Genetic Algorithms: Optimization, Search and Learning

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

Genetic algorithms play a significant role, as search techniques for handling complex spaces, in many fields such as artificial intelligence, engineering, robotics, etc. Genetic algorithms are based on the underlying genetic process in biological organisms and on the natural evolution principles of populations. A short description is given in this lecture, introducing their use for machine learning.

Key words: Genetic Algorithms, Evolutionary Computation, Learning.

1 Introduction

Evolutionary Computation (EC) uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem solving systems. There are a variety of evolutionary computational models that have been proposed and studied which are referred as *Evolutionary Algorithms* (EAs). Shortly, this paradigm covers several variations, such as *Evolutionary Strategies*, addressing continuous function optimization [72], *Evolutionary Programming* , generating finite state automata that describe strategies or behaviors [25], *Genetic Algorithms*, providing continuous and discrete function optimization and search [31,45] and *Genetic Programming*, evolving computer programs to approximately solve problems [52].

In this lecture we will give a short introduction to the most widely studied EA, Genetic Algorithms, and the use of them for Machine Learning.

2 Genetic Algorithms

Genetic algorithms (GAs) have had a great measure of success in search and optimization problems. The reason for a great part of their 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.

GAs are general purpose search algorithms which use principles inspired by natural genetic populations to evolve solutions to problems [45,31]. The basic idea is to maintain a population of chromosomes, which represent candidate solutions to the concrete problem, that evolves over time through a process of competition and controlled variation. Each chromosome in the population has an associated *fitness* to determine which chromosomes are used to form new ones in the competition process, which is called *selection.* The new ones are created using genetic operators such as *crossover* and *mutation.*

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 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.

Next procedure 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)

The basic principles of GAs were first laid down rigorously by Holland ([45]), and are well described in many books, such as [31,60]. 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.).*

2.1 Applications 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 extensible 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 the 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 following references show monograph books of applications in different areas: engineering and computer science [22,84], machine learning [37,28], pattern recognition [63], neuronal networks [83], robotics [21], investment strategies [5], management applications [6], and fuzzy systems [42,67,70].

3 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 ([48,37,30]).

Three alternative approaches, in which GAs have been applied to learning processes, have been proposed, the Michigan ([47]), the Pittsburgh ([74]), and the Iterative Rule Learning (IRL) approaches [32,82]. 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. Below, we will describe them briefly.

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.

A genetic learning process based on the Michigan approach receives the name of Classifier System. A complete description is to be found in [12].

Pittsburgh approach. Each chromosome encodes a whole rule sets. 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.

This model was initially proposed by Smith in 1980 [74]. Recent instances of this approach may be found in [37].

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 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.

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.

A more detailed description of this approach may be found in [32].

3.1 Some Remarks

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 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 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.

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.

4 Concluding Remarks

A short introduction of GAs have been presented. Regarding to their use for machine learning, to point out that GAs are also used for refining parameters in other learning approaches, as is done using GAs for determining weights in a neural network.

Integrating Genetic Algorithms and Fuzzy Logic

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

In this lecture, an evaluation of the current situation regarding to the combination of Genetic Algorithms and Fuzzy Logic is given. This is made by means of a classification in areas, giving a short introduction to each one of them.

Key words: Fuzzy Logic, Genetic Algorithms.

1 Introduction

Recentely, numerous papers and applications combining Fuzzy Logic (FL) and Genetic Algorithms (GAs) have become known, and there is an increasing interest in the integration of these two topics.

In the following we explore this combination from the bidirectional integration:

- the use of FL based techniques for either improving GA behaviour and modeling GA components, the results obtained have been called *fuzzy genetic algorithms* (FGAs), and
- the application of GAs in various optimization and search problems involving fuzzy systems.

The present lecture tries to give a short review of the combination of FL and GAs, introducing a classification of the publications in fourteen areas, presenting briefly them.

Before to introduce the aforementioned areas, a few remarks seem to be necessary.

- The first is regarding to the bibliography. It is collected in our technical report *O. Cordón, F. Herrera, M. Lozano, "A Classified Review on the Combination Fuzzy Logic-Genetic Algorithms Bibliography", Dept. of Computer Science and A.I., University of Granada, Tech.Report 95129, October 1995 (Last version December 1996). Available at the URL address: http://decsai.ugr.es/~herrera/fl-ga.html.* It classifies and lists 562 references. This report classifies the bibliography in 15 sections according to the following table. It contains the keywords and the number of papers on each of them. These keywords covers the application of FL based tools to GAs (with the name of fuzzy genetic algorithms) and the different areas of FL and fuzzy set theory where GAs have been applied. The underlying report is continuously being updated.

1	Fuzzy genetic algorithms	24
2	Fuzzy clustering	14
3	Fuzzy optimization	39
4	Fuzzy neural networks	34
5	Fuzzy relational equations	6
6	Fuzzy expert systems	8
7	Fuzzy classifier systems	33
8	Fuzzy information retrieval and database quering	6
9	Fuzzy decision making, financial, and economic models	10
10	Fuzzy regression analysis	6
11	Fuzzy pattern recognition and image processing	24
12	Fuzzzy classification - Concept Learning	24
13	Fuzzy logic controllers (Design, Learning, Tuning, Applications)	287
14	Fuzzy logic - Genetic algorithms framework	13
15	Fuzzy logic miscellaneous	38

Table 1. Classification keywords

- The second, is regarding to this lecture.It is a summary of the contribution: *O. Cordón, F. Herrera, M. Lozano, "On the Combination of Fuzzy Logic and Evolutionary Computation: A Short Review and Bibliography", In: Fuzzy Evolutionary Computation. W. Pedrycz (Ed.), Kluwer Academic Pub., 1997, pp. 33-56.*

We consider fourteen areas, we join the table areas 7 and 13 in a global area with the name *Genetic fuzzy rule-based control systems*. In the following we describe the classification areas. The exhaustive bibliography is found in the aforementioned reference.

2 Classification Areas

In this section we introduce a description of every area and describe shortly theapplication of GAs to them.

Fuzzy genetic algorithms. A Fuzzy Genetic Algorithm (FGA) is considered as a GA that uses fuzzy logic based techniques or fuzzy tools to improve the GA behaviour modeling different GA components.

An FGA may be defined as an ordering sequence of instructions in which some of the instructions or algorithm components may be designed with fuzzy logic based tools, such as, fuzzy operators and fuzzy connectives for designing genetic operators with different properties, fuzzy logic control systems for controlling the GA parameters according to some performance measures, fuzzy stop criteria, representation tasks, etc.

Fuzzy clustering. Clustering plays a key role in searching for structures in data. Given a finite set of data, X, the problem of clustering in X is to find several cluster centers that can properly characterize relevant classes of X. In classical cluster analysis, these classes are required to form a partition of X such that the degree of association is strong for data within blocks of the partition and weak for data in different blocks. However, this requirement is too strong in many practical applications, and it is thus desirable to replace it with a weaker requirement. When the requirement of a crisp partition of X is replaced with a weaker requirement of a *fuzzy partition* or a *fuzzy pseudopartition* on X, the emerged problem area is refered as *fuzzy clustering.*

GAs are used for a global search of the space of possible data partitions given a choice of the number of clusters or classes in the data, for determining the number of clusters, etc.

Fuzzy optimization. Fuzzy optimization deals with how to find a best point under some fuzzy goals and restrictions given as linguistic terms or fuzzy sets.

GAs are used for solving different fuzzy optimization problems. This is the case for instance of fuzzy flowshop scheduling problems, vehicle routing problems with fuzzy due-time, fuzzy mixed integer programming applied to resource distribution, interactive fuzzy satisfycing method for multiobjective 0-1, fuzzy optimal reliability design problems, job-shop scheduling problem with fuzzy processing time, fuzzy optimization of distribution networks, etc.

Fuzzy neural networks. Neural networks have been recognized as an important tool for constructing membership functions, operations on membership functions, fuzzy inference rules, and other context-dependent entities in fuzzy set theory.

On other hand, attempts have been made to develop alternative neural networks, more attuned to the various procedures of approximate reasoning. These alternative neural networks are usually referred to as *fuzzy neural networks*. The following features, or some of them, distinguish fuzzy neural networks from their classical counterparts: inputs are fuzzy numbers, outputs are fuzzy numbers, weights are fuzzy numbers, weighted inputs of each neuron are not aggregated by summation, but by some other aggregation operation. A deviation from classical neural networks in any of these features requires a properly modified learning algorithm to be developed.

GAs are used for designing an overall good architecture of fuzzy neural networks and fuzzy neural networks, for determining an optimal set of link weightd, for participating in hybrid learning algorithms, etc.

Fuzzy relational equations. The notion of fuzzy relational equations is associated with the concept of composition of binary relations. This operation involves exactly the same combinations of matrix entries as in the regular matrix multiplication. However, the multiplications and additions that are applied to these combinations in the matrix multiplication are replaced with other operations. These alternative operations represent, in each given context, the appropriate operations of fuzzy set intersection and union, respectively. Fuzzy relational equations have been intensively exploited in many areas of applications of fuzzy sets.

GAs may be used either for finding approximate solutions to a system of fuzzy relational equations or for learning in relational structures.

Fuzzy expert systems. An expert system is a computer-based system that emulates the reasoning process of a human expert within a specific domain of knowledge. In fuzzy expert systems, the knowledge is usually represented by a set of fuzzy production rules, which connect antecedents with consequent, premises with conclusions, or conditions with actions.

GAs can solve two basical problems of the knowledge base, the knowledge base buliding and the knowledge filtering.

Fuzzy information retrieval Information retrieval may be defined as the problem of the selection of documentary information from storage in response to search questions. The motivation of the application of fuzzy set theory to the design of databases and information storage and retrieval systems lies in the need to handle imprecise information. The database that can accommodate imprecise information can store and manipulate not only precise facts, but also subjective expert opinions, judgments, and values that can be specified in linguistic terms.

GAs are used for designing models for optimization of queries in a fuzzy information retrieval system.

Fuzzy decision making, financial, and economic models. Decision making is the study of how decisions are actually made and how they can be made better or more successfully. Fuzzy set theory has been widely used in the field of decision making. For the most part, the application consisted on fuzzifications of the classical theories of decision making. Also it is used for modelling some financial and economic problems.

GAs are used for cooperating in the design and resolution of these models.

Fuzzy regression analysis Regression analysis is an area of statistics that deals with the investigation of the dependence of a variable upon one or more other variables. Two distinct motivations, fuzzy relation seems intuitively more realistic and the nature of data which in some applications are inherently fuzzy, lead to two types of fuzzy regression analysis. One involves fuzzy parameters and crisp data, while the other one involves crisp parameters and fuzzy data.

GAs are used for solving the underlying optimization problems.

Fuzzy pattern recognition and image processing. There are various aspects of image processing and analysis problems where the theory of fuzzy sets has been applied: as generalizations of classical membership-roster methods, generalizations of classical syntactic methods, providing image ambiguity/information measures and quantitative evaluation, computing fuzzy geometrical properties, etc.

In handling uncertainty in pattern analysis, GAs may be helpful in determining the appropriate membership functions, rules and parameter space, and in providing a reasonably suitable solution. For this purpose, a suitable fuzzy fitness function needs to be defined depending on the problem.

Fuzzy classification - Concept learning.

Fuzzy classification sytems based on fuzzy logic are capable of dealing with cognitive uncertainties such as the vagueness and ambiguity involved in classification problems. In a fuzzy classification system, a case or an object can be classifed by applying (mainly) a set of fuzzy rules based on the linguistic values of its attributes.

GAs are used in a fuzzy classification system for learning fuzzy rules, membership functions, fuzzy partitions, etc.

Genetic fuzzy rule based control systems. Fuzzy rule based systems have been shown to be an important tool for modelling complex systems in which, due to the complexity or the imprecision, classical tools are unsuccessful.

GAs have demonstrated to be a powerful tool for automating the definition of the Kownledge Base of a Fuzzy Controller since adaptive control, learning, and self-organization may be considered in a lot of cases as optimization or search processes. Their adavantages have extended the use of GAs in the development of a wide range of approaches for designing Fuzzy Controllers over the last few years. In particular, the application to the design, learning and tuning of KBs has produced quite promising results. These approaches can receive the general name of *Genetic Fuzzy Systems* (GFSs). On other hand, we also must understand the GFSs as the application of GAs to any fuzzy system being the fuzzy rule based systems a particular case although the most extended, this is the reason of calling this area as *genetic fuzzy rule based control systems*.

3 Concluding Remarks

After the short description of areas, to point out that the use of fuzzy logic techniques permits GA behaviour to be improved in different ways, as well as emphasize the potential of GAs in fuzzy environments as a flexible tool for optimization and search.

Finally, to mention six references. Two of them are technical reports that collect bibliography on the combination of GAs and FL [1,19], the third reference is the paper basis for this summary [20], and the last three references, the three edited books [42,67,70], present a collection of papers dealing with the topic.

