Mathware & Soft Computing 4 (1997) 233-249

Multi-stage Genetic Fuzzy Systems Based on the Iterative Rule Learning Approach*

Antonio González and Francisco Herrera
Dpto. de Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingeniería Informática
Universidad de Granada. 18071-Granada (Spain)

Abstract

Genetic algorithms (GAs) represent a class of adaptive search techniques inspired by natural evolution mechanisms. The search properties of GAs make them suitable to be used in machine learning processes and for developing fuzzy systems, the so-called genetic fuzzy systems (GFSs).

In this contribution, we discuss genetics-based machine learning processes presenting the iterative rule learning approach, and a special kind of GFS, a multi-stage GFS based on the iterative rule learning approach, by learning from examples.

Keywords: Fuzzy logic, fuzzy rules, genetic algorithms, machine learning.

1 Introduction

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 proven to provide robust search capabilities in complex spaces, offering a valid approach to problems requiring efficient and effective searching.

Much of the interest in GAs is due to the fact that they provide a set of efficient domain-independent search heuristic which are a significant improvement over traditional methods without the need for incorporating highly domain-specific knowledge.

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 [22, 14].

Two alternative approaches, in which GAs have been applied to learning processes, have been mainly used, the Michigan ([29]) and the Pittsburgh ([38]) approaches. In the first one, the chromosomes correspond to classifier rules which

*This work has been supported by the CICYT under Projects TIC95-0453 and TIC96-0778 are evolved as a whole, whereas in the Pittsburgh approach, each chromosome encodes a complete set of classifiers. A third way will be presented as an alternative to these models, the *iterative rule learning approach* where each chromosomes represents only one rule learning.

On other hand, GAs have proven to be a powerful tool for automating the definition of the fuzzy systems knowledge base (KB), since adaptive control, learning and self-organization fuzzy systems can be considered in a lot of cases as optimization or search processes. Their advantages have extended the use of GA s in the development of a wide range of approaches for designing fuzzy systems in the last few years. These approaches receive the general name of *Genetic Fuzzy Systems* (GFSs) [8].

The KB is composed of two components, a *Data Base* (DB), containing 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 representing the expert knowledge. It is possible to distinguish different groups of GFSs according to the KB components included in the learning process: learning or tuning the DB with a fixed set of rules, learning the RB with fixed membership function sets and learning the KB, that is, the fuzzy membership functions and fuzzy rules.

The genetic learning processes belonging to the latter two last classes can do the learning simultaneously or in various stages. In the following we present a *multi-stage GFS* (MSGFS) for learning RBs or KBs based on the *iterative rule learning approach.*

In order to do so, the paper is organized as follows: The next Section is devoted to presenting the GAs; in Section 3, we introduce the genetic learning approaches with special attention to the *iterative rule learning approach*; in Section 4 the GFSs and the MSGFS are presented; and in the final Section, some concluding remarks are made.

2 Genetic Algorithms

GAs are search algorithms that use operations found in natural genetics to guide the trek through a search space. GAs use a direct analogy of natural behaviour. They work with a population of chromosomes, each one representing a possible solution to a given problem. Each chromosome is assigned a fitness score according to how good a solution to the problem it is. GAs are theoretically and empirically proven to provide robust search in complex spaces, giving a valid approach to problems requiring efficient and effective searching [15].

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

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

(1) evaluation of individual fitness,

(2) formation of a gene pool (intermediate population) and

(3) recombination and mutation.

The next procedure shows the structure of a simple GA.

```
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)
```

A fitness function must be devised for each problem to be solved. Given a particular chromosome, a solution, the fitness function returns a single numerical fitness, which is supposed to be proportional to the utility or adaptation of the individual which that chromosome represents.

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 procedure proposed in [1], and 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.

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

A crossover operator combines the features of two parent structures to form two similar offspring. It is applied at a random position with a probability of performance, the crossover probability, P_c. 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, P_m.

It is generally accepted that 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.).*

The basic principles of GAs were first laid down rigorously by Holland [28], and are well described in many books such as [15, 35].

