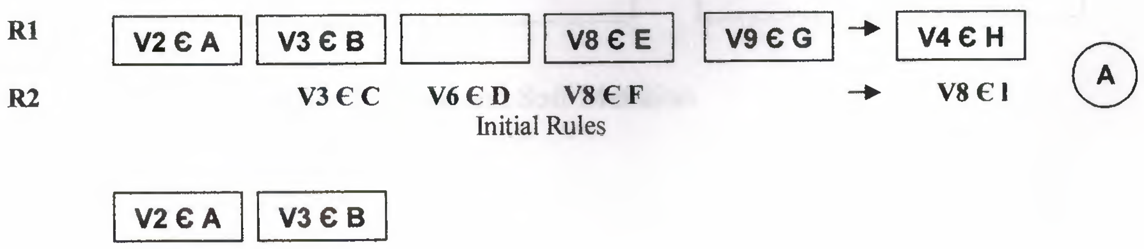
Defuzzification is based on the COA method after pondering the consequents with rule strength, truth value and knowledge base activation value. This activation value represents the degree of membership of the actual state to the macrostate associated with the corresponding KB.

2.4.2 *Rule Generation.*

Genetic algorithms have been selected for creating new knowledge because they allow the systems to find new rules both near to good ones and far from them, looking for unknown good control actions. Unlike traditional systems, in this architecture GAs are not fired in every cycle of the system. In fact, the rule generation process only works from time to time, when a particular KB has been used several times. While GAs do not work, the rule evaluation algorithm adjusts rule strength. The application of the GA will have four steps.

1. Selection: It is made randomly according to rule strength: the more strength, the more the probability of being selected.

2. Uniform Crossover: Mixes genetic material for creating new individuals. This operator has been selected over standard crossover because it allows the system to create new rules with whatever possible antecedents. Traditional crossover needs some kind of antecedents reordering (inversion operator).



Initial Rules

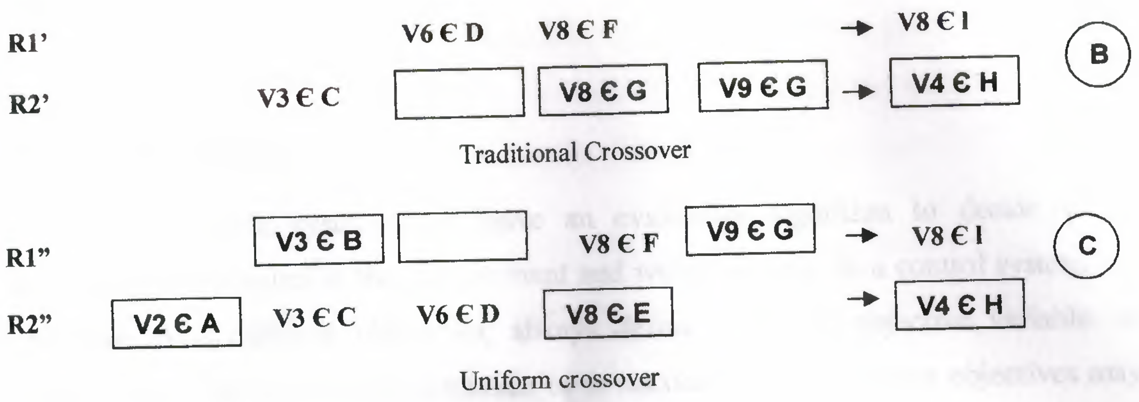


Figure 4 Crossover

3. Soft and Hard Mutation: Changes the fuzzy set (soft) or the variable (hard) ofthe fact (fig 5). Soft mutation moves the *(a, b,* c, *d)* values of the trapezium to create a new fuzzy set. New values are in ([0, *a+(b-a)/2], [a+ (b-a)/2, b+(c-b)/2*], *[b+ (c-b)/2, c+ (d-c)/2*], *[c+ (d-c)/2,* 1]) ranges. This operator is introduced to create variety in fuzzy sets. On the other hand, hard mutation changes the variable of the fact (antecedent or consequent). It ensures that every variable is going to be taken into account.

4. Insertion: New rules are placed in the *Limbo,* a special place where rules are tested and evaluated without affecting the output of the controller .Only when a rule has proved a good one, it is accepted and inserted into the KB.

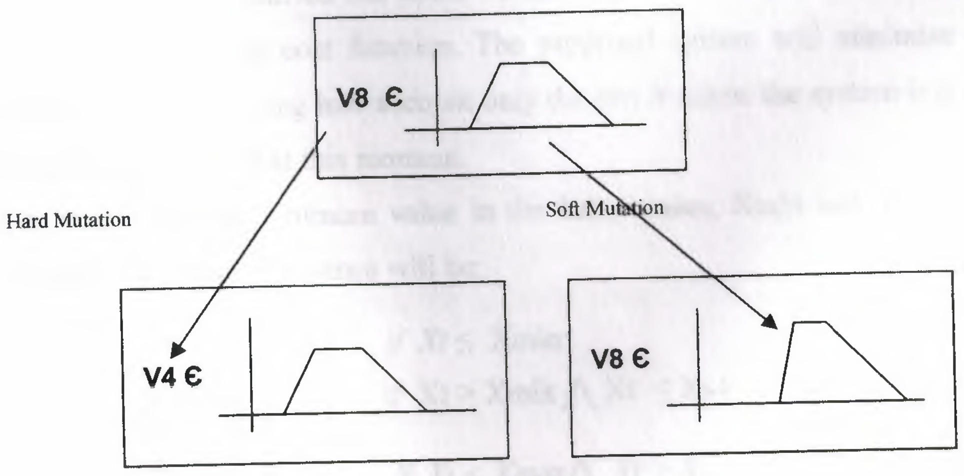


Figure 5 Hard and Soft Mutation

2.4.3 *Evaluation*

Every genetic system must have an evaluation algorithm to decide which individuals are adapted to the environment and which are not. In a control system, we can have three different objectives, always defined from an objective variable: to maintain it at a given value, to minimize or to maximize it. These three objectives may be seen as a cost minimization problem, where the cost function is defined as:

$C(t) = V(t)$ if the problem is to minimize,

$C(t) = -V(t)$ if the problem is to maximize,

$C(t) = |V(t) \text{-value}|$ if the problem is to maintain a given value.

To evaluate each particular rule, we first have to calculate whether the system is near reaching its objective. But that is not enough: we have to estimate the influence that this rule has in the final action (in other words, how the output of the rule affects the system). Both combined values will give us the rule evaluation. This calculus must be made for every rule whose premises are true according to the fuzzy logic paradigm used.

System performance: Our objective will be to guide a specified variable X (the cost function) to its minimum value.

It is possible to try to minimize the absolute cost value, but if the system does that, it is almost sure that at a given moment this absolute minimum is unreachable. It is possible too that the action carried out could be the best action at that moment, with only a little decrement in the cost function. The proposed system will minimize the variable in an *N* -window , taking into account only the last *N* cases: the system is going to look for the local minimum at this moment.

If it is looked for the minimum value in the last *n* cases, Xmin and *Xmax* will change with time. System performance will be:

$$SP = \begin{cases} 1 & \text{if } Xt \le \text{Xmin} \\ \dfrac{X - Xt\text{-}1}{Xmin - Xt\text{-}1} & \text{if } Xt > \text{Xmix} \wedge Xt \le Xt\text{-}1 \\ \dfrac{Xt\text{-}1 - x}{Xmax - Xt\text{-}1} & \text{if } Xt < Xmax \wedge Xt > Xt\text{-}1 \\ -1 & \text{if } Xt > Xmax \end{cases}$$

Applied rules: To know the influence of each applied rule, we have to analyze them rule by rule. Our data are: the fuzzy set *(A)* suggested for the variable *Va* in the action part of the rule, and the real values for this variable *(X)* at times *t* and *t* -1. Rule influence will be:

$$RI = \begin{cases} \mu_A(X_t) & \text{if } \mu_A(X_t) > 0 \\ 0 & \text{if } \mu_A(X_t) = 0 \;\wedge\; |A - X_t| \leq |A - X_{t-1}| \\ \dfrac{X_t - X_{t-1}}{X_{x-1} - X_l} & \text{if } \mu_A(X_t) = 0 \;\wedge\; |A - X_t| < |A - X_{t-1}| \end{cases}$$

where $|A - X|$ is the distance from X to the fuzzy set A (i.e., to the nearest point of the support of *A),* and X_l is the upper (if $X_t > X_{t-1}$) or lower (if $X_t < X_{t-1}$) limit of the Universe of Discourse of variable X.