Genetic Fuzzy Rule Based Systems

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

The search capabilities and ability for incorporating a priori knowledge have extended the use of Genetic Algorithms 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.

In this lecture we focus our presentation on genetic fuzzy rule-based systems.

Key words: Genetic Algorithms, Fuzzy Rule Based Systems, Learning, Tuning.

1 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 ([66,85]).

Recently, a great number of publications explore the use of Genetic Algorithms (GAs) for designing fuzzy systems. These approaches receive the general name of *Genetic Fuzzy Systems* (GFSs).

The automatic definition of an FS can be considered in many 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 in 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.

We shall center this lecture on Fuzzy Rule Based Systems (FRBSs), [2], 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 ([23]). 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. GAs may be applied to adapting/learning the DB and/or the RB of an FRBS. This tutorial will summarize and analyze the GFSs, paying a special attention to FRBSs incorporating tuning/learning through GAs.

This lecture presents some characteristics of genetic fuzzy rule based systems.

2 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 an FRBS to a controlled system in order to obtain a certain overall behavior. Next, we analyze some aspects of the Genetic FRBSs.

2.1 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 defining the DB. In learning problems, a more elaborate process including the modification of the RB is performed. We can distinguish between three different groups of GFSs depending on the KB components included in the genetic learning process.

For an extensive bibliography see [19] (section 3.13), some approaches may be found in [32].

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.

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.

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.

2.2 The Keys to the Tuning/Learning Process

Regardless of the kind of optimization problem, i.e., given a system to be modeled/controlled (hereafter we use this notation), 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 knowledge, 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 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. 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.

The process. Summarizing the points that characterize a specific learning process, these 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.

2.3 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)* ([10]). The difficulty of solving the introduced problem depends directly on the genetic learning approach followed by the GFS (Michigan, Pittsburgh or IRL approaches). Below we briefly analyze them.

Michigan approach. It is difficult to solve the CCP when working with the Michigan approach. In this case, the evolution is performed at the level of fuzzy rules instead of at the level of KBs and it is not easy to obtain an adequate cooperation between fuzzy rules that are competing with one another. To do this, we need a fitness function able to measure both the goodness of a single fuzzy rule and the quality of its cooperation with the other fuzzy rules in the population to give the best action as output. As mentioned in [10], the design of a fitness function of this kind is not an easy task.

Pittsburgh approach. This approach is able to solve adequately the CCP. When using this approach, the GFS evolves populations of KBs and the fitness function associated to each individual is computed taking into account the real action that the FS encoded into the chromosome should give as output when it receives a concrete input. Thus, each time an individual is evaluated, the cooperation between the fuzzy rules composing the KB is measured, so the GFS is able to evolve adequately the population to obtain the FS presenting the best possible cooperation between the fuzzy rules composing its KB. Unfortunately, this approach presents the drawback of having to deal with very large search spaces, which makes it difficult to find optimal solutions. This drawback is usual when designing GFSs belonging to the third family, i.e., when the generation of the whole KB is considered in the genetic learning process. In this case, a large quantity of KB parameters have to be included in the genetic representation, which therefore becomes larger. This fact will be more pronounced if an approximate fuzzy model is considered, the use of different membership function definitions for each rule makes the number of KB parameters increase, and then the search space becomes more complex, making the problem computationally hard.

IRL approach. Finally, GFSs based on the IRL approach try to solve the CCP at the same time reducing the search space by encoding a single fuzzy rule in each chromosome. To put this into effect, these processes follow the usual problem partitioning working way and divide the genetic learning process into, at least, two stages. Therefore, the CCP is solved in two steps acting at two different levels, with the competition between fuzzy rules in the first one, the genetic generation stage, and with the cooperation between these generated fuzzy rules in the second one, the post-processing stage.

3 Concluding Remarks

In this lecture we have introduced the GFSs, presenting the basic keys to the tuning/learning processes and the problem of the cooperation vs. competition in the different learning approaches.

Genetic Tuning of Fuzzy Rule Based Systems: Basic Models

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

The tuning of the membership functions is an important task in the design of a fuzzy system. Genetic Algorithms are used for the optimization of membership functions and the scaling functions. This lecture introduces the use of Genetic Algorithms in the tuning of the fuzzy systems.

Key words: Genetic Algorithms, Fuzzy Rule Based Systems, Tuning.

1 Introduction

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 Genetic Algorithms 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 Data Base (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.

In this lecture we analyze the use of GAs for the tuning of DBs according to the two metioned areas, the adaptation of contexts using scaling functions and the tuning of membership functions, we shall present briefly them.

2 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 ([62]) and linear scaling functions ([59]).

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 ([56]) and sigmoidal ([38]) 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 1 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).

3 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. Some approaches are found to be in [8,11,40,50,76].

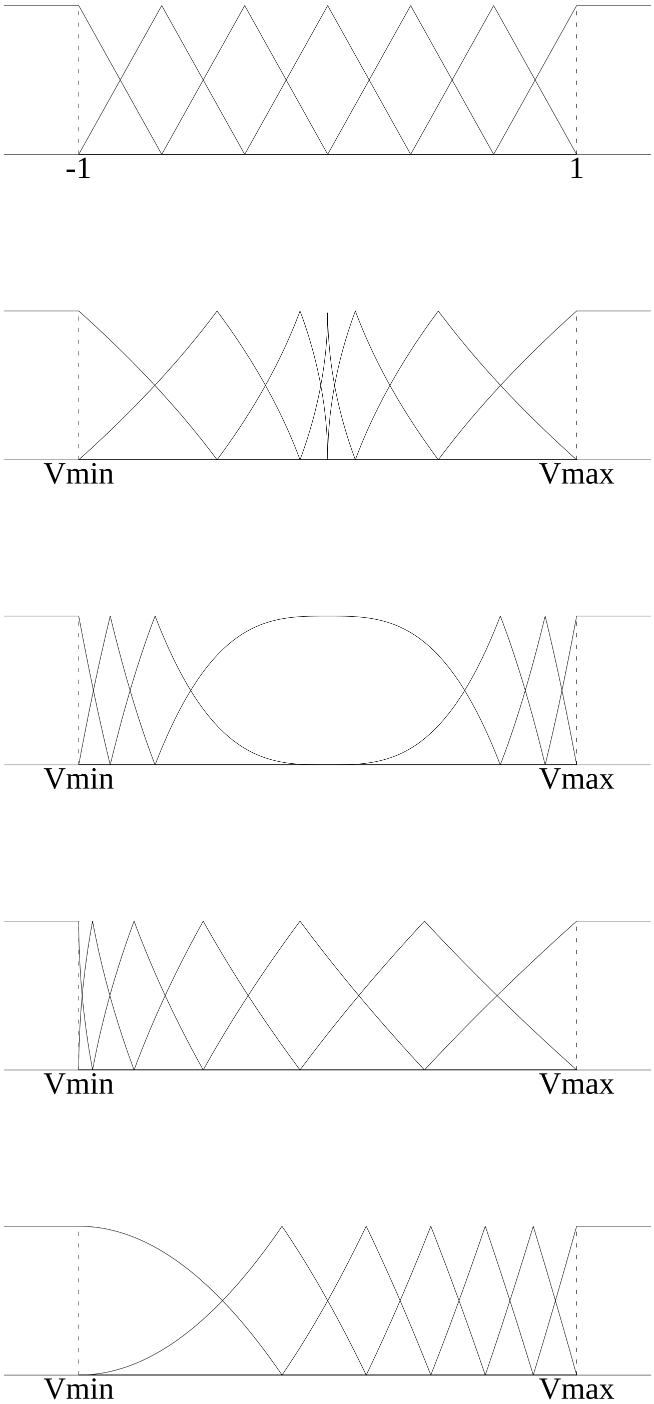

Fig. 1. Nonlinear contexts adaption

The different proposals differ in the coding scheme and the management of the solutions (fitness functions, ...).

3.1 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 ([14,24,49,64]) and other times they are irregular ([53]). A second possibility are trapezoidal membership functions ([51]).

Each parameter of the function constitutes a gene of the chromosome that may be a binary code representing the parameter ([14,49,51,53]) or a real number (the parameter itself, [24,40,64]).

Differentiable functions. Gaussian, bell and sigmoidal are examples of parameterized differentiable functions. These membership functions have been broadly applied in different fuzzy-neural systems ([57]) but radial functions ([26]) and Gaussian functions ([54,71]) are used in GFSs too. To translate the parameters of the function into genetic information a binary code is used in [71,26] and the coefficient itself in [54].

3.2 Scope of the Semantics

The genetic tuning process of membership functions is based on two variants, depending on the fuzzy model nature, whether approximate ([40]) or descriptive ([18,50]).

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.

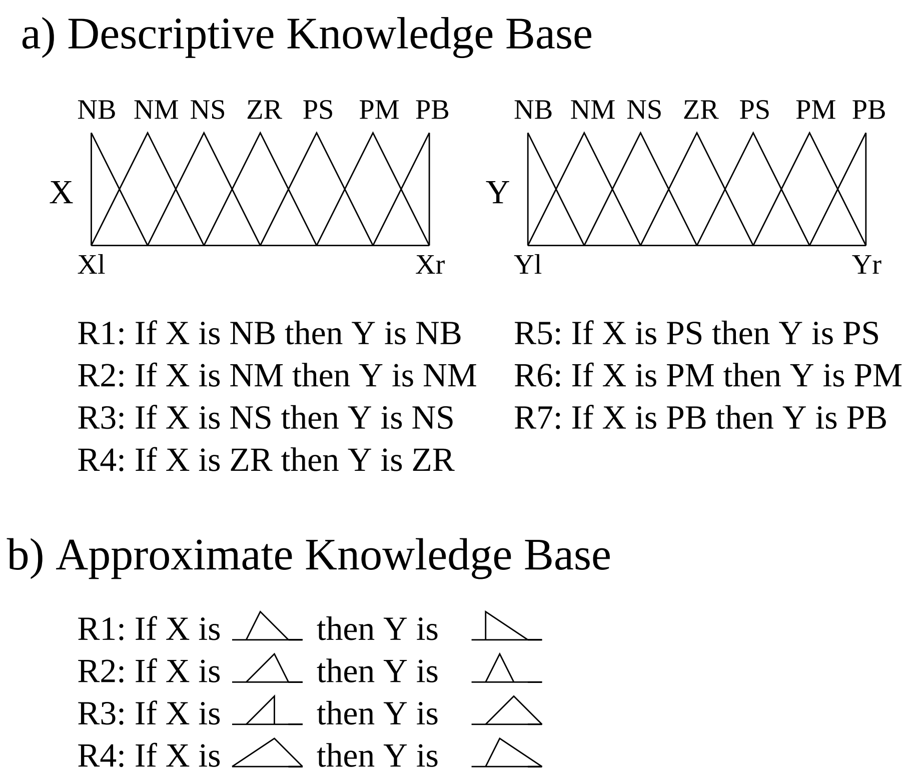

Fig. 2. Descriptive versus Approximate fuzzy models

Figure 2 and the examples described in the following paragraphs illustrate these two variants, and their particular aspects reflected in the coding scheme.

3.3 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, each rule

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

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

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

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

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

$$C_l = C_{l1}\ C_{l2}\ ...\ C_{lm}$$

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

3.4 The Descriptive Genetic Ttuning Process

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

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

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

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

4 Concluding Remarks

This lecture have presented the use of GAs for tuning fuzzy systems. Finally, to point out that the genetic tuning process may be combined with the fuzzy rule learning process for improving the learning capabilities of them.

Learning with Genetic Fuzzy Systems: An Application

Ulrich Bodenhofer

Fuzzy Logic Laboratorium Linz-Hagenberg, Dept. of Mathematics
University of Linz, A-4040 Linz, Austria
E-mail: ulrich@flll.uni-linz.ac.at

Abstract

In this part, an application of genetic tuning of a fuzzy system is presented. First, a fuzzy method for a certain kind of pixel classification is introduced. In the second part, genetic methods for tuning the membership functions of the classification system are discussed and compared with other probilistic optimization methods.

Key words: Hybrid Genetic Algorithm, Pixel classification, Tuning Problem.

1 Introduction

Pixel classification is an important preprocessing task in many image processing applications. In this project, where the FLLL developed an inspection system for a silk-screen printing process, it was necessary to extract regions from the print image which had to be checked by applying different criteria:

(1) Homogeneous area: uniformly colored area;
(2) Edge area: pixels within or close to visually significant edges;
(3) Halftone: area which looks rather homogeneous from a certain distance, although it is actually obtained by printing small raster dots of two or more colors;
(4) Picture: rastered area with high, chaotic deviations, in particular small high-contrasted details.

The magnifications in Figure 3 show how these areas typically look like at the pixel level. Of course, transitions between two or more of these areas are possible, hence a fuzzy model is recommendable.