3 Genetic Learning Approaches

Since the beginning of the 80s there has been growing interest in applying methods based on GAs to automatic learning problems, especially the learning of production rules on the basis of attribute-evaluated example sets. The main problem in these applications consists of finding a "comfortable" representation in the sense that it might be capable both of gathering the problem's characteristics and representing the potential solutions.

Classically, two genetic learning approaches have been proposed:

The Michigan approach: The chromosomes are individual rules and a rules 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,
2. the *credit assignment system* developing learning by the modification and adjustment of conflict-resolution parameters of the classifier set, their strengths; Holland's Bucket Brigade is one example of it [30], and
3. the *classifier discovery process* that generates new classifiers from a classifier set by means of GAs.

A complete description is to be found in [3].

> **The Pittsburgh approach**: Each chromosome encodes a whole classifier set. Credit is assigned to the complete set of rules via interaction with the environment. Crossover serves to provide a new combination of rules and mutation provides new rules. In some cases, variable-length classifier sets are used, employing modified genetic operators for dealing with these variable-length and position independent genomes.
>
> This model was initially proposed by Smith in 1980 [38]. Recent instances of this approach are the GABIL [13] and GIL [31] systems.

As mentioned in [12], the Michigan approach will prove to be most useful in an on-line, real-time environment in which radical changes in behaviour cannot be tolerated, whereas the Pittsburgh approach will be more useful for off-line environments in which more leisurely exploration and more radical behavioral changes are acceptable.

As commented in [4], the roles of the GAs in the Pittsburgh and Michigan approaches are rather different, and the distinction arises from the difference in the level at which the GAs are applied. Both approaches, at least in their simplest forms, suffer from distinct known problems which arise from the different way in which the GA is applied.

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 problem. 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. Such a 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 and 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.

As compared to the two classic models (the Michigan and Pittsburgh ones), in recent literature we may find different algorithms that use a new learning model based on GAs, the *iterative rule learning approach*. In the latter model, as in the Michigan one, each chromosome in the population represents a single rule, but contrary to the latter, only the best individual is considered as 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. This model has been used in papers such as [42, 16, 18, 19, 24, 25, 9] and attempts to reduce the search space for the possible solutions.

In order to obtain a set of rules, which will be a true solution to the problem, the GA 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 is adequate to represent the examples in the training set, 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, consists of eliminating from the training set all those examples that are covered by the set of rules obtained previously. Some learning algorithms not based on GAs, such as those in the AQ family or the CN2 algorithm [5], use this way of penalizing rules.

This learning way 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 reduces substantially the search space, because in each sequence of iterations only one rule is searched.

In the literature we can find some genetic learning processes that use this model such as *SLAVE* [18], *SIA* [42] and the *genetic generation process* proposed in [24]. These three genetic learning processes use the *iterative rule learning approach* with light difference:

- *SLAVE* launches a new GA to find a new rule after having eliminated the examples covered by the last rule obtained. SLAVE was designed to work with or without linguistic information.
- *SIA* uses a single GA that goes on detecting rules and eliminating the examples covered by the latter. SIA can only work with crisp data.
- The *genetic generation process* runs a GA for obtaining the best rule according to different features, assigns a relative covering value to every example, and removes the examples with a covering value greater than a constant.

From the description above, we may see that in order to implement a learning algorithm based on GAs using the *iterative rule learning approach*, we need, at least, the following:

1. a criterion for selecting the best rule in each iteration,
2. a penalization criterion, and
3. a criterion for determining when enough rules are available to represent the examples in the training set.

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 we can say that all the examples in the training set are sufficiently covered and no more rules are needed to represent them.

4 Genetic Fuzzy Systems

In this Section we briefly introduce the GFSs and present a fuzzy rule genetic learning process, the MSGFS for learning either RBs or KBs in different stages, generating the fuzzy rules using the *iterative rule learning approach*.

4.1 Regarding Genetic Fuzzy Systems

The GAs' properties make them suitable to be used in order to design and optimize fuzzy systems. The automatic definition of the KB may be considered in many cases as optimization or search processes. The application to the learning and/or tuning of KB has provided fairly promising results.