Rule evaluation: Final rule evaluation is obtained by multiplying the rule influence estimation by system performance. If this performance is positive (the process is working well), rule evaluation depends on the rule influence: if the rule has been applied, final evaluation will be positive; if not, it will be negative. Dual considerations may be made for negative performance. And, of course, once have the rule evaluation, it must be pondered by its overall truth value, to obtain the final value 4.

New strength for the rule: Rules increase their strength if their evaluation is positive, and decrease it otherwise. The expression to obtain this strength variation is:

$$S_{R,t} = \begin{cases} S_{R,t-1} + K \bullet T_R \bullet E \bullet (1 - S_{R,t-1}) & \text{if } E \geq 0 \\ S_{R,t-1} + K \bullet T_R \bullet E \bullet S_{R,t-1} & \text{if } E < 0 \end{cases}$$

where K is a constant that allows fixing of the memory of the system *(K* near O produces slow variations and vice-versa) *TR* is the truth value of the rule, and *E* is the final evaluation of rule *R* .

2.4.4 *The limbo: How it works.*

As has been shown, the evaluation algorithm allows the assignation of strength to every rule with positive truth value, regardless of whether they have been used by the fuzzy inference engine or not. In fact, what the algorithm uses is the *system* *performance* and the proposed actions of true rules. The real action made by the controller is not taken into account. This means that rule strength may be updated for those rules that are in the KB and for those that are in the limbo and would have been able to be fired. The rules that live in the limbo have been created by GAs but they have to prove their suitability before being used by the inference engine. In order to select or reject a rule from the limbo several parameters are used:

•Rule parameters:

-Rule Age *(R.A.):* Number of inferences made since rule joined on to the limbo.

-Rule Use *(R.U.):* Sum, over these inferences, of the successive truth values of the rule.

-Rule Activations *(R.Ac.):* Number of inferences in which the rule should have taken part (the antecedent had a truth value grater than zero).

-Equivalent Rule Evaluation *(E.R.E.):* Constant evaluation that the rule should have obtained to reach its present strength value after the same number of inferences.

•Limbo parameters

-Limit Age *(L.A.):* Rule Age at which a rule has to leave the limbo.

-Minimum Rule Activations *(M.R.Ac.):* Value of *R.Ac.* that once reached allows a rule to be promoted from the limbo to the KB, before its Limit Age arrives.

-Minimum Equivalent Evaluation *(M.E.E.):* Minimium value of the *E.R.E.* that allows a rule to be promoted to the KB.

With these parameters in mind; a rule is killed once it has reached the Limit Age *(R.A.≥L.A.),* if it has not been used *(R.Ac.* = 0) or it has not been a good rule *(E.R.E.<M.E.E.).* On the other hand, a rule is promoted from the limbo to the KB *if R.A. ≥ L.A.* and *E.R.E.*≥M.E.*E.;* or *if R.Ac. ≥ M.R.Ac.* and *E.R.E.≥M.E.E.* (in this second case, the system does not wait to reach *L.A.:* if the rule is proving a good one, after a minimum number of uses it is promoted to the KB). In any other case the rule continues in the limbo.

In order to prevent the convergence of the KB (what in this case means to have a lot of identical rules in it), there is an additional filter to avoid too many copies of any rule: if a new rule is going to pass from the limbo to the KB, it should not have more than a given number of copies of it in the KB. In other situations, the rule is not inserted in the Rule Base.

2.4.5 *How to create rules from data records.*

With the proposed learning paradigm the control system can be trained with data files. The system will take a vector from a file, will determine the suggested actions after fuzzy evaluation, will read a new vector from the file, and will adjust rule strength both in the KB and in the limbo. From time to time GAs generate new rules to insert in the limbo and some rules go from the limbo to the KB.

That means that working with data files, and starting from a randomrule base, the system can teach itself the initial rule base. This rule base will be adapted on line, once the program is controlling the process.

2.4.6 *An application example and some learning results.*

A control system with this algorithm is being installed in a fossil power plant at Velilla. The objective of the system is to get low consumption while generated power is constant. In fact it is not a control system but a suggestion system.

An acquisition module gives 23 variables to the suggestion system. It will suggest operating 11 operation variables. The objective is to minimize the heat rate (ratio between used coal and generated power). It has been decided to use a continuous learning system because this environment is very complex and time variant (broken or dirty pipes, slow pumps, air temperature, humidity, ...are parameters that are out of our control and can affect the control system).

In order to analyze how the system learns, the following experiment has been designed: The control system will start with a 100-rule random Knowledge Base, and it will be trained with three different files as shown in the previous section:

1. A real historic data file.

2. A random data file, to compare learning results with the first case.

3. The real file, with random cost function, to assure that learning is achieved thanks to the algorithm and it has nothing to do with the data file used.

Figure 6 Final (upper) and average (lower) strength distribution.

The parameters of the three tests are:

1. Number of cases: 3514.
2. Max. Number of Rules in the KB: 1000.
3. Learning Constant *K* (for strength update): 0.25. 4. Age Limit for the limbo: 300.
5. Minimum Rule Activations: 10.
6. Minimum Equivalent Evaluation: 0.1.

The six graphics (Figure 6) show the strength and average strength distribution after the 3514 cases. As it is shown, in both random cases the strength distribution is around 0.5 (the initial strength), while in the real case it is distributed over the upper part of the axis. Only less than a 10% of learned rules have a strength lower than 0.5.

As final conclusion, the proposed learning system based on Genetic Algorithms applied over fuzzy rules is able to learn good fuzzy control rules and can be used in a fuzzy control system for complex time variant processes.

2.5 Working with a Population of Knowledge Bases

In this case it will be used each Knowledge Base of the FLC as an individual of the population, then it will be maintained a population of different Knowledge Bases to be applied by the FLC. This idea is illustrated in Figure 7 .

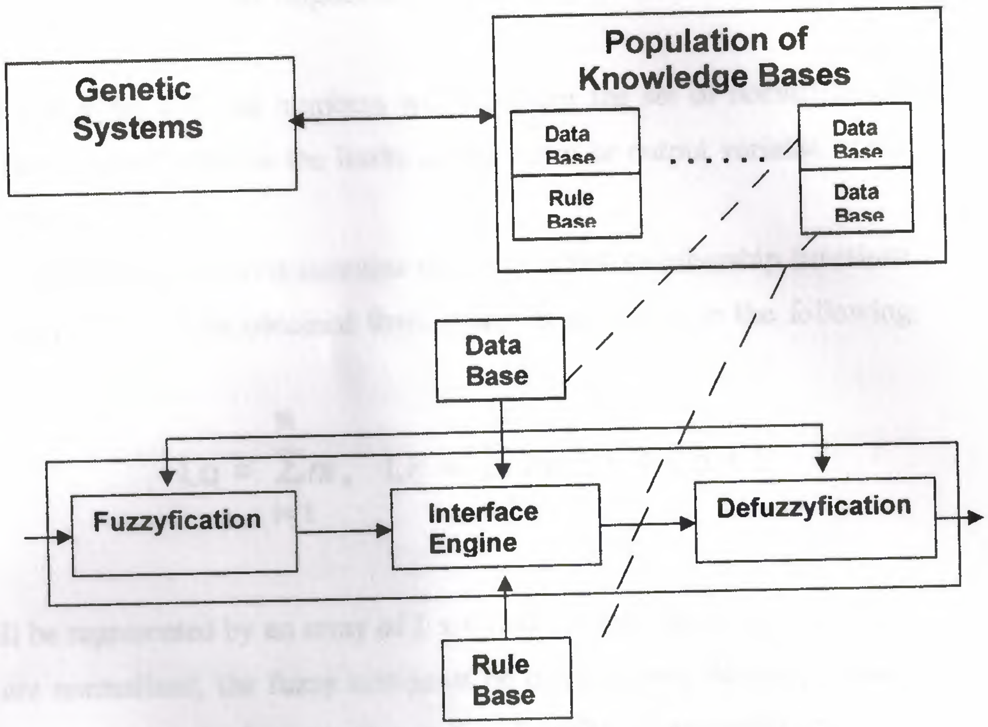


Figure 7 The genetic systemworks with a population of Knowledge Bases.

The Knowledge Base of the FLC contains the information to be coded, and is divided into a Data Base and a Rule Base. From the encoding point of view , the Data Base contains three different types of information: a set of parameters, a set of normalization limits and a set of membership functions; and the Rule Base contains a set of fuzzy control rules. All these types of information and their encoding schemes will be described in this section.