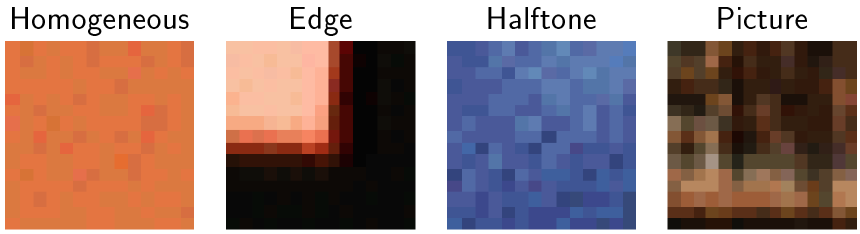

Fig. 3. Magnifications of typical representatives of the four types

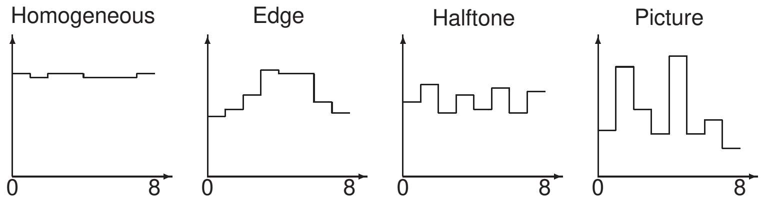

Fig. 4. Typical gray value curves

If we plot one color extraction of the eight neighbor pixels with respect to a clockwise enumeration of the eight neighbors, we typically get curves like those shown in Figure 4. Seemingly, the size of the deviations, e.g., by computing the variance, can be used to distinguish between homogeneous areas, halftones and the other two types. On the other hand, a method which judges the width and connectedness of the peaks should be used in order to separate edge areas from pictures. A simple but effective method for this purpose is the so-called discrepancy norm, for which there are already other applications in pattern recognition (cf. [61]):

$$\|\vec{x}\|_D = \max_{1\leq\alpha\leq\beta\leq n} \left| \sum_{i=\alpha}^{\beta} x_i \right| \tag{1}$$

A more detailed analysis of the discrepancy norm, especially how it can be computed in linear time, can be found in [3].

2 The Fuzzy System

For each pixel (i, j) we consider its nearest eight neighbors enumerated as described above, which yields three vectors of gray values with 8 entries — one for each color extraction. As already mentioned, the sum of the variances of the three vectors can be taken as a measure for the size of the deviations in the neighborhood of the pixel. Let us denote this value with $v(i, j)$. On the other hand, the sum of the discrepancy norms of the vectors, where we subtract each entry by the mean value of all entries, can be used as a criterion whether the pixel is within or close to a visually significant edge.

The fuzzy decision is then carried out for each pixel (i, j) independently: First of all,

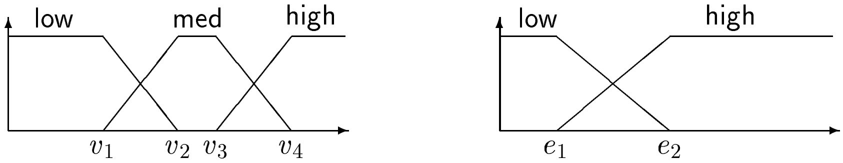

Fig. 5. The linguistic variables v and e

the characteristic values $v(i,j)$ and $e(i,j)$ are computed. These values are taken as the input of a small fuzzy system with two inputs and one output. Let us denote the linguistic variables on the input side with v and e. Since the position of the pixel is of no relevance for the decision in this concrete application, indices can be omitted here. The input space of the variable v is represented by three fuzzy sets which are labeled "low", "med", and "high". Analogously, the input space of the variable e is represented by two fuzzy sets, which are labeled "low" and "high". Experiments have shown that $[0, 600]$ and $[0, 200]$ are appropriate universes of discourse for v and e, respectively. For the decomposition of the input domains simple Ruspini partitions (see [69]) consisting of trapezoidal fuzzy subsets were chosen. Figure 5 shows how these partitions typically look like.

The output space is a set of linguistic labels, namely "Ho", "Ed", "Ha", and "Pi", which are, of course, just abbreviations of the names of the four types. Let us denote the output variable itself with t. Finally, the output of the system for each pixel (i,j) is a fuzzy subset of {"Ho", "Ed", "Ha", "Pi"}. This output set is computed by processing the values $v(i,j)$ and $e(i,j)$ through a rulebase with five rules, which cover all the possible combinations:

IF	v is low			THEN	t = Ho
IF	v is med	AND	e is high	THEN	t = Ed
IF	v is high	AND	e is high	THEN	t = Ed
IF	v is med	AND	e is low	THEN	t = Ha
IF	v is high	AND	e is low	THEN	t = Pi

In this application, ordinary Mamdani min/max-inference is used. Finally, the degree to which "Ho", "Ed", "Ha", or "Pi" belong to the output set can be regarded as the degree to which the particular pixel belongs to area Homogeneous, Edge, Halftone, or Picture, respectively.

For details on the integration of the classification algorithm into the printing process and information about the performance and robustness of the algorithm see [4].

3 The Optimization of the Classification System

The behavior of the fuzzy system depends on six parameters, $v_1, \ldots, v_4$, e_1, and e_2, which determine the shape of the two fuzzy partitions. In the first step, these parameters were tuned manually. Of course, we have also taken into consideration to use (semi)automatic methods for finding the optimal parameters.

Our optimization procedure consists of a painting program which offers tools, such as a pencil, a rubber, a filling algorithm, and many more, which can be used to make a classification of a given representative image by hand. Then an optimization algorithm can be used to find that configuration of parameters which yields the maximal degree of matching between the desired result and the output actually obtained by the classification system.

Assume that we have N sample pixels for which the input values $(\tilde{v}_k, \tilde{e}_k)_{k\in\{1,\dots,N\}}$ are computed and that we already have a reference classification of these pixels $\tilde{t}(k) = (\tilde{t}_{\mathsf{Ho}}(k), \tilde{t}_{\mathsf{Ed}}(k), \tilde{t}_{\mathsf{Ha}}(k), \tilde{t}_{\mathsf{Pi}}(k))$, where $k \in \{1,\dots,N\}$. Since, as soon as the values $\tilde{v}$ and $\tilde{e}$ are computed, the geometry of the image plays no role anymore, we can switch to one-dimensional indices here. Then one possibility to define the performance (fitness) of the fuzzy system would be

$$\frac{1}{N}\sum_{k=1}^{N} d(t(k), \tilde{t}(k)), \tag{2}$$

where $t(k) = (t_{\mathsf{Ho}}(k), t_{\mathsf{Ed}}(k), t_{\mathsf{Ha}}(k), t_{\mathsf{Pi}}(k))$ are the classifications actually obtained by the fuzzy system for the input pairs $(\tilde{v}_k, \tilde{e}_k)$ with respect to the parameters v_1, v_2, v_3, v_4, e_1, and e_2; $d(.,.)$ is an arbitrary (pseudo-)metric on $[0,1]^4$. The problem of this brute force approach is that the output of the fuzzy system has to be evaluated for each pair (v_k, e_k), even if many of these values are similar or even equal. In order to keep the amount of computation low, we "simplified" the procedure by a "clustering process" as follows:

We choose a partition $(P_1,\dots,P_K)$ of the input space, where $(n_1,\dots,n_K)$ are the numbers of sample points $\{p^i_1,\dots,p^i_{n_i}\}$ each part contains. Then the desired classification of a certain part (cluster) can be defined as

$$\tilde{t}_X(P_i) = \frac{1}{n_i}\sum_{j=1}^{n_i} \tilde{t}_X(p^i_j), \tag{3}$$

where $X \in \{\mathsf{Ho}, \mathsf{Ed}, \mathsf{Ha}, \mathsf{Pi}\}$.

If ϕ is a function which maps each cluster to a representative value (e.g., its center of gravity), we can define the fitness (objective) function as

$$\frac{100}{N}\sum_{i=1}^{K} n_i \cdot \left(1 - \frac{1}{2}\cdot \sum_{X\in\{\mathsf{Ho},\mathsf{Ed},\mathsf{Ha},\mathsf{Pi}\}} \left(\tilde{t}_X(P_i) - t_X(\phi(P_i))\right)^2\right), \tag{4}$$

If the number of parts is chosen moderately (e.g. a rectangular 64×32 net which

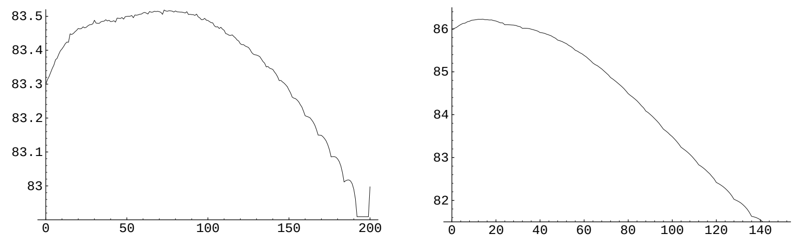

Fig. 6. Cross sections of a function of type (4)

yields $K = 2048$) the evaluation of the fitness function takes considerably less time than a direct application of formula (2).

Note that in (4) the fitness is already transformed such that it can be regarded as a degree of matching between the desired and the actually obtained classification measured in percent. This value has then to be maximized.

In fact, fitness functions of this type are, in almost all cases, continuous but not differentiable and have a lot of local maxima. Figure 6 shows cross sections of such functions. Therefore, it is more reasonable rather to use probabilistic optimization algorithms than to apply continuous optimization methods, which make excessive use of derivatives. This, first of all, requires a (binary) coding of the parameters. We decided to use a coding which maps the parameters v_1, v_2, v_3, v_4, e_1, and e_2 to a string of six 8-bit integers $s_1, \ldots, s_6$ which range from 0 to 255. The following table shows how the encoding and decoding is done:

$$
\begin{array}{ll}
s_1 = v_1 & v_1 = s_1 \\
s_2 = v_2 - v_1 & v_2 = s_1 + s_2 \\
s_3 = v_3 - v_2 & v_3 = s_1 + s_2 + s_3 \\
s_4 = v_4 - v_3 & v_4 = s_1 + s_2 + s_3 + s_4 \\
s_5 = e_1 & e_1 = s_5 \\
s_6 = e_2 - e_1 & e_2 = s_5 + s_6
\end{array}
$$

We first tried a standard GA (see [31] or [45]) with proportional (standard roulette wheel) selection, one-point crossing over with uniform selection of the crossing point, and bitwise mutation. The size of the population m was constant, the length of the strings was, as shown above, 48.

In order to compare the performance of the GAs with other well-known probabilistic optimization methods, we additionally considered the following methods:

Hill climbing: always moves to the best-fitted neighbor of the current string until a local maximum is reached; the initial string is generated randomly.

Simulated annealing: powerful, often used probabilistic method which is based on the imitation of the solidification of a crystal under slowly decreasing temperature (see [81] for a detailed description)

Each one of these methods requires only a few binary operations in each step. Most

	f_{max}	f_{min}	$\bar{f}$	σ_f	#
Hill Climbing	94.3659	89.6629	93.5536	1.106	862
Simulated Annealing	94.3648	89.6625	93.5639	1.390	1510
Improved Simulated Annealing	94.3773	93.7056	94.2697	0.229	21968
GA	94.3760	93.5927	94.2485	0.218	9910
Hybrid GA (elite)	94.3760	93.6299	94.2775	0.207	7460
Hybrid GA (random)	94.3776	94.3362	94.3693	0.009	18631

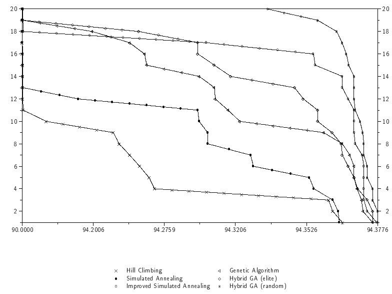

Fig. 7. A comparison of the results of various probabilistic optimization methods

of the time is consumed by the evaluation of the fitness function. So, it is near at hand to take the number of evaluations as a measure for the speed of the algorithms.

Results All these algorithms are probabilistic methods, therefore their results are not well-determined, they can differ randomly within certain boundaries. In order to get more information about their average behavior, we tried out each one of them 20 times for one certain problem. For the given problem we found out that the maximal degree of matching between the reference classification and the classification actually obtained by the fuzzy system was 94.3776%. In the table in Figure 7, f_{max} is the fitness of the best and f_{min} is the fitness of the worst solution; $\bar{f}$ denotes the average fitness of the 20 solutions, σ_f denotes the standard deviation of the fitness values of the 20 solutions, and # stands for the average number of evaluations of the fitness function which was necessary to reach the solution.

The hill climbing method with a random selection of the initial string converged rather quickly. Unfortunately, it was always trapped in a local maximum, but never reached the global solution (at least in these 20 trials).