As mentioned in the introduction, GAs are applied to modify/learning the DB and/or the RB, and it is possible to distinguish three different groups of GFSs depending on the KB components included in the genetic learning process.

Genetic definition of the DB. The tuning of the fuzzy rule membership functions is an important task in the design of fuzzy systems. The tuning method using GAs fits the membership functions of the fuzzy rules dealing with their parameters according to a fitness function. Several methods have been proposed in order to define the DB using GAs, based on the existence of a previously defined RB. Each chromosome involved in the evolution process represents different DB definitions, that is, each chromosome contains a coding of the whole membership functions giving meaning to the linguistic terms. Two possibilities can be considered depending on whether the fuzzy model nature is descriptive or approximative, 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 approximative model. Different approaches are presented in [32, 39, 2, 23].

Genetic derivation of the RB. All the methods belonging to this family are suppose the existence a collection of fuzzy set membership functions giving meaning to the labels, a DB, and learning a rule base. Some approaches are presented in [33, 40, 36, 18, 19].

Genetic learning of the KB. There are many approaches for the genetic learning processes of a complete KB, fuzzy rules and membership functions. We find approaches presenting variable chromosome length, 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. Some approaches are presented in [6, 34, 37, 25, 41, 4, 9].

For a more detailed description see [8], for an extensive bibliography see [7] (section 3.13), and some approaches may be found in [27].

In the following, we present the MSGFS for learning RB or KB based on the *iterative rule learning approach*.

4.2 A Multi-Stage Genetic Fuzzy System

Learning algorithms that use the *iterative rule learning approach* do not envisage any relationship between them in the process for obtaining rules. Therefore, the final set of rules usually needs an a posteriori process that will modify and/or fit the said set. The methodology that is presently applied includes different processes that are not necessarily applied simultaneously. This methodology, which we call *multi-stage genetic fuzzy systems* and has been abbreviated as MSGFS, consists of three component parts:

I A *genetic generation stage* for generating fuzzy rules using the *iterative rule learning approach.*

II A *postprocessing stage* working on the rule set obtained in the previous stage in order to either to refine rules or eliminate redundant rules.

III A *genetic tuning stage* that tunes the membership functions of the fuzzy rules.

We describe these shortly below.

4.2.1 Genetic generation stage

In this stage the *iterative rule learning approach* is used for learning fuzzy rules capable of including the complete knowledge from the set of examples.

A chromosome represents a fuzzy rule, the generation method selects the best rule according to different features included in the fitness function of the GA, features that include general properties of the KB and particular requirements to the fuzzy rule. This features lead to the definition of the covering degree between a rule and an example and the use of the concept of positive and negative examples.

The *iterative rule learning approach* uses a covering method of the set of examples. This covering method assigns a relative covering value to every example, and removes the examples with an adequate covering value, according to a covering criterion.

As we have indicated, this model may be used for learning RB as *SLAVE* [16, 18] and for learning KB as the *genetic generation process* proposed in [24, 25]. In the following we shortly show how both learning algorithms use this approach.

SLAVE is a learning system developed in [16, 18], that uses induction and fuzzy rules for representing knowledge. This learning algorithm obtains a set of rules for describing the consequent variable. The selection of the best rule in each iteration is done by a GA and the goal of this GA is finding the rule that covers the maximum number of positive examples and it satisfies the *weak consistency condition*.

Given the concept of the best rule, the learning algorithm will use it for selecting the set of rules that best describe the examples. Thus, once a class is selected, we obtain the best rule for this class and eliminate the examples covered by this rule and this process is repeated. Two important elements in this cycle must be clarified: the concept of covering when the examples and rules are fuzzy, and the criterion of termination of this cycle, i.e., how we know when the current rule set is sufficient for describing a class. In the first problem, SLAVE uses a concept of partial covering based on a λ parameter and for the second problem it uses the definition of *weak completeness condition* proposed in [18]. The GA is used for selecting the best rule in each iteration of the learning process and this GA and its parameters are described in [17]. The goal of the GA is to return the rule with the maximum number of positive examples satisfying the weak consistency condition. Two different definitions have been proposed on this condition in [18], *the k-consistency condition* and *the k_1k_2-consistency condition*.