2.5.1 *Encoding the Data Base Information.*

Three elements must be encoded: the parameters of the FLC, the normalization limits and the membership functions. The set of parameters defines the system dimensions, that is, the number of input variables *(N)* and output variables *(M),* and (assembled on vectors *n* and m) the number of linguistic terms (or the number of fuzzy sets) associated with each member of the set of input variables and output variables. The i - th component of vector *n* $(n = \{ n_1 , ..., n_N \})$ represents the number of linguistic terms associated with the i-th input variable. The j-th component of vector m

($m=\{m_1, ..., m_M\}$) is the number of linguistic terms associated with the j-th output variable.

An array of ($N + M$) x 2 real numbers will represent the set of normalization limits. Each row in this array contains the limits of one input or output variable of the system ({ *Vmin* , *Vmax* }).

The set of membership functions contains the trapezoidal membership functions of L fuzzy sets, where L could be obtained from n and m as shown in the following equations:

$$La = \sum_{i=1}^{N} n_i, \quad Lc = \sum_{j=1}^{M} m_j \ , \ L = La + L$$

This set will be represented by an array of L x 4 real numbers ranged in [- 1 , 1] (as the variables are normalized, the fuzzy sets must be defined with the same range). Each row in the array contains the four parameters that describe a trapezoidal fuzzy set. It can be obtained a string of reals by concatenating the rows of this array as has been done in some works presented in Section 3. The code of membership functions will not be included in the code of rules.

2.5.2 *Fuzzy Rules Representation.*

As has been previously said, the fuzzy system will be characterized by a set of fuzzy rules.

When working with a multiple input system, decision tables become multidimensional. A system with three input variables produces three-dimensional decision tables. The number of "cells" of these decision tables is obtained by multiplying the number of linguistic terms associated with each input variable. Using the previous definitions of n and N, the number of "cells" (Lr) is:

$$Lr = \prod_{i=1}^{N} n_i ,$$

Each cell of the table describes a fuzzy rule, we will refer to these fuzzy rules as elemental fuzzy rules.

The structure of the fuzzy control rules contained in our FLC with parameters *{N,M,n,m}* is:

If x_i is A_{i0} and ... and x_k is A_{kp} ,

then y_j is B_{jq} and ... and y_l is B_{lr}

where *Xi* is an input variable, *Cio* is a fuzzy set associated with this variable ($0 \leq$ *ni*), *Yj* is an output variable and *Djq* is a fuzzy set associated with this variable ($q \leq$ mj) .All fuzzy inputs are 'connected' by the fuzzy connective 'and'. Several fuzzy sets related to the same variable could be connected with the aggregation operator 'or', appearing in a single rule, such as:

If x_i is *(Aio* or *Aip)* and ... then *yj* is *(Bjq* or *Bjr)* and ...

2.5.3 *Encoding the Rule Base Information.*

A set of rules represented by numerical values on a decision table or a relation matrix, have a direct translation into a string by means of a concatenating process. This method does not apply to a set of rules with a structure.

In our system, each rule will be encoded into two strings of bits: one string of length *La* for the antecedent (a bit for each linguistic term related to each input variable) and one string of length *Lc,* for the consequent. To encode the antecedent it will be startede with a string of *La* bits all of them with an initial value *0.* If the antecedent of the rule contains a fuzzy input like *Xi* is *Cij* , a 1 will substitute the *0* at a certain position *(p)* in the string:

$$P = j + \sum_{k=1}^{i-1} n_k$$

This process will be repeated for all the fuzzy inputs of the rule. It is important to point out that using this code, an input variable for which all the corresponding bits have value *0,* is an input variable whose value has no effect over the rule. The process to encode the consequent is similar to that described above, by only replacing n with m in expression 19. In this case, when all the bits corresponding to an output variable have value *0,* the rule has no effect over that output variable.

Considering an FLC with three input and one output variables *(N* = 3, *M* = 1) and parameters $n = \{5, 3, 5\}$ and m = {7} *(La* = 13, *Lc* = 7), the fuzzy rule

If x1 is *(A13* or *A14)* and *x3* is *(A31* or *A32)* ,

then y1 is *(B14* or *B15)*

is encoded as :

0011000011000 - 0001100.

Each rule of this FLC will be represented as a string of 20 bits (fixed length). The rule base will contain an unfixed number of rules, with a maximum value of *Lr* = 75, then it will be encoded as a string of up to 75 strings (unfixed length) of 20 bits.

2.5.4 *Evolving the Knowledge Base.*

The process of evolution learning may be described with the following scheme:

1. Start with a first generation *G(O)*.
2. Evaluate *G(O):* taking each member (string or other structure) of the population, decoding it, evaluating it by means of the evaluation function, and assigning a fitness value to the member .
3. While the termination condition was not met:

   a) Create a new generation *G(t+l),* by applying the evolution operators to the generation *G(t)*.

   (b) Evaluate G(t+l).
4. *Stop.*

The keys of this process are: the code, the evaluation function, the termination condition and the evolution operators. The code has been widely presented before and will be summarized here. The evaluation function and the termination condition are application specific. The main questions in this section are the evolution operators, or in a more general sense, the creation of *G(t* + 1) from *G(t)*.

The code that contains the information of the knowledge base is:

1. A string of 2 + *N* + *M* integers containing the dimensions of the FLC.
2. A string *of (N +M)* x 2 real numbers containing the normalization limits of the variables.

3. A string of *(La + Lc)* x 4 real numbers containing the definition of the trapezoidal membership functions.
4. A string of up to *Lr* rules, where each rule is a string of La + *Lc* bits.

It is possible to apply the evolution learning to any part of this code, according to the conditions of the learning process. From this point and in the application example we will work only with normalization limits (2) and with rule bases (4). The dimensions of the system and the membership functions will not be modified.

The effect produced by the modification of the normalization limits of a certain variable on the corresponding fuzzy sets (membership functions) are two:

•Each fuzzy set shrinks or expands in the same proportion as variable ranges do. The effect is the same as that produced when changing the gain of a controller .

•Each fuzzy set may be shifted to the right or to the left depending on its position and on the modifications of the normalization limits.

These changes are more restricted than those obtained with other methods, but the length of the employed code is reduced substantially, producing a shorter learning process; and a shorter or larger process may produce a treatable or untreatable problem.

Some evolution operators are obtained by directly adapting the classical genetic operators to the code. Others are new operators taking advantage of the code structure, or reducing its disadvantages.

Reproduction

The reproduction operator starts with an elite process that may be defined on the basis of a number of members, a percentage of members or an evaluation threshold (fixed or variable). By this process, a subset of *G(t),* referred to as the elite of generation *t (E(t)),* will be directly reproduced (copied) on *G(t* + 1).

In a second step, individuals of *G(t)* will be copied in the mating pool with a probability criterion based on the fitness of each one. According to the classical reproduction operator , members with a larger fitness value receive a larger number of copies. In addition to this reproduction operator , a second definition of reproduction based on a slightly modified operator has been defined. When working with this modified version of the operator, members (including those from the elite) with a larger fitness value have a higher probability of receiving a single copy (each individual will or will not receive a copy). This modified operator has been applied to the gait synthesis problem and is defined to avoid a sort of degenerative effect that produced a loss of diversity of members, when working with a small population (this is only an experimental effect, with no theoretical base),

Once the elite and the chromosomes to be reproduced have been selected, the number of members of $G(t + 1)$ must be adjusted to the maximum population, This process is performed by adding new elements to the elite (if the number of members is under the maximum population) or by extracting chromosomes from the mating pool (if the number of members is over the maximum population),

Crossover

Once a pair of parents *(m;* and m; taken from the mating pool) has been selected to be crossed (first step of the process), the crossover operator produces two new chromosomes by mixing the information provided by the parents' genes, A set of strings contains this information, two strings in our case (Normalization limits and Rules), The information contained in each string is not independent, then, if possible, the operator must work simultaneously (and not independently) on both strings, Each chromosome is composed of a pair of subchromosomes encoding the rule base (r) and the data base *(d),* The crossover of $mi = (ri, di)$ and $mj = (rj , dj)$ will produce two new chromosomes (mu and mv) ,

Rule base subchromosomes have no fixed length, and their genes are rules:

$$ri = \{ril, ..., rik\}$$

$$ri = \{rjl, ..., rj\imath\}.$$

To cross ri and rj a cutting point must be selected for each string, Since the lengths of the strings may not be equal, cutting points (β and γ) will be obtained independently for ri and rj

$$ri = \{ri1, \ldots ,ri\beta \mathrm{I} ri\beta{+}1, \ldots ,rik \}$$
$$rj = \{rj1, \ldots ,rj\gamma \mid rj\gamma{+}1, \ldots ,rj\imath \},$$
producing the new rule bases

$$ru = \{ri1, \ldots ,ri\beta \mid rj\gamma{+}1, \ldots ,rjI \}$$

$$rv = \{rj1, \ldots ,rj\gamma \mid ri\beta{+}1, \ldots ,rik \},$$

After rule bases are crossed, the process of crossing data bases will consider which rules from ri and rv go to ru or rv, An elemental fuzzy rule contains fuzzy inputs for all the variables. The rules we use contain fuzzy inputs and fuzzy outputs for only a subset of the input and output variables, then, normalization limits for the remaining variables have no influence on the meaning of the rule, A larger influence of a certain variable on rules that proceeding from r*i* go to *ru* , will produce a higher probability for this variable to reproduce in *du,* its corresponding range from *di,* The influence is evaluated by simply counting the number of rules that, containing the variable, are reproduced from *ri* to *ru* .The process of selection is independent for each variable and for each descendent (m*u* and *mv*) , consequently it is possible for both descendent to reproduce a certain range from the same antecedent.