The simulated annealing algorithm showed similar behavior at the very beginning. After tuning the parameters involved, the performance improved remarkably.

The raw genetic algorithm was implemented with a population size of 20; the crossing over probability was set to 0.15, the mutation probability was 0.005 for each byte. It behaved pretty well from the beginning, but it seemed inferior to the improved simulated annealing.

Next, we tried a hybrid GA, where we kept the genetic operations and parameters of the raw GA, but every 50th generation the best-fitted individual was taken as initial string for a hill climbing method. Although the performance increased slightly, the hybrid method still seemed to be worse than the improved simulated annealing algorithm. The reason that the effects of this modification were not so dramatic might be that the probability is rather high that the best individual is already a local maximum. So we modified the procedure again. This time a *randomly chosen individual* of every 25th generation was used as initial solution of the hill climbing method. The result exceeded the expectations by far. The algorithm was, in all cases, nearer to the global solution than the improved simulated annealing (compare with table in Figure 7), but, surprisingly, sufficed with less invocations of the fitness function. The graph in Figure 7 shows the results graphically. Each line in this graph corresponds to one algorithm. The curve shows, for a given fitness value x, how many of the 20 different solutions had a fitness higher or equal to x. It can be seen easily from this graph that the hybrid GA with random selection led to the best results. Note that the x-axis is not a linear scale in this figure. It was transformed in order to make small differences visible.

4 Concluding Remarks

In this lecture we have investigated the suitability of genetic algorithms for finding the optimal parameters of a fuzzy system, especially if the analytical properties of the objective function are bad. Moreover, hybridization has been discovered as an enormous potential for improvements of genetic algorithms.

Learning with Genetic Fuzzy Systems: Pittsburgh Approach

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

In this lecture we shortly describe the use of Genetic Fuzzy Systems with the Pittsburgh learning approach for learning Rule Bases and Knowledge Bases for Fuzzy Rule Bases Systems.

Key words: Genetic Algorithms, Fuzzy Rule Based Systems, Learning, Pittsburgh Approach.

1 Introduction

Recently, there has been a growing interest in using Genetic Algorithms (GAs) for machine learning problems, appearing different genetic learning approaches. One of them, the Pittsburgh approach adopts the view that each individual in a population, each chromosome, encodes a whole rule sets. 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. This model was initially proposed by Smith in 1980 [74].

In this lecture we shortly describe the use of Genetic Fuzzy Systems (GFSs) with this learning approach for learning Rule Bases (RB) and Knowledge Bases (KB) for Fuzzy Rule Bases Systems (FRBSs).

2 Genetic Learning of RB

It is possible to represent the RB of an FRBS with three different representations. These representations are: relational matrix, decision table and list or set of rules. 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.

2.1 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 ([62,78]).

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 $m \times n$ 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 [68] or real numbers.

2.2 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 of fixed 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 [58].

Rules of variable length. Codes with position independent meaning and based on pairs {*variable, membership function*} (the membership functions is described using a label) are used in [43].

3 Genetic Learning of KB

The simultaneous use as genetic material of the DB and the RB of an FRBS has produced different and interesting results. The most general approach is the use of a set of parameterized membership functions and a list of fuzzy rules that are jointly coded to generate a chromosome, then applying a Pittsburgh-type GA to evolve a population of such chromosomes. This kind of GFSs use chromosomes containing two subchromosomes that encode separately, but not independently, the DB and the RB.

It is possible to maintain, at this point, the same division that was stated when talking about genetic learning of RBs with a Pittsburgh approach: learning complete rule bases or partial rule bases.

3.1 Using a Complete RB

In [64] the rule base is represented as a fuzzy relation matrix (R), and the GA modifies R or the fuzzy membership functions (triangular) or both of them simultaneously, on a Fuzzy Logic Controller (FLC) with one input and one output variables. Each gene is a real number. When generating the optimal fuzzy relation matrix this real number corresponds to a fuzzy relation degree whose value is between 0 and 1. The genetic string is obtained by concatenating the $m \times n$ real numbers that constitute R. When finding simultaneously the optimal rule base and the fuzzy membership functions, each chromosome allocates two subchromosomes: the genes of the rule base and the genes of the fuzzy membership functions. Both subchromosomes are treated as independent entities as far as crossover and mutation are concerned but as a single entity as far as reproduction is concerned.

A slightly different approach is to use a TSK-type rule base, structuring its genetic code as if it came from a decision table. In this case, the contents of the code of a rule base is an ordered and complete list containing the consequents of all possible rules, where the antecedents are implicitly defined as a function of the position the consequent occupies in the list.

The fuzzy membership functions constitute a first subchromosome while the coefficients of the consequents for a TSK fuzzy model constitute the second subchromosome. One gene is used to code each coefficient of a TSK-type in [53], in [26] a single coefficient is considered for the output.

3.2 Using a Partial RB

Liska and Melsheimer ([54]) use a rule base defined as a set of a fixed number of rules, and code each rule with integer numbers that define the membership function related with a certain input or output variable that is applied by the rule (membership functions for every variable are ordered). The systems use radial membership functions coded through two real numbers (two genes). The genetic string is obtained by concatenating the two genes in each membership function.

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 rule codes. To represent a single rule, it is possible to use a position dependent code with as many elements as the number of variables of the system. A possible content in these elements is: a label pointing to a certain fuzzy set in the fuzzy partition of the variable ([71]) or a binary string with a bit per fuzzy set in the fuzzy partition of the variable ([55]).

Using an approximate approach, [13,14] include the definition of the membership functions into the rules, coding each rule through the corresponding set of membership functions.

4 A Learning Process of Fuzzy Logic Controllers

FLCs represent a particular and widely applied kind of FRBSs. A genetic process using a Pittsburgh approach and working on an FLC may be rewritten as follows in such a situation:

(1) Start with an initial population of solutions that constitutes the first generation (*P(0)*).
(2) Evaluate *P(0)*:
   (a) take each chromosome (KB) from the population and introduce it into the 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+1)*) by applying the evolution operators to the individuals in *P(t)*,
(b) evaluate *P(t+1)* and
(c) $t = t + 1$.

(4) Stop.

5 An Example of GFS with Pittsburgh Approach

This section describe, in a few lines, one of the GFSs previously cited, specifically a GFS learning RBs and representing the rule base with a decision table. This method was proposed by Philip Thrift ([78]). This example will be analyzed according to the keys of the learning process, the population of potential solutions, the set of evolution operators and the performance index.

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 $\{0, 1, \ldots, 5\}$, 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 5×5 places and consequently will generate a chromosome with 25 genes.

The population where the genetic process will be applied is a number of chromosomes (31 in the example described in the paper) coded as strings with 25 integers in $\{0, 1, \ldots, 5\}$.

The set of evolution operators. The system uses a standard two point crossover ([60]) 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 ($\max(|x|, |v|) < 0.5$). 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).

6 Concluding Remarks

We have reviewed the GFS for learning FRBSs based on the Pittsburgh approach,. showing the different proposal developped under this approach.

Learning with Genetic Fuzzy Systems: Iterative Rule Learning Approach

Francisco Herrera

Dept. of Computer Science and A.I., E.T.S. Ingeniería Informática
University of Granada, E-18071 Granada, Spain
E-mail: herrera@decsai.ugr.es

Abstract

In this lecture we shortly describe the use of Genetic Fuzzy Systems with the Iterative Rule Learning approach for learning Rule Bases and Knowledge Bases for Fuzzy Rule Bases Systems.

Key words: Genetic Algorithms, Fuzzy Rule Based Systems, Learning, Iterative Rule Learning Approach.

1 Introduction

Since the beginning of the 80s there has been growing interest in applying methods based on Genetic Algorithms (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.

In recent literature we may find different algorithms that use a new learning model based on GAs, the *Iterative Rule Learning (IRL) approach* [82,32]. 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. This model has been used in papers such as [82,36,35,33,39,41,15–17].

This lecture describes the IRL approach for learning fuzzy rule based sytems (FRBSs).

2 IRL Approach

In this approach the GA provides a partial solution to the problem of learning, 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.

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* [35], *SIA* [82] and the *genetic generation process* proposed in [39]. These three genetic learning processes use the IRL 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 IRL 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.

2.1 Multi-Stage Genetic Fuzzy System Based on the IRL Approach

Learning algorithms that use the IRL 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 IRL 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.

Genetic generation stage. In this stage the IRL 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 IRL 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* [36,35] and for learning KB as the *genetic generation process* proposed in [39,41].

Postprocessing stage: selection and refinement. As we mentioned earlier, the IRL 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 have been used , a refinement algorithm [34] and a selection or simplification algorithm [40].

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. The main difference between both processes is the coding scheme.

2.2 A Multi-stage Genetic Fuzzy Rule-Based System Structure

In the following we present a guideline structure for multi-stage GFRBSs used in [15–17,41]:

a) A fuzzy rule generation process. This process will determine the type of the final FRBS generated, so the generated fuzzy rules may present a descriptive, constrained approximative or unconstrained approximative semantics. In all cases, it will present two components: a *fuzzy rule generating method* composed of an inductive or evolutionary process which uses a niche criterion for obtaining the best possible cooperation among the fuzzy rules generated when working with the approximative approach, and an *iterative covering method* of the system behaviour example set, which penalizes each rule generated by the fuzzy rule generating method by considering its covering over the examples in the training set and removes the ones yet covered from it. This process allows us to obtain a set of fuzzy rules with a concrete semantics covering the training set in an adequate form.

b) A genetic multisimplification process for selecting rules, based on a binary coded GA with a phenotypic sharing function and a measure of the FRBS accuracy in the problem being solved. It will save the overlearning that the previous component may cause due to the existence of redundant rules, with the aim of obtaining a simplified KB presenting the best possible cooperation among the fuzzy rules composing it. This process will obtain different possibilities for this simplified KB thanks to a genotypic niching scheme.

c) An evolutionary tuning process based on any kind of real coded EA and a measure of the FRBS performance. It will give the final KB as output by adjusting the membership functions for each fuzzy rule in each possible KB obtained from the genetic multisimplification process. The type of tuning performed will depend on the nature of the FRBS being generated, i.e., when generating a descriptive FRBS, a global tuning of the fuzzy partition associated to each linguistic variable will be performed, but when working with any of the approximative approaches, the membership functions involved in each fuzzy rule will be adjusted. The most accurate KB obtained in this stage will constitute the final output of the whole learning process.

Properties required for the generated Knowledge Base. Several important statical properties have to be verified by the KB in order to obtain an accurate FRBS. The multi-stage GFRBSs obtained from our methodology will consider two of them, the *completeness* and *consistency*, by including some criteria in the different stage fitness functions. These criteria will penalize those solutions not verifying adequatelly both properties. For a wider description, refers to [16,17].

3 Concluding Remarks

In this paper, we have presented the IRL 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*.

Learning with Genetic Algorithms: Michigan Approach

Ulrich Bodenhofer

Fuzzy Logic Laboratorium Linz-Hagenberg, Dept. of Mathematics
University of Linz, A-4040 Linz, Austria
E-mail: ulrich@flll.uni-linz.ac.at

Abstract

This lecture gives an introduction to a class of classifier systems which employ genetic learning in an online process. Such systems are often called classifier systems of the Michigan type. A simple variant will be discussed in detail — the so-called Holland classifier system.

Key words: Holland Classifier System, Michigan Approach, Online Learning.

---

1 Introduction

Classifier systems of the Pittsburgh type have in common that (1) the genetic algorithms operate on whole rulebases, they work with populations of rulebases, and (2) rulebases are judged globally, i.e., the performance of whole rulebases is evaluated by the fitness function.

The other approach — the so-called Michigan approach — is to observe the behavior of the system throughout a certain period of time adjusting the rules according to temporal payoff from the environment. While in the Pittsburgh approach whole rulebases are considered in an offline process, the Michigan approach operates on single rules in an online process or a simulated environment. Obviously, in the Michigan approach, techniques for judging the performance of single rules are necessary. In fact, this is, in most cases, a nontrivial task, since positive effects of some rules are not always observable immediately. Consider, for instance, the game of chess, where early moves can contribute to a late success.