The **genetic generating process** proposed in [24, 25] generates fuzzy rules with a free semantic, without any initial referential set of fuzzy sets in the universes of discourse, learning the fuzzy rules and the associated fuzzy sets. It is developed by means of a real coded GA (RCGA), where a chromosome represents a fuzzy rule, and it is evaluated by means of a frequency method. The RCGA finds the best rule in every running from the set of examples according to different features that are included in the fitness function of the GA: High *frequency value*, High *average covering degree over positive examples*, Small *negative examples set*, Small *membership functions width*, and High *symmetrical membership functions*.

The covering method is developed as an iterative process that permits a set of fuzzy rules to be obtained covering the set of examples. In each iteration, it runs the RCGA choosing the best chromosome, assigns the relative covering value to every example and removes the examples with a covering value greater than a value ϵ. It finishes when the set of examples is empty.

An additional condition, the *High niche condition rate*, has been included in [10] for maintaining a suitable interaction between neighbour rules by sharing their fitness payoff. In [9, 11] other versions of the method are presented where the rules have their semantic within performance intervals established by a fuzzy partition membership functions.

The advantage of this approximative representation (free semantic) is its expressive power for learning rules which present its own specificity in terms of the fuzzy sets involved in it.

4.2.2 Postprocessing stage: selection and refinement

As we mentioned earlier, the *iterative rule learning approach* does not analyze any relationship between the rules that it is obtaining. That is why, once the rule base has been obtained, it may be improved either because there are rules that may be refined or redundant rules if high degrees of coverage are used. Two possible post-processing methods are briefly introduced below, a refinement algorithm and a selection or simplification algorithm.

A Refinement algorithm

This algorithm, proposed in [20], is basically composed by a heuristic process of generation, specification, addition and elimination of rules. The module is composed of the following tasks: The first one, consists of improving the correctness of each rule. For this purpose, a specification process is used, trying to make each rule cover the highest number of well-classified examples from the original rule without covering its badly-classified examples. After this task, it is possible that some badly-classified examples covered by some rules turn into unclassified examples. The next task tries to cover these unclassified examples using a general ization process over the existent rules or adding new rules. The last task in the refinement process uses a special generalization process for determining each rule, the antecedent variables that are relevant for representing the objects from a class. The previous tasks are repeated on the rule set until a termination condition is satisfied.

The refinement uses a heuristic function and a hill climbing strategy for selecting the most promising action in each step of the algorithm toward a good solution. A function is considered that measures the global precision of the current rule set on the training set. Thus, in order to define this function it is necessary to describe the predictive module used. The inference process begins with an ordered rule set and the classification of an example is done in the following way: the adaptation between the example and the antecedent part of each rule is evaluated and the class of the rule with the best adaptation is returned. If there are some rules with the best adaptation (conflict problem), the class from the rule with the lowest order in the rule set is returned. Thus, it is necessary to establish a priori criterion of relevance between the rules for sorting them. The refinement algorithm uses the same order returned by SLAVE. Basically, this criterion is the following: the most relevant rules are those that removed the highest number of examples in the learning process. In this sense, the most relevant rules are in the first positions and the least relevant rules are in the last positions. The heuristic component of the refinement algorithm selects rules through the order previously described for the rule set. However, no special ordering is considered for variables or values. They are taken by considering the default order.

In [20] each one of the steps of the refinement algorithm are described.

This refinement algorithm has been successfully applied together with SLAVE and it improved the rule set obtained from SLAVE and simplified the problem of choosing parameters in the learning algorithm [20]. The refinement algorithm has been also successfully applied to other learning algorithms [21].

A Selection algorithm

Due to the iterative nature of the *genetic generation process*, redundant rules may appear. This occurs when some examples are covered to a higher degree than the desired one and it makes the RB obtained perform worse due to the existence of redundant rules. In order to solve this problem and improve its accuracy, it is necessary to simplify the rule set obtained from the previous process for deriving the final RB.