Rules Reordering

When a fuzzy system is characterized by a set of fuzzy rules, their ordering is immaterial, the sentence connective *also* has properties of commutativity and associativity. When concatenating decision tables or relation matrices, the information of a certain gene depends on its content and its position. In our string of rules, the meaning of a gene becomes independent of the position. Therefore, rule position is arbitrarily defined for the members of the first population, it is immaterial for the output, but it biases crossover, then an operator to reorder rules will be added to the system.

This operator is applied to each set of rules produced by crossover operator, with a probability defined as a parameter of the evolution system. To reorder a rule base (*ri*) a cutting point *(β)* is selected uniformly at random , to create a new rule base (*r* j)

$$ri = \{ri1, \dots ,ri\beta \mid ri\beta{+}1, \dots ,rik\}$$
$$rj = \{rj\beta{+}1, \dots ,rik \mid ri{+}1, \dots ,ri\beta\},$$

The operator has no effect over the data base.

Mutation

The mutation is composed of two different processes: rule mutation and range mutation.

The rule mutation process will work at the level of bits that compose a rule. Each rule is composed of *(La + Lc)* bits and has the structure

$$p_{11} \dots p_{1n_1} \dots p_{N1} \dots p_{Nn_N},$$

C_{11} ... C_{1m_1} ...C_{lm_l} ... C_{Mm_M}

where P_{ij} is the bit related to j-th fuzzy set of the i-th input variable, and C_{ij} is the bit related to j-th fuzzy set of the i-th output variable. The rule mutation operator works as the classic genetic mutation applied to the string of bits defined by equation 26. This operator differs from classical mutation because it does not work at the level of genes (rules), but with the simplest element (bits) that compose genes.

The range mutation operator for a variable with range $[\lambda_1, \lambda_u]$, could be described by the following equation:

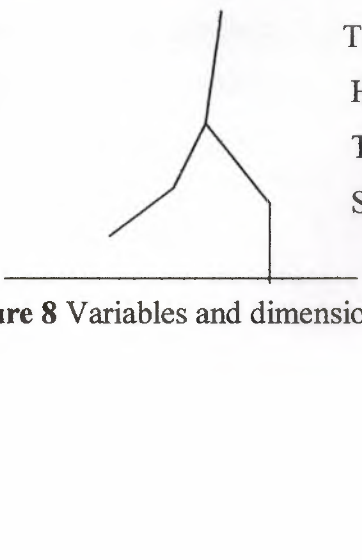
$$\lambda_1(t+1) = \lambda_1(t) + KP_1S_1(\lambda_u(t) - \lambda_1(t))/2$$

$$\lambda_u(t+1) = \lambda_u(t) + KP_2S_2(\lambda_u(t) - \lambda_1(t))/2,$$

where $K \in [0, 1]$ is a parameter of the learning system that defines the maximum variation (shift, expansion or shrinkage). P , P_1 and P_2 are random values uniformly distributed on [0, 1], and S, S_1 and S_2 take values -1 or 1 by a 50% chance. The symmetry of ranges is maintained, then when a variable has symmetric ranges ($\lambda_1 = -\lambda_u$) the following conditions are imposed: $P_2 = P_1$ and $S_2 = -S_1$.

2.5.5 *An application example and some learning results.*

This approach has been applied to the control of a simulated anthropomorphic (1. *75m, 70kg)* biped walking machine. The model is a six-link, 2-D structure, with two legs, a hip and a trunk (Figure 8). Each leg has a punctual foot, a knee with a degree of freedom and a hip with another degree of freedom. The contact of the stance-foot with the ground defines and unpowered degree of freedom(a_1) whose behavior is not directly controlled , but controlled through the adequate movements of the joints *(a_2 -a_6).* The control cycle of the FLC is 0.01 sc.



Trunk :	0.65m,	35kg.
Hip:	0.10m,	7kg.
Thigh:	0.50m,	9kg.
Shank:	0.50m,	5kg.

Figure 8 Variables and dimensions of the model.

The goal of the FLC is to define joint trajectories in such a way that the biped system describes a regular walk without falling or stopping. In this case, the GAs are applied not with an optimization aim, but with a diversification aim. The idea is to use biomechanical studies to obtain the KB of an FLC controlling a regular walk with a certain speed and stride length, and then apply GAs to create other knowledge bases capable of controlling regular walks with different speeds and stride lengths.

When the information contained in a KB6 is applied by the FLC, a sequence of movements is produced on the biped model. This sequence of movements will be evaluated, based on the *stability* and *regularity* of the walk, over a ten-second simulation (a thousand control cycles). The evaluation function measures the stability as a function of the time and the number of steps before falling (if the system falls before ending the simulation), or as a fixed value if the system has not fallen at the end of the simulation. The regularity of the walk is only computed if the system does not fall or stop, and is a function of the deviation of the stride duration from the mean period along the ten-second simulation. Some chromosomes will contain valid gait patterns, that is, gait description that produces a walk simulation without falling, or stopping the biped.

Some tests have demonstrated the ability of the learning system to generate valid gaits. Figure 9 shows four sequences of walk produced by genetically generated KBs. All of them have been obtained as the result of a single learning process with a reproduction rate of 0.8, a reordering rate of 0.5, a rule mutation rate of 0.01, a range mutation rate of 0.05 and a mutation constant (the parameter *K* on expression 27) equal to 0.5. The maximum population was 500 individuals the number of generations is forty and the initial population contained five KBs producing valid gaits, and a set of other individuals producing the fall of the biped system. These five KBs were extracted from biomechanical studies and the generated gaits showed speeds in the [1 .05,1 *.15]m/ sc* interval and stride lengths in the interval [0.67 *,O.68]m.* The sequences (Figure 9) present simulation results obtained with the biped model, containing a twenty images per second representation, except sequence 4 that is a ten images per second representation. Each sequence represents a particular case: the best, the shortest, the longest and fastest, and the slowest gait.

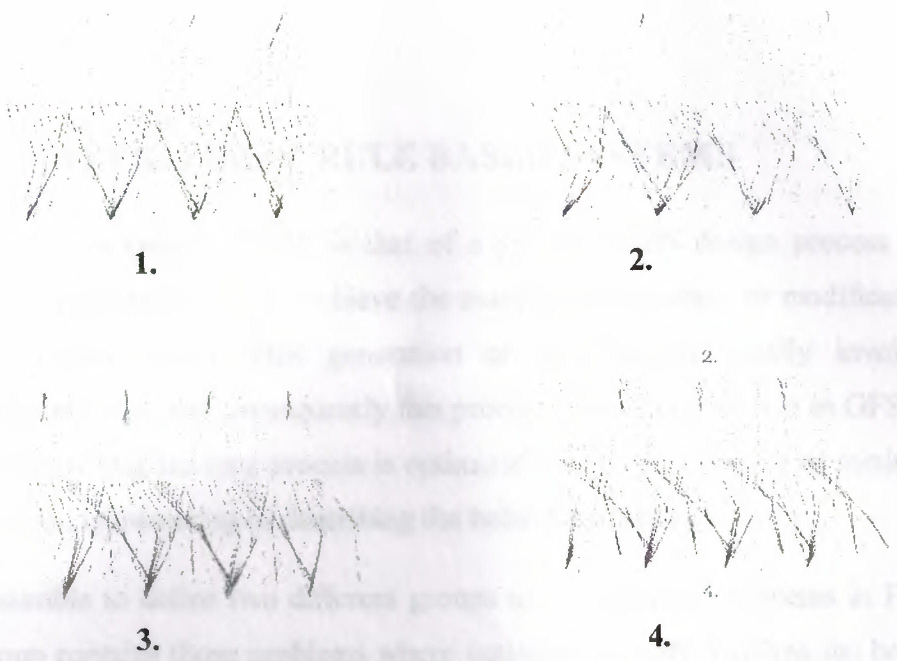


Figure 9 Different genetic generated gaits.