Figure 8 shows the typical architecture of Michigan type system. The main compo-

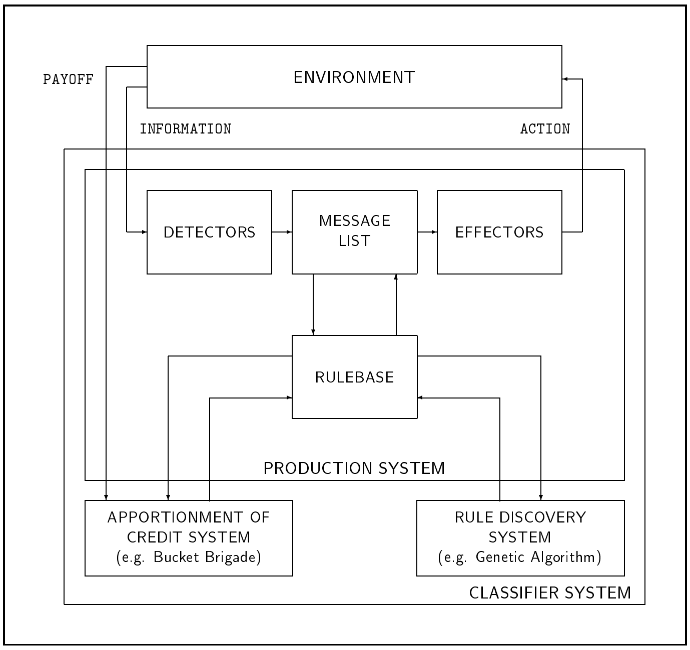

Fig. 8. A classifier system of the Michigan type

nents are:

(1) A production system containing a rulebase which processes incoming messages from the environment and sends output messages to the environment
(2) An apportionment of credit system which receives payoff from the environment and determines which rules had been responsible for that feedback.
(3) A genetic algorithm which recombines existing rules and introduces new ones.

Obviously, the learning task is divided into two subtasks — the judgment of already existing and the discovery of new rules.

2 The Holland Classifier System

A Holland classifier system is a classifier system of the Michigan type which processes binary messages of a fixed length through a rulebase whose rules are adapted according to the response of the environment ([44–46,28]).

2.1 The Production System

First of all, the communication of the production system with the environment is done via an arbitrarily long list of messages. The detectors translate responses from the environment into binary messages and place them on the message list which is then scanned and changed by the rulebase. Finally, the effectors translate output messages into actions on the environment, such as forces or movements.

Messages are binary strings of the same length k. More formally, a message belongs to $\{0,1\}^k$. The rulebase consists of a fixed number m of rules (classifiers) which consist of a fixed number r of conditions and an action, where both conditions and actions are strings of length k over the alphabet $\{0,1,*\}$. The asterisk plays the role of a wildcard, a "don't care" symbol.

A condition is matched, if there is a message in the list which matches the condition in all non-wildcard positions. Moreover, conditions, except the first one, may be negated by adding a "–" prefix. Such a prefixed condition is satisfied if there is *no* message in the list which matches the string associated with the condition. Finally, a rule fires if all the conditions are satisfied, i.e., the conditions are connected with AND. Such "firing" rules compete to put their action messages on the message list (see 2.2).

In the action parts, the wildcard symbols have a different meaning. They take the role of "pass through" element. The output message of a firing rule, whose action part contains a wildcard, is composed from the non-wildcard positions of the action and the message which satisfies the first condition of the classifier (this is actually the reason why negations of the first conditions are not allowed). More formally, the outgoing message $\tilde{m}$ is defined as

$$\tilde{m}[i] := \begin{cases} a[i] & \text{if } a[i] \neq * \\ m[i] & \text{if } a[i] = * \end{cases} \qquad i = 1, \dots, k, \tag{5}$$

where a is the action part of the classifier and m is the message which matches the first condition. Formally, a classifier is a string of the form

$$\text{Cond}_1, [\text{“–”}]\text{Cond}_2, \dots, [\text{“–”}]\text{Cond}_r/\text{Action}, \tag{6}$$

where the brackets should express the optionality of the "–" prefixes.

Moreover, it can be of advantage to supply the messages with prefixes, so-called tags, which identify the origin of the message. Consequently, these prefixes must also be appended to the conditions and actions of the classifiers. In this case, we must take special care that no action specifies the prefix reserved for the input interface. Tagging offers new opportunities to transfer information about the current step into the next step. This can be accomplished by placing tagged messages on the list which are not interpreted by the output interface. These messages, which, obviously, contain information about the previous step, can support the decisions in the next step. So, appropriate use of tags permits rules to be coupled to act sequentially. In some sense, such messages are the memory of the system.

To summarize this, a single execution cycle of the production system consists of the following steps:

(1) Messages from the environment are appended to the message list.
(2) All the conditions of all classifiers are checked against the message list to obtain the set of firing rules.
(3) The message list is erased.
(4) The firing classifiers participate in a competition to place their messages on the list (see below).
(5) The winning classifiers place their actions on the list.
(6) The messages directed to the effectors are executed.

This procedure is repeated iteratively.

How (6) is done, if these messages are deleted or not, and so on, depends on the concrete implementation. It is, on the one hand, possible to choose a representation such that every output message can be interpreted by the effectors. On the other hand, it is possible to direct messages explicitely to the effectors with a special tag. In this case, if no messages are directed to the effectors, the system is in a thinking phase.

If a classifier R_1 produces a message m', which is not directed to the effectors, but tagged as an internal message, and m' satisfies a condition of a classifier R_2 in the next timestep, R_2 is called a consumer of R_1. Reversly, R_1 is called a supplier of R_2.

2.2 Credit Assignment — The Bucket Brigade Algorithm

The purpose of the credit assignment system is to assign a strength value to each classifier. This strength value represents the correctness and importance of a classifier. On the one hand, the strength value influences the chance of a classifier to place its action on the output list. On the other hand, the strength values are used by the rule discovery system. Let us denote the strength value of classifier R_i in timestep T with $u_{i,t}$.

The competition for having the right to post an action and the adaptation of the strength values depending on the feedback (payoff) from the environment is called *Bucket Brigade Algorithm*. It can be regarded as a simulated economic system in which various agents, in our case classifiers, participate in an auction, where the chance to buy the right to post the action depends on the strength of the agents.

In one of its simpliest forms, the bid of a classifier is defined as

$$b_{i,t} := c_L \cdot u_{i,t} \cdot s_i, \tag{7}$$

where $c_L \in [0,1]$ is a learning parameter, similar to learning rates in artificial neural nets, and s_i is the specifity, the number of non-wildcard symbols in the condition part of the classifier. If c_L is chosen small, the system adapts slowly. If it is chosen too high, the strengths can tend to oscillate chaotically.

Then, depending on the bids, the rules, which are allowed to place their output messages on the list, the so-called winning agents, are selected. In the simpliest case, this can be done by a random experiment. For each bidding classifier it is decided randomly, if it wins or not, where the probability that it wins is proportional to its bid:

$$\mathsf{P}[r_i \text{ wins}] := \frac{b_{i,t}}{\sum\limits_{j \in \mathrm{Sat}_t} b_{j,t}}, \tag{8}$$

where Sat_t is the set of indices which belong to satisfied classifiers at time t.

Obviously, in this approach, more than one winning classifiers are allowed. Of course, other selection schemes are reasonable, for instance the highest bidding agent wins alone. This can be necessary to avoid that two winning classifiers direct mutually excluding actions to the effectors.

Now let us discuss how payment from the environment is distributed and how the strengths are adapted. For this purpose, let us denote the set of classifiers, which have supplied a winning agent R_i in step t, with $S_{i,t}$. Then the new strength of a winning agent is reduced by its bid and increased by its portion of the payoff P_t received from the environment:

$$u_{i,t+1} := u_{i,t} + \frac{P_t}{w_t} - b_{i,t}, \tag{9}$$

where w_t is the number of winning agents in the actual time step. A winning agent pays its bid to its suppliers, which share the bid among each other, equally in the simpliest case:

$$u_{l,t+1} := u_{l,t} + \frac{b_{i,t}}{|S_{i,t}|} \qquad \forall r_l \in S_{i,t} \tag{10}$$

If a winning agent has also been active in the previous step and supplies another winning agent, the value above is additionally increased by one portion of the bid the consumer offers. In the extreme case, that two winning agents have supplied each other mutually, the portions of the bids are exchanged in the manner as presented above. The strengths of all other classifiers r_n, which are neither winning agents nor suppliers of winning agents, are reduced by a certain factor (they pay a tax):

$$u_{n,t+1} := u_{n,t} \cdot (1 - T), \tag{11}$$

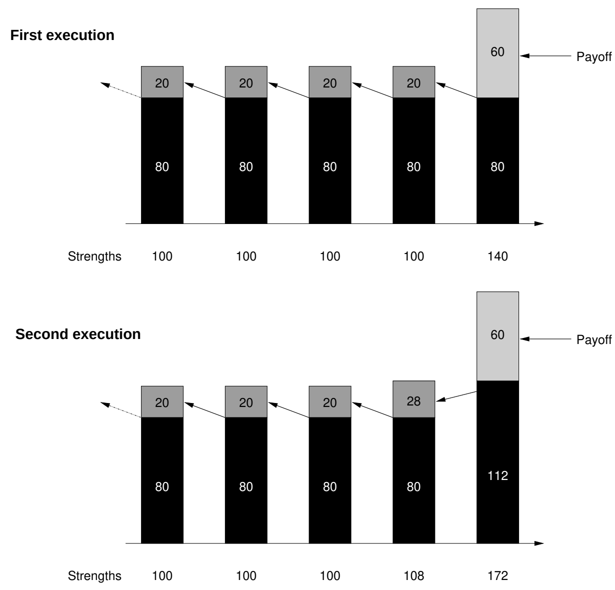

Fig. 9. The bucket brigade principle

where $T \in [0, 1]$ is a small value. The intention of taxation is to punish classifiers which never contribute anything to the output of the system. With this concept redundant classifiers, which never become active, can be filtered out.

The idea behind credit assignment in general and bucket brigade in particular is to increase the strengths of rules which have set the stage for later successful actions. The problem of determining such classifiers, which were responsible for conditions under which it was later on possible to receive a high payoff, can be very difficult. However, the bucket brigade algorithm can solve this problem, although, obviously, strength is only transferred to the suppliers which were active in the previous step. Each time the same sequence is activated, a little bit of the payoff is transferred one step back in the sequence. It is easy to see, that repeated successful execution of a sequence can increase the strengths of all coupled classifiers involved.

Figure 9 shows a simple example how the bucket brigade algorithm works. For simplicity, we consider a sequence of five classifiers which always bid 20 percents of their strength. Only after the fifth step, after the activation of the fifth classifier, a payoff of 60 is received. The further future of this sequence would be the one shown in figure 10. It is easy to see from this example that the reinforcement of the strengths is slow at the beginning but it accelerates later. Exactly this property contributes much to the robustness of classifier systems — they tend to be cautious at the beginning, trying not to rush conclusions, but, after a certain number of similar situations, the system adopts the rules more and more. Figure 10 also shows a graphical visualization of this fact interpreting the table as a two-dimensional surface.

Strength after the					
3rd	100.00	100.00	101.60	120.80	172.00
4th	100.00	100.32	105.44	136.16	197.60
5th	100.06	101.34	111.58	152.54	234.46
6th	100.32	103.39	119.78	168.93	247.57
:					
10th	106.56	124.17	164.44	224.84	278.52
:					
25th	215.86	253.20	280.36	294.52	299.24
:					
execution of the sequence					

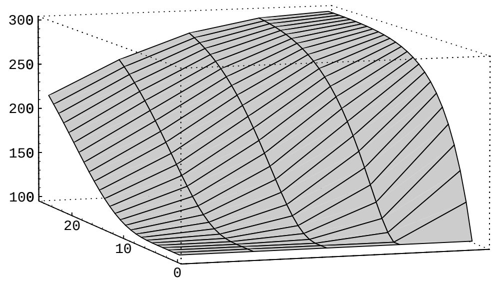

Fig. 10. Repeated bucket brigade

2.3 Rule Generation

While the apportionment of credit system just judges the rules, the purpose of the rule discovery system is to eliminate low-fitted rules and to replace them by hopefully better performing ones. The fitness of a rule is given by its strength. Since the classifiers of a Holland classifier system themselves are strings, the adaptation of a genetic algorithm to the problem of rule induction is straightforward, though many variants are reasonable. Almost all variants have in common that the GA is not invoked in each time step, but only every n-th step, where n has to be set such that enough information about the performance of new classifiers can be obtained in the meantime.

Furthermore, this process of acquiring new rules has an interesting side effect. It is more than only the exchange of parts of conditions and actions. Since we have not stated restrictions for manipulating tags, the genetic algorithm can recombine parts of established tags to invent new tags. In the following, tags spawn related tags establishing new couplings. These new tags survive if they contribute to useful interactions. In this sense, the GA additionally creates experience-based internal structures.

3 Concluding Remarks

In this part, the Michigan approach has been introduced. As one of the most important representatives, the Holland classifier system has been studied in detail. We have seen that the main idea is that single rules are manipulated in an online process, which requires a profound analysis of the performance of every rule. It is important to point out that there is no explicit distinction between learning and regular work of the system. Hence, such a system can adapt to varying environmental circumstances automatically. However, one should not forget that the random modification of rules, which is done by a genetic algorithm, can be a risk in some applications where security is of special importance. Moreover, the Michigan approach is subject to fail if the environment is so complex that there is only a low probability that important state sequences are observed repeatedly.