The simplification process was proposed in [25]. It is based on a binary coded GA, where the coding scheme maintains fixed-length chromosomes. Considering the rules contained in the RB counted from 1 to m, an m-bit string $C = (c_1, ..., c_m)$ represents a subset of candidate rules to form the RB finally obtained as this stage's output, B^s, such that,

$$\textit{If } c_i = 1 \textit{ then } R_i \in B^s \textit{ else } R_i \notin B^s \ .$$

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

Regarding the fitness function, $E(\cdot)$, it is based on an application specific measure usually employed in the design of GFSs, either the medium square error (SE) over a training data set, E_{TDS}, in control problems or the percentage of classified examples in classification problems.

For example, in the case of designing fuzzy logic controllers (FLC), it may be represented by the following expression:

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

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

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

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

where $R_j(e_l)$ is the compatibility degree between the rule and the example, and τ is the minimal training set completeness degree accepted in the simplification process.

Therefore, a *training set completeness degree* of $R(C_j)$ over the set of examples E_{TDS} is defined as

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

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

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

This selection algorithm has been applied together with the aforementioned *genetic generation process* and improves the rule set obtained from it [25, 9].

4.2.3 Genetic tuning stage

At this stage *the genetic tuning process* is applied over the KB for obtaining a more accurate one.

We can consider two possibilities, depending on the fuzzy model's nature:

a) an approximative model based on a KB composed of a collection of fuzzy rules without a fixed relationship between the fuzzy rules and some primary fuzzy partitions giving meaning to them, or

b) a descriptive model based on a linguistic description of the system with a fuzzy partition that assigns a membership function to every linguistic label.

In both cases, each chromosome forming the genetic population will encode a complete DB, but in the first case each piece of chromosome codes the membership functions associated to one rule and in the second one each piece of chromosome codes the fuzzy partition of a variable.

We can use RCGAs where every variable value is a gene, this GA and its components are described in [23, 26]. The main difference between both processes is the coding scheme. They are described below.

Approximative scheme

Each chromosome forming the genetic population encodes a complete KB, each one of them contains the RB with a different DB associated [23].

If we consider an MISO control system where the KB consists of a collection of fuzzy rules describing the action with the form:

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

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

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

Each one of the rules will be encoded in pieces of chromosome C_{ri}, $i = 1, \ldots, m$, in the following way:

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

Therefore the KB is represented by a complete chromosome C_r:

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

In [23] a complete description of this approximative genetic tuning process is to be found.

Descriptive scheme

A modified version of the genetic tuning method presented in [23] is applied. Each chromosome forming the genetic population encodes a complete fuzzy partition of the variables.

As we have, already stated, the primary fuzzy sets considered in the initial linguistic fuzzy partitions are triangle-shaped. Thus, each one of the membership functions has associated a parametric representation based on a 3-tuple of real values and a primary fuzzy partition can be represented by an array composed by $3N$ real values, with N being the number of terms forming the linguistic variable term set. The complete DB for a problem in which n linguistic variables are involved is encoded into a fixed length real coded chromosome C_r, built by joining the representation of each one of the variable fuzzy partitions as is shown in the following:

$$C_{ri} = (a_{i1}, b_{i1}, c_{i1}, \ldots, a_{iN_i}, b_{iN_i}, c_{iN_i}).$$
$$C_r = C_{r1}\ C_{r2}\ ...\ C_{rn}.$$

In [11] the application of both *genetic tuning processes* is to be found.

5 Concluding Remarks

In this paper, we have presented the *iterative rule learning approach* as an alternative model to the classical Michigan and Pittsburgh approaches for the design of genetic learning processes, and we have described how it can be applied within a *multi-stage learning process.*

We have introduced the GFSs, and presented the possible steps of an *MSGFS* for learning RBs or KBs based on the *iterative rule learning approach.* It is a general genetic learning context for fuzzy systems where different GFS processes can be de signed. The advantage of this general context is that in the first stage considerably reduces the space of search because it look for only one fuzzy rule in each sequence of iterations, and in stages two and three provides tools that can improve the RB and DB, respectively.

We can conclude pointing out that we have presented a general working methodology in the genetic learning of fuzzy rules that attempts to generalize and place into a common context the different learning processes that have been developed till now.