The main characteristics of each sequence are described below: the mean speed of the biped *(S)*, the stride length *(L)* and the time covered by the sequence *(T)*.

1. The most regular (best): *S* = 1.21m/sc, *L* = O.68m and *T* = *2sc*.
2. The shortest: *S* = 0. 76m/ *sc*, *L* = O.64m and *T* = *3sc*.
3. The longest and fastest: *S* = *1.29m/sc*, *L* = 0.78rn and *T* = *2sc*.
4. The slowest 7 : S = O.54m/ *sc*, *L* = O.67m and *T* = *4.5sc*.

The genetic process has produced the evolution of the covered ranges for speed and stride length from the initial [1.05, 1.15]m/sc and [0.67, *O.68]m*, to [O.54, 1.29]m/sc and *[0.64,0.78]m*.

3. GENETIC FUZZY RULE BASED SYSTEMS

The idea of a Genetic FRBS is that of a genetic FRBS design process which incorporates genetic techniques to achieve the automatic generation or modification of its KB (or a part of it). This generation or modification usually involves a tuning/learning process, and consequently this process plays a central role in GFSs. The objective of this tuning/learning process is optimization, i.e., maximizing or minimizing a certain function representing or describing the behavior of the system.

It is possible to define two different groups of optimization problems in FRBSs. The first group contains those problems where optimization only involves the behavior of the FRBS, while the second one refers to those problems where optimization involves the global behavior of the FRBS and an additional system. The first group contains problems such as modeling, classification, prediction and, in general, identification problems. In this case, the optimization process searches for an FRBS able to reproduce the behavior of a certain target system. The most representative problem in the second group is control, where the objective is to add a FRBS to a controlled system in order to obtain a certain overall behavior. Next, we analyze some aspects of the Genetic FRBSs.

3.1 Obtaining the knowledge for an FRBS

As a first step, it is interesting to distinguish between tuning and learning problems. In tuning problems, a predefined RB is used and the objective is to find a set of parameters de fining the DB. In learning problems, a more elaborate process including the modification of the RB is performed.

It can be distinguished between three different groups of GFSs depending on the KB components included in the genetic learning process.

3.1.1 Genetic tuning of the DB

The tuning of the scaling functions and fuzzy membership functions is an important task in the design of fuzzy systems. It is possible to parameterize the scaling functions or the membership functions and adapt them using GAs to deal with their parameters according to a fitness function.

As regards to the tuning of membership functions, several methods have been proposed in order to define the DB using GAs. Each chromosome involved in the evolution process represents different DB definitions, i.e., each chromosome contains a coding of the whole set of membership functions giving meaning to the linguistic terms. Two possibilities can be considered depending on whether the fuzzy model nature is descriptive or approximate, either to code the fuzzy partition maintaining a linguistic description of the system, or to code the rule membership functions tuning the parameters of a label locally for every rule, thereby obtaining a fuzzy approximate model.

3.1.2 Genetic learning of the RB

All the methods belonging to this family involve the existence of a predefined collection of fuzzy membership functions giving meaning to the linguistic labels contained in the rules, a DB. On this basis GAs are applied to obtain a suitable rule base, using chromosomes that code single rules or complete rule bases.

3.1.3 Genetic learning of the KB

There are many approaches for the genetic learning of a complete KB (RB and DB). We may find approaches presenting variable chromosome lengths, others coding a fixed number of rules and their membership functions, several working with chromosomes encoding single control rules instead of a complete KBs, etc.

3.2 The keys to the tuning/learning process

Regardless of the kind of optimization problem, i.e., given a system to be modeled/controlled, the involved tuning/learning process will be based on evolution. Three points are the keys to an evolutionary based tuning/learning process. These three points are: the population of potential solutions, the set of evolution operators and the performance index.

The population of potential solutions. The learning process works on a population of potential solutions to the problem, in this case, the potential solution is an FRBS. From this point of view , the learning process will work on a population of

FRBSs, but considering that all the systems use an identical processing structure, the individuals in the population will be reduced to DB/RB or KBs. In some cases the process starts off with an initial population obtained from available know ledge, while in other cases the initial population is randomly generated.

The set of evolution operators. The second question is the definition of a set of evolution operators that search for new and/or better potential solutions (KBs). The search reveals two different aspects: the exploitation of the best solution and the exploration of the search space. The success of evolutionary learning is specifically related to obtaining an adequate balance between exploration and exploitation, that finally depends on the selected these set of evolution operators.

The new potential solutions are obtained by applying the evolution operators to the members of the population of knowledge bases, each one of these members is referred to as an individual in the population. The evolution operators, that work with a code (called a chromosome) representing the KB, are basically three: **selection, crossover** and **mutation**. These evolution operators are in depth analyzed in subsections 2.1.2 and 2.1.3.

Since these evolution operators work in a coded representation of the KBs, a certain compatibility between the operators and the structure of the chromosomes is required. This compatibility is stated in two different ways: work with chromosomes coded as binary strings (adapting the problem solutions to binary code) using a set of *classical* genetic operators, or adapt the operators to obtain compatible evolution operators using chromosomes with a non-binary code. Consequently, the question of defining a set of evolution operators involves defining a compatible couple of evolution operators and chromosome coding.

The performance index. Finally, the third question is that of designing an evaluation system capable of generating an appropriate performance index related to each individual in the population, in such a way that a better solution will obtain a higher performance index. This performance index will drive the optimization process.

In identification problems, the performance index will usually be based on error measures that characterize the difference between the desired output and the actual output of the system. In control problems there are two different sources of information to be used when defining the performance index: information describing the desired behavior of the controlled system, or describing the desired behavior of the controller

(FRBS) itself. The second situation is closely related to identification problems. The definition of a performance index is usually more complex for the first situation, where the objective is to find a controller that gives the desired behavior in the controlled system. A possible method is illustrated in subsection 3.3.

The process. Summarizing the points that characterize a specific learning process, are: the initial population of solutions (obtained randomly or from some initial knowledge), the coding scheme for KBs (chromosomes), the set of evolution operators and the evaluation function. The initial population and the evaluation function are related to the specific problem while the coding scheme and the evolution operators could be generic. In addition to these four points, each evolutionary learning process is characterized by a set of parameters such as the dimension of the population (fixed or variable), the parameters regulating the activity of the operators or even theirs effect, and the parameters or conditions defining the end of the process or the time when a qualitative change in the process occurs.

3.3 A learning process of FLCs

FLCs represent a particular and widely applied kind of FRBSs. A genetic process using a Pittsburgh approach and working on an FLC is illustrated in Figure 5.

The process described in Figure 3 may be rewritten as follows in such a situation:

1. Start with an initial population of solutions that constitutes the first generation *(P(O))*.

2. Evaluate *P(O)*:

(a) take each chromosome (KB) from the population and introduce it into the FLC,

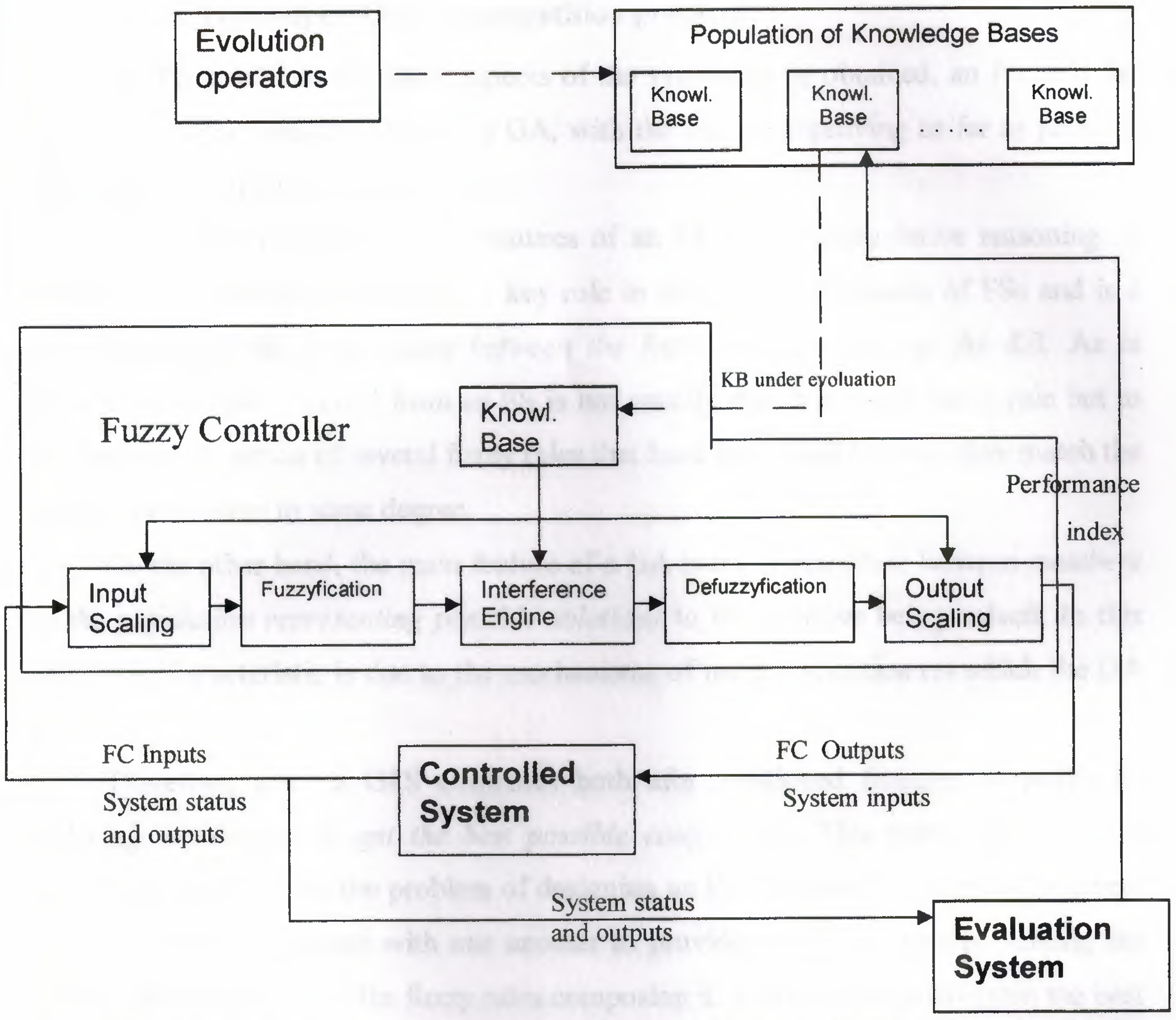


Figure 5: Evolutionary learning, with Pittsburgh approach, of the **KB** of an **FLC**

(b) apply the FLC to the controlled system for an adequate evaluation period (a single control cycle, several control cycles or even several times, starting out from different initial conditions) and

(c) evaluate the behavior of the controlled system by producing a performance index related to the KB.

3. While the Termination Condition is not met, do

(a)create a new generation *(P(t+l))* by applying the evolution operators to the individuals in P*(t),*

(b)evaluate *P(t+l)* and (c) t = t + 1.

(c) *t = t + 1.*

4. Stop.

3.4 The cooperation vs. competition problem

A GFS combines the main aspects of the system to be obtained, an FS, and the design technique used to obtain it, a GA, with the aim of improving as far as possible the accuracy of the final FS generated.

One of the most interesting features of an FS is the interpolative reasoning: it develops. This characteristic plays a key role in the high performance of FSs and is a consequence of the *cooperation between the fuzzy rules composing the KB.* As is known, the output obtained from an FS is not usually due to a single fuzzy rule but to the cooperative action of several fuzzy rules that have been fired because they match the input to the system to some degree.

On the other hand, the main feature of a GA is the *competition between members of the population representing possible solutions* to *the problem* being solved. In this case, this characteristic is due to the mechanisms of natural selection on which the GA is based.

Therefore, since a GFS combines both aforementioned features, it works by *inducing competition* to *get the best possible cooperation.* This seems to be a very interesting way to solve the problem of designing an FS, because the different members of the population compete with one another to provide a final solution presenting the best cooperation between the fuzzy rules composing it. The problem is to obtain the best possible way to put this way of working into effect. This is referred to as *cooperation vs. competition problem (CCP).*

The difficulty of solving the introduced problem depends directly on .the genetic learning approach followed by the GFS (Michigan, Pittsburgh or IRL approaches).

4 GENETIC TUNING of DB

The use of GAs for the tuning of DBs may be developed in two areas, the adaptation of contexts using scaling functions and the tuning of fuzzy membership functions.

4.1 Adapting the context

The use of scaling functions that are applied to the input and output variables of an FRBS, allows us to work with normalized universes of discourse where the fuzzy membership functions are defined. These scaling functions could be interpreted as gains associated with the variables (from a control engineering point of view) or as context information that translates relative semantics into absolute ones (from a knowledge engineering point of view). If using scaling functions, it is possible to fix them or to parameterize the scaling functions and adapt them. Linear and non-linear contexts have been used.

Linear context. It is the simplest scaling. The parameterized function is defined by means of two parameters (one, if used as a scaling factor). The effect of scaling is that of linearly mapping the real interval [a,b] into a reference interval (e.g., [0,1]). The use of a scaling factor maps the interval [-a,a] in a symmetrical reference interval {e.g., [-1,1]). This kind of context is the most broadly applied one. Genetic techniques have been applied to adapting the parameters defining the scaling factors and linear scaling functions.

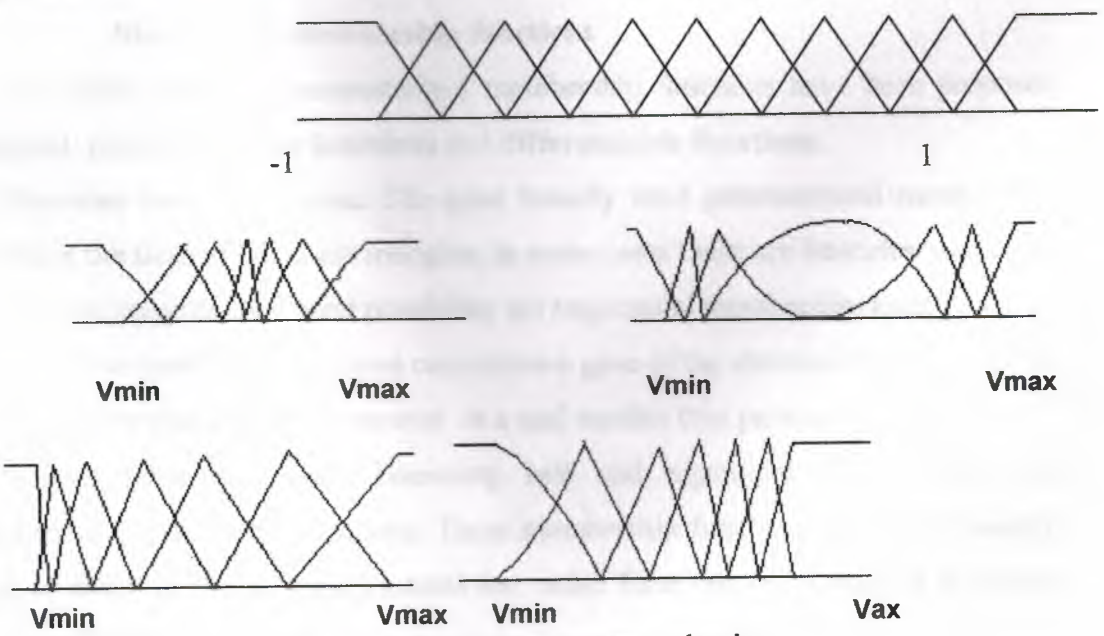


Figure 6: Nonlinear contexts adaption

Nonlinear context. The main disadvantage of linear scaling is the fixed relative distribution of the membership functions (uniformly distributed or not) once they have been generated. To solve this problem nonlinear scaling is used allowing us to obtain a modified relative distribution and a change in the shape of the membership functions. The definition of parameterized nonlinear scaling functions is more complex than in the linear case and a larger number of parameters are needed. The process actually requires two steps: previous scaling (linear) and nonlinear mapping. Parameterized potential and sigmoidal functions have been used when applying GAs to adapt the nonlinear context. Usually, the parameters (real numbers) constitute the genes of the chromosomes without binary representation.