Learning with Genetic Fuzzy Systems: Michigan Approach

Ulrich Bodenhofer

Fuzzy Logic Laboratorium Linz-Hagenberg, Dept. of Mathematics
University of Linz, A-4040 Linz, Austria
E-mail: ulrich@flll.uni-linz.ac.at

Abstract

This lecture deals with fuzzy classifier systems of the Michigan type, where the parameters of a fuzzy system are adapted by genetic algorithms in an online process.

Key words: ELF, Fuzzy Online Learning, Michigan Approach.

1 Introduction

While classifier systems of the Michigan type had been introduced by J. H. Holland in 1976, their fuzzification awaited discovery many years. The first fuzzy classifier system of the Michigan type was introduced by M. Valenzuela-Rendón ([79,80]) and is, more or less, a straightforward fuzzification of a Holland classifier system. An alternative approach has been developed by A. Bonarini ([9,10]), who applies a different scheme of competetion between classifiers. These two approaches have in common that they operate only on the rules — the shape of the membership functions is fixed. A third method, which was introduced by P. Bonelli and A. Parodi ([65]), tries to optimize even the membership functions and the output weights in accordance to payoff from the environment.

2 Fuzzifying Holland Classifier Systems

2.1 The Production System

We consider a fuzzy controller with real-valued input and output. The system has, unlike ordinary fuzzy controllers, three different types of variables — input, output,

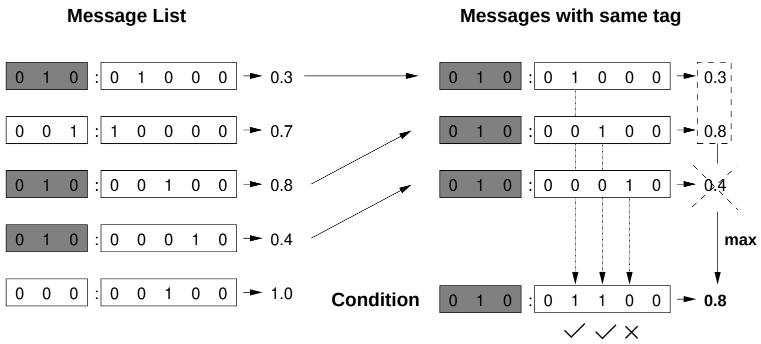

Fig. 11. Matching a fuzzy condition

and internal variables. As we will see later, internal variables are for the purpose of storing information about the near past. They correspond to the internally tagged messages in Holland classifier systems. For the sake of generality and simplicity, all the universes of discourse, are transformed to the unit interval $[0, 1]$. For each variable the same number of membership functions n is assumed. These membership functions are fixed at the beginning. They are not changed throughout the learning process. M. Valenzuela-Rendón took bell-shaped function which divided the interval rather equally.

A message is a binary string of length $l + n$, where n is the number of membership functions defined above and l is the length of the prefix (tag), which identifies the variable to which the message belongs. A good choice for l would be $\lceil \log_2 K \rceil$, where K is the total number of variables we want to consider. To each message an *activity level*, which represents a truth value, is assigned. Consider for instance the following message ($l = 3$, $n = 5$):

$$\underbrace{010}_{=2} : 00010 \to 0.6$$

Its meaning is “Input no. 2 belongs to fuzzy set no. 4 with a degree of 0.6”. On the message list only so-called minimal messages are used, i.e., messages with only one 1 in the part which identifies the numbers of the fuzzy sets.

Classifiers again consist of a fixed number r of conditions and an action part. Note that, in this approach, no wildcards and no “–” prefixes are used. Both condition and action part are also binary strings of length $l + n$, where the tag and the identifiers of the fuzzy sets are separated by a colon. Then the degree to which such a condition is matched is a truth value between 0 and 1. The degree of matching is computed as the maximal activity of messages on the list, which have the same tag and whose 1s are a subset of those of the condition. Figure 11 shows a simple example how this matching is done. The degree of satisfaction of the whole classifier is then computed as the minimum of matching degrees of the conditions. This is then also the activity level which is assigned to the output message (i.e., Mamdani inference).

The whole rulebase consists of a fixed number m of such classifiers. Similarly to Holland classifier systems, one execution step of the production system is done as follows:

(1) The detectors receive crisp input values from the environment and translate them into minimal messages which are then added to the message list.
(2) The degrees of matching are computed for all classifiers.
(3) The message list is erased.
(4) The output messages of some matched classifiers (see below) are placed on the message list.
(5) The output messages are translated into minimal messages. For instance, the message $010 : 00110 \to 0.9$ is split into the two messages $010 : 00010 \to 0.9$ and $010 : 00100 \to 0.9$.
(6) The effectors discard the output messages (referring to output variables) from the list and translate them into instructions to the environment.

From point 2 it can be seen easily that it is of advantage to use fuzzy sets with local support instead of bell-shaped ones, because, if bell-shaped fuzzy sets are used, every rule fires in each time step.

Step 6 is done by a modified Mamdani inference: The sum (instead of the maximum or another t-conorm) of activity levels of messages, which refer to the same fuzzy set of a variable, is computed. The membership functions are then scaled with these sums. Finally, the center of gravity of the "union" (i.e. maximum) of these functions, which belong to one variable, is computed (Sum-Prod inference).

2.2 Credit Assignment

Since fuzzy systems have been designed to model transitions, a probabilistic auction process as discussed in connection with Holland classifier systems, where only a small number of rules is allowed to fire, is not desirable. Of course, we again assign strength values to the classifiers.

If we are dealing with a one-stage system, in which payoff for a certain action is received immediately, where no long-term strategies must be evolved, we can suffice with allowing all matched rules to post their outputs and sharing the payoff among the rules, which were active in the last step, according to their activity levels in this step. For example, if R_t is the set of classifiers, which have been active at time t, and P_t is the payoff received after the t-th step, the modification of the strengths of firing rules can be defined as

$$u_{i,t+1} := u_{i,t} + P_t \cdot \frac{a_{i,t}}{\sum\limits_{r_i \in R_t} a_{i,t}} \qquad \forall r_i \in R_t, \tag{12}$$

where $a_{i,t}$ denotes the activity level of the classifier r_i at time t. It is again possible to reduce the strength of inactive classifiers by a certain tax.

In the case, that the problem is so complex that long-term strategies are indispensable, a fuzzification of the bucket brigade mechanism must be found. While Valenzuela-Rendón only provides a few vague ideas, we state one possible variant, where the firing rules pay a certain value to their suppliers which depends on the activity level. The strength of a classifier, which has recently been active in time step t is then increased by a portion of the payoff as defined in (12), but it is additionally decreased by a value

$$b_{i,t} := c_L \cdot u_{i,t} \cdot a_{i,t}, \tag{13}$$

where $c_L \in [0,1]$ is again the learning rate. Of course, it is again possible to incorporate terms which depend on the specifity of the classifier.

This "bid" is then shared among the suppliers of such a firing classifier according to the amount they have contributed to the matching of the consumer. If we consider an arbitrary but fixed classifier r_j which has been active in step t and if we denote the set of classifiers supplying r_j, which have been active in step $t-1$, with $S_{j,t}$, the change of the strengths of these suppliers can be defined as

$$u_{k,t+1} := u_{k,t} + b_{j,t} \cdot \frac{a_{k,t-1}}{\sum\limits_{r \in S_{j,t}} a_{r,t-1}} \qquad \forall r_k \in S_{j,t}. \tag{14}$$

It is easy to see, that this can be an appropriate generalization of the bucket brigade algorithm as described in the previous lecture.

2.3 Rule Discovery

The adaptation of a genetic algorithm to the problem of manipulating classifiers in our system is again straightforward. We only have to take special care that tags in conditional parts must not refer to output variables and that tags in the action parts of the classifiers must not refer to input variables of the system.

Analogously to our previous considerations, if we admit a certain number of internal variables, the system tends to build up internal chains, coupled sequences, autonomously. If we admit internal variables, a classifier system of this type not only learns stupid input-output actions, it also tries to discover causal interrelations.

3 Bonarini's ELF Method

In [9], A. Bonarini presents his ELF (=evolutionary learning of fuzzy rules) method and applies it to the problem of guiding an autonomous robot. The key issue of ELF is to find a small rulebase which only contains important rules. While he takes over many of M. Valenzuela-Rendón's ideas, his way of modifying the rulebase differs strongly from Valenzuela-Rendón's straightforward fuzzification of Holland's technique.

Bonarini calls the modification scheme "cover-detector algorithm". The number of rules can be varied in each timestep depending on the number of rules which match the actual situation. This is done by two mutually exclusive operations:

(1) If the rules, which match the actual situation, are too many, the worst of them is deleted.
(2) If there are too few rules matching the current inputs, a new rule, whose antecents cover the current state, with randomly chosen consequent value, is added to the rulebase.

The genetic operations are only applied to the consequent values of the rules. Since the antecedents are generated on demand in the different timesteps, no taxation is necessary.

Seemingly, such a simple modification scheme can only be applied to so-called one-stage problems, where the effect of each rule can be observed in the next timestep. For applications where this is not valid, e.g., backing up a truck, Bonarini introduced a modification of his ELF algorithm — the concept of an *episode*, which is a given number of subsequent control actions, after which the reached state is evaluated.

4 Online Modification of the Whole Knowledge Base

While the last two methods only manipulate rules and work with fixed membership functions, there is at least one variant of fuzzy classifier systems where also the membership functions are involved in the learning process. This variant was introduced by A. Parodi and P. Bonelli in [65].

The main idea is that an approximative knowledge base is used instead of a descriptive one as in the two previous examples. So, a fuzzy rule is not represented as a linguistic expression which refers only to labels of fuzzy sets, but a fuzzy relation on $X \times Y$, where X is the input and Y is the output domain. More specifically, each rule is represented as a pair consisting of a fuzzy subset of X and a fuzzy subset of Y.

Since, in many applications, X and Y are themselves cross products, i.e., $X = X_1 \times \cdots \times X_n$ and $Y = Y_1 \times \cdots \times Y_m$, rules in a approximative knowledge base can be written as

$$A_{i1} \times \cdots \times A_{in} \times B_{i1} \times \cdots \times B_{im} \tag{15}$$

where i is the index of the rule.

If one restricts to certain class of fuzzy subsets, such as triangular or bell-shaped membership functions, it is possible to encode a rule as

$$(a_{i1}, \ldots, a_{in}, b_{i1}, \ldots, b_{im})$$

where a_{ij} and b_{ij} are parameters uniquely identifying a fuzzy subset of X_j or Y_j, respectively.

Moreover, in this approach, each rule is additionally equipped with a strength factor, which is taken as a scaling factor of the output set. This strength factor is also used as fitness measure by the genetic algorithm which modifies the knowledge base and modified according to payoff from the environment.

5 Concluding Remarks

The advantage of M. Valenzuela-Rendón's method is generality. Its applicability is not limited to one-stage systems. However, the use of internal variables can lead to difficultly interpretable fuzzy systems.

Bonarini's ELF method is suitable for one-stage systems and multi-stage systems, where the duration of effects of rules can be assumed as limited (length of episode). The advantage of this approach is that also the size of the rulebase is optimized.

Parodi's and Bonelli's approach has the advantage that the membership functions need not to be fixed in the design phase of the system. Therefore, the system can learn to emphasize certain regions of the input and output domains. The disadvantage is that, in general, approximative representations are much more complicated and can result in difficultly interpretable knowledge bases.

Learning with Genetic Fuzzy Systems: Other Approaches

Ulrich Bodenhofer

Fuzzy Logic Laboratorium Linz-Hagenberg, Dept. of Mathematics
University of Linz, A-4040 Linz, Austria
E-mail: ulrich@flll.uni-linz.ac.at

Abstract

In this part, a few other ways, in which genetic algorithms and fuzzy systems can be combined, are discussed.

Key words: Fuzzy Genetic Programming, Hierarchical Structure, Nagoya Approach, TSK Model.

1 Offline Optimization of a Table-Based TSK Controller

In [53,77], M. Lee and H. Takagi introduce a method for finding an optimal TSK controller. In general, a Sugeno controller (see [75]) is a rulebase consisting of rules of the form

$$\mathsf{IF}\ x\ \mathsf{is}\ A_j\ \mathsf{THEN}\ y\ \mathsf{is}\ f_j(x),$$

where j is the index of the rule, A_j is a fuzzy subset of the input domain, y is the crisp (real-valued) output variable, and $f_j(x)$ is an arbitrary mapping from the input to the output domain. If there is at least one firing rule, the output of such a controller can be computed as

$$y = \frac{\sum\limits_{j=1}^{N} \mu_{A_j}(x) \cdot f_j(x)}{\sum\limits_{j=1}^{N} \mu_{A_j}(x)},$$

where N is the total number of rules.