References

[1] Baker, J.E., Reducing Bias and Inefficiency in the Selection Algorithm. Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum, Hillsdale, NJ, 1987, 14-21.

[2] Bolata, F., Nowè, A., From Fuzzy Linguistic Specifications to Fuzzy Controllers using Evolution Strategies. Proc. Fourth IEEE Int. Conference on Fuzzy Systems, Yokohama, 1995, 1089-1094.

[3] Booker, L.B., Goldberg, D.E., Holland, J.H., Classifier Systems and Genetic Algorithms. Artificial Intelligence 40 (1989) 235-282.

[4] Carse, B., Fogarty, T.C., Munro, A., Evolving Fuzzy Rule Based Controllers Using Genetic Algorithms. Fuzzy Sets and Systems 80 (1996) 273-293.

[5] Clark P., Niblett T., Learning If Then Rules in Noisy Domains. TIRM 86-019, The Turing Institute, Glasgow, 1986.

[6] Cooper, M., Vidal, J.J., Genetic Design of Fuzzy Controllers: the Cart and Jointed Pole Problem. Proc. Third IEEE Int. Conf. on Fuzzy Systems, Orlando, 1994, 1332-1337.

[7] Cordón, O., Herrera, F., Lozano, M., A Classified Review on the Combination Fuzzy Logic-Genetic Algorithms Bibliography. Tech. Report $\#DECSAI-$ 95129, Dept. of Computer Science and A.I., University of Granada, 1995. Available at the URL address: http://decsai.ugr.es/~herrera/fl-ga.html

[8] Cordón, O., Herrera, F., A General Study on Genetic Fuzzy Systems. In: G. Winter, J. Periaux, M. Galan, P. Cuesta (Eds.), Genetic Algorithms in Engineering and Computer Science, Wiley and Sons, 1995, 33-57.

[9] Cordón, O., Herrera, F., A Hybrid Genetic Algorithm-Evolution Strategy Process for Learning Fuzzy Logic Controller Knowledge Bases. In: F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, 1996, 251-278.

[10] Cordón, O., Herrera, F., Fuzzy Identification by Means of Genetic Algorithms. In: H. Hellendoorn, D. Driankov (Eds.). Fuzzy Model Identification, Springer-Verlag, 1997.

[11] Cordón, O., Herrera, F., A Three-Stage Evolutionary Process for Learning Descriptive and Approximative Fuzzy Logic Controller Knowledge Bases from Examples. International Journal of Approximate Reasoning, 17:4, 1997.

[12] De Jong, K.A., Learning with Genetic Algorithms: An Overview. Machine Learning 3 (1988) 121-138.

[13] De Jong, K.A., Spears, W.M., Gordon, F.D., Using Genetic Algorithms for Concept Learning. Machine Learning 13 (1993) 161-188.

[14] Giordana, A., Neri, F., Genetic Algorithms in Machine Learning. AI Communications 9 (1996) 21-26.

[15] Goldberg, D.E., Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, 1989.

[16] González, A., Pérez, R., Verdegay, J.L., Learning the Structure of a Fuzzy Rule: a Genetic Approach. Fuzzy System and Artificial Intelligence 3 (1994) 57-70.

[17] González, A., Pérez, R., Structural Learning of Fuzzy Rules from Noisy Examples. Proc. FUZZ-IEEE/IFES'95, Yokohama, vol.III, (1995), 1323-1330.

[18] González, A., Pérez, R., Completeness and Consistency Conditions for Learning Fuzzy Rules. Fuzzy Sets and Systems, 1997. To appear.

[19] González, A., Pérez, R., A Learning System of Fuzzy Control Rules. In: F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing, Physica-Verlag, 1996, 202-225.

[20] González, A., Pérez, R., A Refinement Algorithm of Fuzzy Rules for Classification Problems. Proc. of the Sixth Int. Conference on Information Procesing and Management of Uncertainty in Knowledge Based Systems, Granada, 1996, 533-538.

[21] González, A., Pérez, R., Aplicación de un Sistema de Refinamiento de Reglas a Problemas de Clasificación. Proceedings of IBERAMIA'96, Mexico, 1996, 20-29.