Figure 6 shows a normalized fuzzy partition (top), a nonlinear adaption with lower granularity for middle or for extreme values (center) and lower granularity for lowest or for highest values (bottom).

4.2 Tuning the membership functions

Another element of the KB is the set of membership functions. This is a second point where GAs could be applied with a tuning purpose. As in the previous case of scaling functions, the main idea is the definition of parameterized functions and the subsequent adaptation of parameters. The different proposals differ in the coding scheme and the management of the solutions (fitness functions, ...).

4.2.1 Shape of the membership functions

Two main groups of parameterized membership functions have been proposed and applied: **piecewise linear functions** and **differentiable functions**.

Piecewise linear functions. The most broadly used parameterized membership functions in the field of GFSs are triangles, in some cases these are isosceles and other times they are irregular. A second possibility are trapezoidal membership functions.

Each Darameter of the function constitutes a gene of the chromosome that may be a binary code representing the parameter or a real number (the parameter itself).

Differentiable functions. Gaussian, bell and sigmoidal are examples of parameterized differentiable functions. These membership functions have been broadly applied in different fuzzy-neural systems but radial functions and Gaussian functions are used in GFSs too.

4.2.2 Scope of the semantics

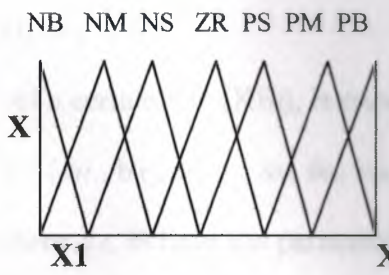
The genetic tuning process of membership functions is based on two variants, depending on the fuzzy model nature, whether approximate or descriptive.

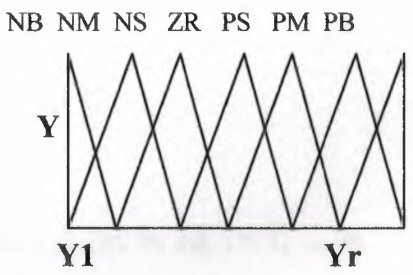
The descriptive fuzzy model is essentially a qualitative expression of the system. A **KB** in which the fuzzy sets giving meaning (semantic) to the linguistic labels are uniformly defined for all rules included in the RB. It constitutes a descriptive approach since the linguistic labels take the same meaning for all the fuzzy rules contained in the RB. The system uses a global semantics.

In the approximate fuzzy model a KB is considered for which each fuzzy rule presents its own meaning, i. e., the linguistic variables involved in the rules do not take as their values any linguistic label from a global term set. In this case, the linguistic variables become fuzzy variables. The system applies local semantics.

Figure 7 and the examples described in the following paragraphs illustrate these two variants, and their particular aspects reflected in the coding scheme.

a) Descriptive Knowledge Base





Rl : If X is NB then Y **is NB**

R2: If X isNM 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

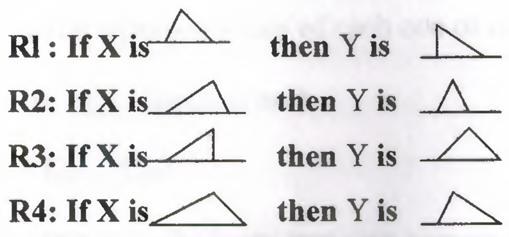


Figure 7: Descriptive versus Approximate fuzzy models

4.2.3 The approximate genetic tuning process

As mentioned earlier , each chromosome forming the genetic population will encode a complete KB. More concretely, all of them encode the RB, *R,* and the difference between them are the fuzzy rule membership functions, i. e., the DB definition.

Taking into account a parametric representation with triangular-shaped membership functions based on a 3-tuple of real values,

R_i : IF x_1 is A_{i1} and ... and x_n is A_{in}

THEN y is B_i,

of a certain KB {KB$_l$), is encoded in a piece chromosome C_{li}:

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

where A_{ij}, B_i have the parametric representation (a_{ij}, b_{ij}, c_{ij}), (a_i, b_i, c_i), $i = 1, ..., m$ (m represents the number of rules), $j = 1, ..., n$ (n is the number of input variables).

Therefore the complete RB with its associated DB is represented by a complete chromosome C_l:

$C_l = C_{li}\ C_{l2} \dots C_{lm}$

This chromosome may be a binary or a real coded individual.

4.2.4 The descriptive genetic tuning process

In this second genetic tuning process each chromosome encodes a different DB definition based on the fuzzy domain partitions. A primary fuzzy partitin is represented as an array composed by 3 . N real values, with N being te 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 up by joining the partial representations of each one of the variable fuzzy partitions,

$$C_{ij} = (a_{il}, b_{il}, c_{il}, \ldots, a_{iN_i}, b_{iN_i}, c_{iN_i})$$

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

where C_{ji} represents the fuzzy partition corresponding to the $i-th$ variable.

5 LEARNING WITH GFSs

In this Section, we study the learning of FRBSs by means of genetic learning processes, dividing it into two subsections, the learning of RKB, and presenting some notes on the application of the different genetic approaches to these problems.

5.1 Genetic learning of RB

The third element of the KB is the RB. It is possible to represent the RB of a fuzzy controller with three different representations and all of them are used in evolutionary fuzzy controllers. These representations are: relational matrix, decision table and list or set of rules.

Genetic learning of RB makes sense only when working with a descriptive approach, since in the approximate approach, modifying the rules implies the modification of membership functions.

5.1.1 Michigan approach

A Michigan learning algorithm of fuzzy rules merges the credit assignment mechanisms of CSs and fuzzy systems, integrating a fuzzy rule base (each classifier represents a fuzzy rule and the population represents the RB) and a fuzzy inference system instead of the rule base and production system in a classical CS. This learning process receives the name of Fuzzy Classifier Systems (FCBs).

Some peculiarities of this model are that a fuzzyfication process is defined in the input interface, which fuzzyfies inputs into fuzzy messages by creating minimal messages, one for each fuzzy set defined over the variable. Then each message has an associated activity level which measures the degree of belonging to the input variable defined by the fuzzy sets represented by the message. For each rule, a previously fixed credit is given. As a result of actions performed in the environment by the fuzzy inference system, the credit assignement system receives the payoff of that action from the environment, and in accordance with the degree of conformity of the rule, the payoff is opportioned to each rule by increasing or decreasing its credit. Therefore, the system learns fuzzy relations between the fixed fuzzy sets.

Figure 8 shows the organization of an FCSs.

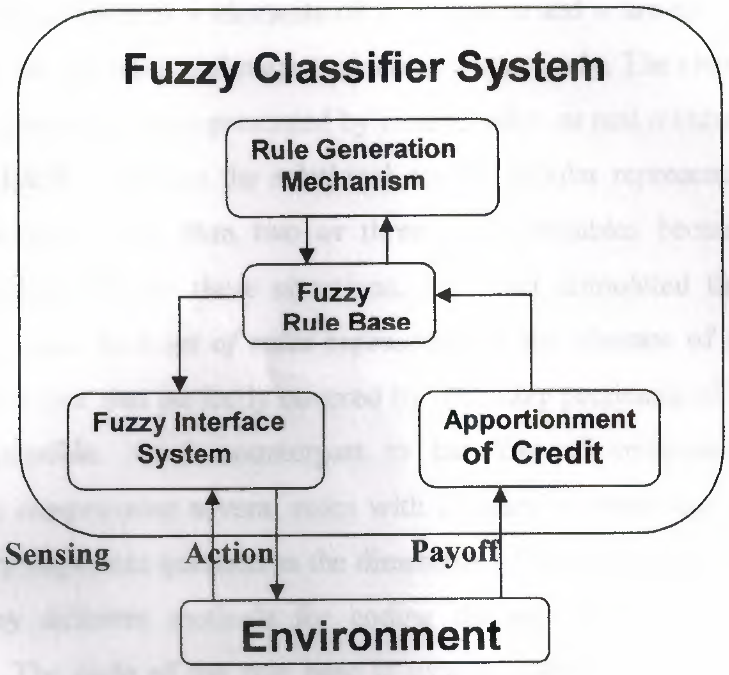


Figure 8: Organization of a fuzzy classifier system

5.1.2 Pittsburgh approach

The Pittsburgh approach has been applied to learn rule bases in two different situations. The first situation refers to those systems using a complete rule base represented by means of a decision table or a relational matrix: The second situation is that of FRBSs, whose RB is represented using a list or set of fuzzy rules.