In most recent applications, the functions f_j are constants. A Sugeno controller with polynomials of degree 1, i.e., affine linear mappings, on the right hand side is called Takagi-Sugeno-Kang (TSK) controller.

If the input domain is a cross product of two real intervals $[a_1, b_1]$ and $[a_2, b_2]$ which are decomposed by N_1 and N_2 fuzzy subsets, respectively, the whole controller can be represented as an $N_1 \times N_2$ matrix (table), where each entry contains the three parameters $(\alpha_{ij}, \beta_{ij}, \gamma_{ij})$ of $f_{ij}(x_1, x_2) = \alpha_{ij} \cdot x_1 + \beta_{ij} \cdot x_2 + \gamma_{ij}$.

In order to find an optimal TSK controller with a genetic algorithm, Lee and Takagi first fixed the numbers of fuzzy sets N_1 and N_2 in advance. In their model, triangular-shaped fuzzy sets with three degrees of freedom — center, left offset, and right offset — were chosen. Then, each fuzzy set was encoded by coding each of the three parameters except that not the centers were encoded but their offsets. The reason for coding the offsets of the centers instead of the centers themselves was to avoid that unreasonable overlappings occur and to guarantee a canonical ordering of the fuzzy sets. Finally, a whole controller is represented by a binary string composed of the parameters of the fuzzy subsets of the two input domains and a binary coding of the table containing the consequent values. Apparently, this is a descriptive model.

In the optimization step, a genetic algorithm, which operates on a population of controllers, was applied to find the optimal one in an offline optimization process (cf. Pittsburgh approach) with respect to a certain fitness function.

The generalization of the techniques presented above to the case of controllers with more than two inputs is, apparently, straightforward. However, it is easy to see that the length of the strings depends exponentially on the dimension.

2 The Nagoya Approach

The second idea goes back to T. Furuhashi, K. Nakaoka, and Y. Uchikawa from the university of Nagoya, Japan. In their approach, a modified kind of genetic algorithm is used. Pretentiously, they have called their idea "Nagoya Approach". It can be found in [27].

The three authors applied a genetic algorithm to the optimization of a Mamdani controller with five inputs and two outputs for guiding a robot to a certain goal through a room which contains one moving obstacle.

For the input variables they used bell-shaped fuzzy sets which were additionally scaled with a factor. For the output variables ordinary triangular membership functions were used with the modification that only one offset was used for both left and right.

In this approach, it was assumed that the whole system is controllable with a certain fixed number N (concretely 15 in this application) of rules. Hence, the coding of a whole controller consisted of N binary representations of rules, where each rule was encoded by encoding the parameters of the five input and the two output sets (approximative model).

The most interesting novelty of this paper is that a new modification of an ordinary genetic algorithm is introduced. It provides a better local improvement of single rules. The idea is the following: The whole individual is divided into N_p parts which can be judged independently (the parts and how they can be judged depends on the concrete application). In the mutation step, for each individual of the actual population, a certain number M of clones of each part is produced. $M-1$ of them are mutated. The best clone of each part is then implanted into the new individual. After this step, normal selection and crossing over are performed. Normally, mutation yields bad individuals very often, because one modification can deteriorate the fitness of the whole individual. In this approach, various local phenomena are judged independently which can result in better local improvement.

3 Optimizing Hierarchical Structures of Fuzzy Systems

Consider for instance a fuzzy system with 14 inputs, each represented by three fuzzy sets, and one output with five verbal values. Then the total number of different premises, which specify each variable in the premise, is $3^{14} = 4782969$ and the total number of rules with premises of such a kind is even $5^{4782969}$. Obviously, this is a size which is difficult to survey for a human and impossibly large for an optimization algorithm. This entails the necessity either to use generalizing rules with wildcards in their premises or to prepare a hierarchical structure, which bundles the information such that the decisions are divided into a certain number of subdecisions.

In many applications, where the relationships between the different sets of data are unknown, the preparation of an appropriate hierarchy is a very difficult task. Of course, it is desireable to have methods which can help to find such a hierarchy. One approach for finding an appropriate hierarchical structure by means of genetic algorithms was introduced by T. Fukuda, Y. Hasegawa, and K. Shimojima ([26,73]). These four researchers have presented a coding technique for hierarchies which can be integrated in a genetic optimization process. The applicability of this coding is limited to tree structures, which is not a serious restriction.

Starting point is a binary tree which consists of a certain number of units, i.e., rulebases, which are enumerated from the root to the leaves. In [26,73], 2^{n-2} is recommended, where n is the total number of input variables. Then the numbers of the units, to which the input variables are connected, can be encoded. Of course, a

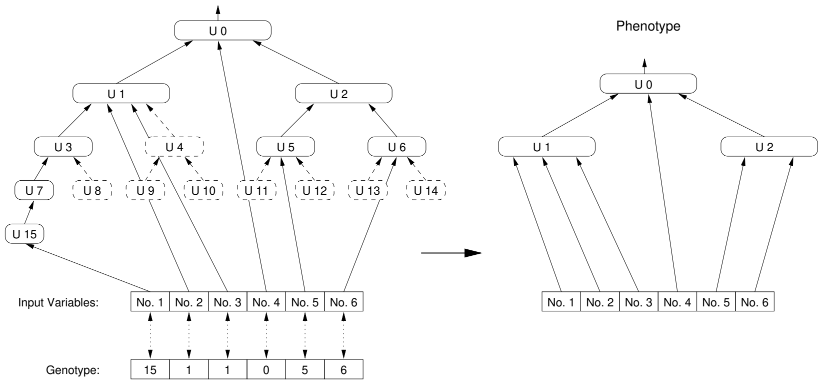

Fig. 12. Example for coding a hierarchical structure

few simplications have to be done:

(1) Only units, which are connected to input variables, and the units above are considered. Units below are removed.
(2) If, after step (1) a bottommost unit has only one input, it is removed.

Figure 12 shows an example for six inputs which illustrates the coding and how the structure is simplified:

- Units 8, 4, and 9–14 are removed, since none of them or of their predecessors is connected to an input variable.
- Units 3, 7, 15, 5, and 6 are removed since they only have one input. Consequently, input No. 1 is considered to be connected to unit 1, No. 5 and No. 6 are connected to unit 2.

Note that, due to the simplifcations, the decoding function is not injective — different strings can result in the same hierarchies.

The concept of coding a hierarchy can now be incorporated in a optimization process. In [73], the hierarchy is tuned before the rulebases. The process of tuning the hierarchical structure can be outlined as follows:

(1) The complete tree is prepared. For each input of each unit, a certain number of fuzzy sets is assumed. Initially, they are set equal for all variables, such that they divide the input space rather equally.
(2) Hierarchies are chosen randomly. The consequent parameters of the rules are initially set to 0.
(3) The consequent parameters (we have a Sugeno controller with constants on the right hand side) are tuned with a gradient method, where the desired output of an intermediate unit is computed using backpropagation.
(4) Selection, crossing over, and mutation are applied to the population.
(5) If the stopping condition is not fulfilled, return to (3).

Apparently, this is only a raw search for an optimal fuzzy system, because the fuzzy sets are fixed, but it can be a way for acquiring a fairly good hierarchical structure. In order to optimize all the parameters involved, it is, after finding an appropriate structure, recommendable to apply further optimization techniques.

4 Fuzzy Genetic Programming

A comparatively new idea is to apply genetic programming to the acquistation of optimal rulebases. Much of the theory goes back to A. Geyer-Schulz ([28]) who also implemented the first application, where he tried to improve stock management strategies with fuzzy genetic programming ([29]).

Within this promising approach, all kinds of constructs for representing fuzzy knowledge, such as adverbs (hedges), different connectives, etc. can be used. The one and only indispensable thing is a rule language whose grammar is given in Backus-Naur form (recursive definition).

An example of such a rule language could be the following:

$$\begin{array}{rcl} \langle\text{rule}\rangle & := & \text{“}\mathsf{IF}\text{”}\ \langle\text{premise}\rangle\ \text{“}\mathsf{THEN}\text{”}\ \langle\text{conclusion}\rangle; \\ \langle\text{premise}\rangle & := & \langle\text{conditional}\rangle\ | \\ & & \text{“(”}\ \langle\text{unary}\rangle\ \langle\text{premise}\rangle\ \text{“)”}\ | \\ & & \text{“(”}\ \langle\text{premise}\rangle\ \langle\text{connective}\rangle\ \langle\text{premise}\rangle\ \text{“)”}\ ; \\ \langle\text{conditional}\rangle & := & \text{“}x_1\text{”}\ \text{“}\mathsf{is}\text{”}\ \langle\text{expr}_1\rangle\ |\ \cdots\ |\ \text{“}x_n\text{”}\ \text{“}\mathsf{is}\text{”}\ \langle\text{expr}_n\rangle\ ; \\ \langle\text{unary}\rangle & := & \text{“}\mathsf{NOT}\text{”}\ ; \\ \langle\text{connective}\rangle & := & \text{“}\mathsf{AND}\text{”}\ |\ \text{“}\mathsf{OR}\text{”}\ ; \\ \langle\text{conclusion}\rangle & := & \text{“}y\text{”}\ \text{“}\mathsf{is}\text{”}\ \langle\text{expr}_y\rangle\ ; \end{array}$$

where the expressions $\langle\text{expr}_1\rangle \ldots \langle\text{expr}_n\rangle$ and $\langle\text{expr}_y\rangle$ are verbal values of the linguistic variables $x_1 \ldots x_n$ and y, respectively. Of course, these values can also be built up recursively of adverbs, adjectives, and connectives.

For coding a whole rulebase, two methods are reasonable:

(1) Fixing the number of rules m:

$$\text{¡rulebase¿} \quad := \quad \text{“(”}\ \underbrace{\langle\text{rule}\rangle\ \text{“,”}\ \ldots\ \text{“,”}\ \langle\text{rule}\rangle}_{m \text{ times}}\ \text{“)”}\ ;$$

(2) Allow an arbitrary number of rules:

$$\begin{array}{rcl} \langle\text{rulebase}\rangle & := & \text{“(”}\ \langle\text{rulelist}\rangle\ \text{“)”}\ ; \\ \langle\text{rulelist}\rangle & := & \langle\text{rule}\rangle\ |\ \text{“,”}\ \langle\text{rulelist}\rangle\ ; \end{array}$$

Furthermore, if the set of verbal values of the linguistic variable y is a finite set of adjectives, it is, under some additional assumptions, possible to suffice with a finite number of rules (see [7, section 5.1.2]).

Genetic operations can be applied as usual in genetic programming. However, it can be of advantage to incorporate mechanisms into the fitness function which additionally take the complexity and the number of the rules into account. Moreover, it is useful to simplify the expressions after each generation in order to avoid wild growth of the premises and conclusions. If simplification is desired, the operations (t-norms and t-conorms) should be chosen such that some derivation laws, such as the De-Morgan law, are fulfilled.

More, especially theoretical details on fuzzy genetic programming can be found in [28], where also a global convergence proof is provided.

5 Concluding Remarks

One of the most important advantages of fuzzy systems is that the functions are parameterized in a way which is interpretable for humans. More specifically, it is possible to translate human knowledge into fuzzy rules and fuzzy sets, but, on the contrary, not every system, which is formally a fuzzy system, is really interpretable. In fact, the probability, that difficultly interpretable configurations are obtained, is rather high when representations with lots of degrees of freedom are tried to be optimized. An alternative, which can help to overcome this problem, is to encode whole fuzzy partitions as shown in the fifth lecture. Obviously, this approach allows less degrees of freedom, which can also speed up convergence.

There have been a lot of publications concerning with genetic optimization of fuzzy systems (see [1,19,20] for recent bibliographies). Each of these approaches — many of them are rather similar — has only been applied to a few benchmark problems. So far, there are no proofs (neither theoretical nor empirical) which methods are suitable for which problems.

Bibliography

[1] J. T. Alander. An indexed bibliography of genetic algorithms and fuzzy logic. Technical Report 94-1-FUZZY, Department of Information Technology and Industrial Management, University of Vaasa, Finland, November 1996. Available at `http://www.uwasa.fi/~jal`.

[2] A. Bárdossy and L. Duckstein. *Fuzzy Rule-Based Modeling with Applications to Geophysical, Biological and Engineering Systems.* CRC Press, Boca Raton, 1995.

[3] P. Bauer, U. Bodenhofer, and E. P. Klement. A fuzzy algorithm for pixel classification based on the discrepancy norm. In *Proc. FUZZ-IEEE'96*, volume III, pages 2007–2012, 1996.

[4] P. Bauer, U. Bodenhofer, and E. P. Klement. A fuzzy method for pixel classification and its application to print inspection. In *Proc. IPMU'96*, volume 3, pages 1301–1305, 1996.

[5] R. J. Bauer. *Genetic Algorithms and Investment Strategies.* John Wiley & Sons, New York, 1994.