[22] Grefenstette, J.J. (Ed.), Genetic Algorithms for Machine Learning. Kluwer Academic, 1994.

[23] Herrera, F., Lozano, M., Verdegay, J.L., Tuning Fuzzy Logic Controllers by Genetic Algorithms. International Journal of Approximate Reasoning 12 (1995) 299-315.

[24] Herrera, F., Lozano, M., Verdegay, J.L. Generating Rules from Examples using Genetic Algorithms. In: B. Bouchon, R. Yager, L. Zadeh (Eds.), Fuzzy Logic and Soft Computing, Word Scientific, 1995, 11-20.

[25] Herrera, F., Lozano, M., Verdegay, J.L., A Learning Process for Fuzzy Control Rules using Genetic Algorithms. Fuzzy Sets and Systems 1997. To appear.

[26] Herrera, F., Lozano, M., Verdegay, J.L., Fuzzy Connectives based Crossover Operators to Model Genetic Algorithms Population Diversity. Fuzzy Sets and Systems, 92 (1997) 21-30.

[27] F. Herrera, J.L. Verdegay (Eds.), Genetic Algorithms and Soft Computing. Physica-Verlag, 1996.

[28] Holland, J.H, Adaptation in Natural and Artificial Systems. Ann Arbor, 1975.(MIT Press, 1992).

[29] Holland, J.H., Reitman, J.S., Cognitive Systems Based on Adaptive Algorithms. In D.A. Waterman, F. Hayes-Roth (Eds.), Pattern-Directed Inference Systems Academic Press, New York, 1978.

[30] Holland, J.H., Escaping Britleness: The Possibilities of General Purpose Learning Algorithms Applied to Parallel rule-Based Systems. In: R. Michalski, J. Carbonell, T. Michel (Eds.), Machine Learning: An AI Approach, Vol. II, Morgan-Kaufmann, 1986, 593-623.

[31] Janikow, C.Z., A Knowledge Intensive Genetic Algorithm for Supervised Learning. Machine Learning 13 (1993) 198-228.

[32] Karr, C., Genetic Algorithms for Fuzzy Controllers. AI Expert 6 (1991) 26-33.

[33] Karr, C., Applying Genetic Algorithms to Fuzzy Logic. AI Expert 6 (1991) 38-43.

[34] Lee, M.A., Takagi, H., Embedding Apriori Knowledge into an Integrated Fuzzy System Design Method Based on Genetic Algorithms. Proc. Fifth Int. Fuzzy Systems Association World Congress, Seoul, 1993, 1293-1296.

[35] Michalewicz, Z., Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, 1992.

[36] Pham, D.T., Karoboga, D., Optimum Design of Fuzzy Logic Controllers using Genetic Algorithms. J. Systems Engineering 1 (1991) 114-118.

[37] Satyadas, A., Krishanakumar, K., GA-Optimized Fuzzy Controller for Spacecraft Attitude Control. Proc. Third IEEE International Conference on Fuzzy Systems, Orlando, 1994, 1979-1984.

[38] Smith, S.F., A Learning System Based on Genetic Adaptive Algorithms. Ph. D. Thesis, University of Pittsburgh, 1980.

[39] Surmann, H., Kanstein, A., Goser, K., Self-Organizing and Genetic Algorithms for an Automatic Design of Fuzzy Control and Decision Systems. Proc. First Europen Congress on Fuzzy and Intelligent Technologies, Aachen, 1993, 1097-1104.

[40] Thriff, P., Fuzzy Logic Synthesis with Genetic Algorithms. Proc. Fourth International Conference on Genetic Algorithms, San Diego, 1991, 509-513.

[41] Velasco, J.R., Magdalena, L., Genetic Learning Applied to Fuzzy Rules and Fuzzy Knowledge Bases. Proc. Sixth Int. Fuzzy Systems Association World Congress, Sao Paulo, 1995, 257-260.

[42] Venturini, G., SIA: a Supervised Inductive Algorithm with Genetic Search for Learning Attribute Based Concepts. Proc. European Conference on Machine Learning, Vienna, 1993, 280-296.