Using a complete RB. A tabular representation guarantees the completeness of the knowledge of the FRBS in the sense that the coverage of the input space (the Cartesian product of universes of the input variables) is only related to the level of coverage of each input variable (the corresponding fuzzy partitions), and not to the rules.

Decision tables. A possible representation for the RB of an FS is a decision table. It is a classical representation used in different GFSs. A chromosome is obtained from the decision table by going row-wise and coding each output fuzzy set as an integer or any other kind of label. It is possible to include the "no output" definition in a certain position, using a "null" label.

Relational matrices. Occasionally GAs are used to modify the fuzzy relational matrix (R) of a Fuzzy System with one input and one output. The chromosome is obtained by concatenating the m x *n* elements of R, where m and *n* are the number of fuzzy sets associated with the input and output variables respectively. The elements of R that will make up the genes may be represented by binary codes or real numbers.

Using a partial RB. Neither the relational nor the tabular representations are adaptable to systems with more than two or three input variables because of the dimension of a complete RB for these situations. This fact stimulated the idea of working with sets of rules. In a *set of rules* representation the absence of applicable rules for a certain input that was perfectly covered by the fuzzy partitions of individual input variables is possible. As a counterpart to the loss of completeness, this representation allows *compressing* several rules with identical outputs into a singular rule and this is a really important question as the dimension of the system grows.

There are many different methods for coding the rule base in this kind of evolutionary system. The code of the rule base is usually obtained by concatenating rules codes.

Rules offixed length. A first approach is to represent a rule with a code of fixed length and position dependent meaning. The code will have as many elements as the number of variables in the system. A possible content of these elements is: a label pointing to a certain fuzzy set in the fuzzy partition of the variable or a binary string with a bit per fuzzy set in the fuzzy partition of the variable coding the presence or absence of the fuzzy set in the rule.

5.1.3 Learning an RB with the IRL approach

Using this approach a chromosome represents a fuzzy rule and the whole rule base is obtained through an iterative process where in dividual rules are obtained at each iteration based on a Genetic process.

From the description given in subsection 2.2.3, may be seen that in order to learn rules using an algorithm based on GAs with an IRL approach, it is needed, at least, the following:

1. a criterion for selecting the best rule at each iteration,
2. a penalization criterion, and
3. a criterion for determining when enough rules are available to have a solution to the problem.

The first criterion is normally associated with one or several characteristics that are desirable so as to determine good rules. Usually criteria about the rule strength have been proposed (number of examples covered), criteria of consistency of the rule or criteria of simplicity.

The second criterion is often associated, although it is not necessary, with the elimination of the examples covered by the previous rules.

Finally, the third criterion is associated with the completeness of the set of rules and must be taken into account when all the examples in the training set are sufficiently covered and no more rules are needed to represent them.

The IRL approach does not analyze any relationship between the rules that are obtained. That is why, once the rule base has been obtained, it may be improved either be cause there are rules that may be refined or redundant rules if high degrees of coverage are used. Therefore, after this is done, some post-processing methods are used for improving the accuracy of the rule base.

An inductive learning algorithm for RB, called SLAVE has been designed based on this approach. SLAVE selects a rule covering the maximum number of positive examples and simultaneously verifies a soft consistency condition to a high degree. SLAVE uses a GA in this process.

6 AN EXAMPLE OF GFS

This section will describe, in a few lines, one of the GFSs previously cited, specifically a GFS learning RBs using a Pittsburgh approach and represening the rule base with a decision table. This method was proposed by Philip Thrift.

Given a single output FRBS with n input variables, a fuzzy partition is defined for each variable (n + 1 fuzzy partitions). In this case each fuzzy partition contains five or seven fuzzy sets. An n-dimensional decision table is then made up by placing the consequents of each rule in the place corresponding to its premise. Entries in the table can be either one of the labels representing a fuzzy set of the output variable partition, or a blank representing no fuzzy set output for the corresponding rule.

The population of potential solutions. The population of potential solutions will be made up of RBs applied by a common processing structure to solve a specific problem. Because the learning process is centered on rules and all the KBs will contain an identical DB, consequently the population of solutions can be reduced to a population of RBs. Each RB is represented by a decision table, and these decision tables must by coded to constitute suitable genetic material.

Each position in the decision table will represent a gene of the chromosome coded with an integer in {0, 1, ..., 5}, with its 6 possible values corresponding to the 5 components of the fuzzy partition and the blank output. A chromosome is obtained by going rowwise through the table and producing a string with the integers found at each place in it. For a system with two input variables and five fuzzy sets per partition, the decision table will contain 5 x 5 places and consequently will generate a chromosome with 25 genes.

The population where the genetic process will be applied is a number of chromosomes coded as strings with 25 integers in {0, 1, ..., 5}.

The set of evolution operators. The system uses a standard two point crossover and a mutation operator that changes a fuzzy code either up one level or down one level, or to the blank code. When the mutation operator acts on a blank code, a non-blank code is generated at random. An elite strategy allows the best solution at a given generation to be directly promoted to the next.

The performance index. The system described is applied to center a cart by applying a force on it. The objective is to move the cart to the zero position and velocity in a minimum time. Each RB is tested by applying the FRBS to control the cart starting at 25 equally spaced starting points and over 500 steps (0.02 sc. for each step). The performance index assigned to an RB is *500-T* where *T* is the average time (number of steps) required to place the cart sufficiently close to the center (max(lxl,lvl)<0.5). If, for a certain starting point, more than 500 steps are required, the process times out and 500 steps are recorded.

With this performance index the learning process becomes a minimization problem since the best solution is the one with the lowest average time to center the cart (the highest performance index).

7 Concluding Remarks

In this tutorial it is dealt with several issues relating to GFSs, focusing on the use of GAs for designing FRBSs. It is presented the most important keys of the tuning/learning processes. The two genetic tuning approaches (adapting the context or the membership functions) and the three different modes to cope with the problem of designing RB or KBs with GAs (Michigan, Pittsburgh and IRL approaches) have been attached, and different proposals developing each one of them have been analyzed.

Finally, it should point out that although the application of GAs for designing fuzzy systems is recent, it has seen of increasing interest over the last few years and will allow to fruitful research to be carried out in the building of fuzzy logic-based intelligent systems.

Fuzzy Logic Control constitutes a growing and promissing area of control theory . The main goal of this chapter was the description of a learning methodology for Fuzzy Logic Controllers, based on the evolution of their knowledge base. Two different approaches have been analyzed. Both of them use rule bases constructed as sets of rules, reducing the dimensionality of the learning space. Both methods have been successfully applied to different and complex control problems. A brief description of two of these applications has been included in the chapter .

REFERENCES

O. Cordon, F. Herrera, M. Lozano. A classified review on the combination fuzzy logic genetic algorithms bibliography. Tech. Report *#DECSAI* -95129, Dept. of Computer Science and A.I., Univ. of Granada, 1995. Available at the URL address: http:/ /decsai.ugr.es/ herrera/flga.html.

O. Cordon, F .Herrera, M. Lozano. A three-stage method for designing genetic fuzzy systems by learning from examples. In: H. M. Voight, W. Ebeling, E. Rechembering , H.P. Schwefel (Eds.), L.N.C.S 1141, Proc. 4th International Conference on Parallel Problem Solving from Nature(PPSN iv) Berlin 1996 720-729.

O. Cordon, F .Herrera. A hybrid genetic algorithm-evolution strategy process for learning fuzzy logic controller knowledge bases. In: F. Herrera and J. L. Verdegay, (Eds.), Genetic Algorithms and Soft Computing Physica Verlag ,1996, 251-278.

O. Cordon, F. Herrera. A three-stage evolutionary process for learning descriptive and approximate fuzzy logic controller knowledge bases from examples. International Journal of Approximate Reasoning (1997) To appear.

Magdalena L. (1994) *Estudio de Ia coordinaci6n inteligente en robots bipedos: aplicacion de 1ogica borrosa y algoritmos geneticos.* Doctoral dissertation, Universidad politecnica de Madrid (Spain).

Magdalena L. and Monasterio F. (March 1995) Evolutionary-based learning applied to fuzzy controllers. In *Proceedings 4th lEEE International Conference* on *Fuzzy Systems and the Second lnternational Fuzzy Engineering Symposium, FUZZ-lEEE/lFES'95,* volume III, pages 1111-1118.

Velasco J. and Ventero F. (1994) Some applications of fuzzy clustering to fuzzy control systems. In *9rd* Int. *Conf.* on *Fuzzy Theory and Technology.* Durham, NC, USA.