[6] J. Biethahn and V. Nissen, editors. *Evolutionary Algorithms in Management Applications.* Springer, Heidelberg, 1995.

[7] U. Bodenhofer. Tuning of fuzzy systems using genetic algorithms. Master's thesis, Johannes Kepler Universität Linz, March 1996.

[8] F. Bolata and A. Nowé. From fuzzy linguistic specifications to fuzzy controllers using evolution strategies. In *Proc. FUZZ-IEEE'95*, volume III, pages 1089–1094, 1995.

[9] A. Bonarini. ELF: Learning incomplete fuzzy rule sets for an autonomous robot. In *Proc. EUFIT'93*, volume I, pages 69–75, 1993.

[10] A. Bonarini. Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modeling: Paradigms and Practice*, pages 265–283. Kluwer Academic Publishers, Dordrecht, 1996.

[11] P. P. Bonissone, P. S. Khedkar, and Y. Chen. Genetic algorithms for automated tuning of fuzzy controllers: A train handling application. In *Proc. FUZZ-IEEE'96*, volume I, pages 675–680, 1996.

[12] L. B. Booker, D. E. Goldberg, and J. H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[13] B. Carse, T. C. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *Fuzzy Sets and Systems*, 80:273–294, 1996.

[14] M. G. Cooper and J. J. Vidal. Genetic design of fuzzy controllers. In *Proc. Int. Conf. on Fuzzy Theory and Technology*, 1993.

[15] O. Cordón and F. Herrera. A hybrid genetic algorithm-evolutionary strategy process for learning fuzzy logic controller knowledge bases. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 251–278. Physica Verlag, Heidelberg, 1996.

[16] O. Cordón and F. Herrera. Hybridizing genetic algorithms with sharing scheme and evolution strategies for designing approximative fuzzy logic controllers. Technical Report DECSAI-96126, Dept. of Computer Science and AI, University of Granada, Spain, December 1996.

[17] O. Cordón and F. Herrera. A three-stage evolutionary process for learning descriptive and approximative fuzzy logic controller knowledge bases from examples. *Internat. J. Approx. Reason.*, 1997.

[18] O. Cordón, F. Herrera, and M. Lozano. A three-stage method for designing genetic fuzzy systems by learning from examples. In H. M. Voight, W. Ebeling, I. Rechenberg, and H. P. Schwefel, editors, *Proc. Int. Conf. on Parallel Problem Solving from Nature*, pages 720–729, Berlin, 1994.

[19] O. Cordón, F. Herrera, and M. Lozano. A classified review on the combination fuzzy logic–genetic algorithms. Technical Report DECSAI-95129, Dept. of Computer Science and AI, University of Granada, Spain, December 1995. Available at `http://decsai.ugr.es/~herrera/fl-ga.html`.

[20] O. Cordón, F. Herrera, and M. Lozano. On the combination of fuzzy logic and evolutionary computation: A short review and bibliography. In W. Pedrycz, editor, *Fuzzy Evolutionary Computation*, pages 33–56. Kluwer Academic Publishers, Dordrecht, 1997.

[21] Y. Davidor, editor. *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization.* World Scientific, Singapore, 1991.

[22] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[23] D. Driankov, H. Hellendoorn, and M. Reinfrank. *An Introduction to Fuzzy Control.* Springer, Heidelberg, 1993.

[24] B. Filipič and D. Juričič. A genetic algorithm to support learning fuzzy control rules from examples. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 403–418. Physica Verlag, Heidelberg, 1996.

[25] D. B. Fogel. *Evolutionary Computation.* IEEE Press, New York, 1995.

[26] T. Fukuda, Y. Hasegawa, and K. Shimojima. Structure organization of hierarchical fuzzy model using by genetic algorithm. In *Proc. FUZZ-IEEE'95*, volume I, pages 295–300, 1995.

[27] T. Furuhashi, K. Nakaoka, and Y. Uchikawa. An efficient finding of fuzzy rules using a new approach to genetic based machine learning. In *Proc. FUZZ-IEEE'95*, volume II, pages 715–722, 1995.

[28] A. Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness*. Physica Verlag, Heidelberg, 1995.

[29] A. Geyer-Schulz. The MIT beer distribution game revisited: Genetic machine learning and managerial behavior in a dynamic decision making experiment. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 658–682. Physica Verlag, Heidelberg, 1996.

[30] A. Giordana and F. Neri. Genetic algorithms in machine learning. *AI Communications*, 9:21–26, 1994.

[31] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[32] A. Gonzalez and F. Herrera. Multi-stage genetic fuzzy systems based on the iterative rule learning approach. *Mathware Soft Comput.*, 4(3), 1997.

[33] A. Gonzalez and R. Perez. A learning system of fuzzy control rules. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 205–225. Physica Verlag, Heidelberg, 1996.

[34] A. Gonzalez and R. Perez. A refinement algorithm of fuzzy rules for classification problems. In *Proc. IPMU'96*, volume II, pages 533–538, 1996.

[35] A. Gonzalez and R. Perez. Completeness and consistency conditions for learning fuzzy rules. *Fuzzy Sets and Systems*, 96(1), 1998.

[36] A. Gonzalez, R. Perez, and J. L. Verdegay. Learning the structure of a fuzzy rule: A genetic approach. *Fuzzy Systems and Artificial Intelligence*, 3:57–70, 1994.

[37] J. J. Grefenstette, editor. *Genetic Algorithms for Machine Learning*. Kluwer Academic Publishers, Boston, 1995.

[38] R. R. Gudwin, F. Gomide, and W. Pedrycz. Nonlinear context adaptation with genetic algorithms. In *Proc. IFSA'97*, 1997.

[39] F. Herrera, M. Lozano, and J. L. Verdegay. Generating rules from examples using genetic algorithms. In B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, editors, *Fuzzy Logic and Soft Computing*. World Scientific, Singapore, 1995.

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

[41] F. Herrera, M. Lozano, and J. L. Verdegay. A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, 1997. (to appear).

[42] F. Herrera and J. L. Verdegay, editors. *Genetic Algorithms and Soft Computing*, volume 8 of *Studies in Fuzziness and Soft Computing*. Physica Verlag, Heidelberg, 1996.

[43] F. Hoffmann and G. Pfister. Learning of a fuzzy control rule base using messy genetic algorithms. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 279–305. Physica Verlag, Heidelberg, 1996.

[44] J. H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to rule-based systems. In R. S. Michalsky, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, volume II*, pages 593–623. Morgan Kaufmann, Los Altos, CA, 1986.

[45] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, MA, first MIT Press edition, 1992. First edition: University of Michigan Press, 1975.

[46] J. H. Holland, K. J. Holyoak, R. E. Nisbett, and P. R. Thagard. *Induction: Processes of Inference, Learning, and Discovery.* Computational Models of Cognition and Perception. The MIT Press, Cambridge, MA, 1986.

[47] J. H. Holland and J. S Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems.* Academic Press, San Diego, CA, 1978.

[48] K. A. De Jong. Learning with genetic algorithms: An overview. *Mach. Learn.*, 3:121–138, 1988.

[49] C. L. Karr. Design of an adaptive fuzzy logic controller using a genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proc. ICGA'91*, pages 450–457, Los Altos, CA, 1991. Morgan Kaufmann.

[50] C. L. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2):26–33, 1991.

[51] C. L. Karr and E. J. Gentry. Fuzzy control of pH using genetic algorithms. *IEEE Trans. Fuzzy Systems*, 1(1):46–53, 1993.

[52] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA, 1992.

[53] M. Lee and H. Takagi. Integrating design stages of fuzzy systems using genetic algorithms. In *Proc. FUZZ-IEE'93*, volume I, pages 612–617, 1993.

[54] J. Liska and S. Melsheimer. Complete design of fuzzy logic systems using genetic algorithms. In *Proc. FUZZ-IEEE'94*, volume II, pages 1377–1382, 1994.

[55] L. Magdalena. *Estudio de la coordinación inteligente en robots bípedos: aplicación de lógica borrosa y algoritmos genéticos.* PhD thesis, Universidad de Politécnica de Madrid, 1994.

[56] L. Magdalena. Adapting gain and sensibility of FLCs with genetic algorithms. In *Proc. IPMU'96*, volume II, pages 739–744, 1996.

[57] L. Magdalena. A first approach to a taxonomy of fuzzy-neural systems. In R. Sun and F. Alexandre, editors, *Connectionist Symbolic Integration*, chapter 5. Lawrence Erlbaum Associates, 1996.

[58] L. Magdalena. Adapting the gain of an FLC with genetic algorithms. *Internat. J. Approx. Reason.*, 17(4):327–350, 1997.

[59] L. Magdalena and F. Monasterio. Evolutionary-based learning applied to fuzzy controllers. In *Proc. FUZZ-IEEE'95*, volume III, pages 1111–1118, 1995.

[60] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Heidelberg, third edition, 1992.

[61] H. Neunzert and B. Wetton. Pattern recognition using measure space metrics. Technical Report 28, Universität Kaiserslautern, Fachbereich Mathematik, November 1987.

[62] K. Ng and Y. Li. Design of sophisticated fuzzy logic controllers using genetic algorithms. In *Proc. FUZZ-IEEE'94*, pages 1708–1712, 1994.

[63] S. K. Pal and P. P. Wang, editors. *Genetic Algorithms for Pattern Recognition.* CRC Press, Boca Raton, FL, 1996.

[64] D. Park, A. Kandel, and G. Langholz. Genetic-based new fuzzy reasoning models with application to fuzzy control. *IEEE Trans. Syst. Man Cybern.*, 24(1):39–47, 1994.

[65] A. Parodi and P. Bonelli. A new approach to fuzzy classifier systems. In S. Forrest, editor, *Proc. ICGA'97*, pages 223–230, Los Altos, CA, 1993. Morgan Kaufmann.

[66] W. Pedrycz, editor. *Fuzzy Control and Fuzzy Systems.* John Wiley & Sons, New York, 1989.

[67] W. Pedrycz, editor. *Fuzzy Evolutionary Computation.* Kluwer Academic Publishers, Dordrecht, 1997.

[68] D. T. Pham and D. Karaboga. Optimum design of fuzzy logic controllers using genetic algorithms. *J. Systems Engineering*, 1:114–118, 1991.

[69] E. H. Ruspini. A new approach to clustering. *Inf. Control*, 15:22–32, 1969.

[70] E. Sanchez, T. Shibata, and L. A. Zadeh, editors. *Genetic Algorithms and Fuzzy Logic Systems.* Soft Computing Perspectives. World Scientific, New York, 1997.

[71] A. Satyadas and K. Krishnakumar. EFM-based controllers for space attitude control: Applications and analysis. In F. Herrera and J. L. Verdegay, editors, *Genetic Algorithms and Soft Computing*, pages 152–171. Physica Verlag, Heidelberg, 1996.

[72] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technologie Series. John Wiley & Sons, New York, 1995.

[73] K. Shimojima, T. Fukuda, and Y. Hasegawa. Self-tuning fuzzy modeling with adaptive membership function, rules, and hierarchical structure based on genetic algorithm. *Fuzzy Sets and Systems*, 71(3):295–309, 1995.

[74] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[75] M. Sugeno. An introductory survey of fuzzy control. *Inform. Sci.*, 36:59–93, 1985.

[76] H. Surmann, A. Kanstein, and K. Goser. Self-organizing and genetic algorithms for an automatic design of fuzzy control and decision systems. In *Proc. EUFIT'93*, volume II, pages 1097–1104, 1993.

[77] H. Takagi and M. Lee. Neural networks and genetic algorithms to auto design of fuzzy systems. In E. P. Klement and W. Slany, editors, *Fuzzy Logic in Artificial Intelligence*, volume 695 of *Lecture Notes in Artificial Intelligence*, pages 68–79. Springer, Berlin, 1993.

[78] P. Thrift. Fuzzy logic synthesis with genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proc. ICGA'91*, pages 509–513, Los Altos, CA, 1991. Morgan Kaufmann.

[79] M. Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In R. K. Belew and L. B. Booker, editors, *Proc. ICGA'91*, pages 346–353, San Mateo, CA, 1991. Morgan Kaufmann.

[80] M. Valenzuela-Rendón. The fuzzy classifier system: Motivations and first results. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 330–334. Springer, Berlin, 1991.

[81] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, Dordrecht, 1987.

[82] G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attribute-based concepts. In *Proc. European Conf. on Machine Learning*, pages 280–296, 1993.

[83] L. D. Whitley and J. D. Schaffer, editors. *Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks.* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[84] G. Winter, J. Périaux, M. Galán, and P. Cuesta, editors. *Genetic Algorithms in Engineering and Computer Science.* John Wiley & Sons, New York, 1995.

[85] R. R. Yager, editor. *Essentials of Fuzzy Modeling and Control.* John Wiley & Sons, New York, 1995.