# Implementation of Evolutionary Fuzzy Systems

Yuhui Shi, *Senior Member, IEEE*, Russell Eberhart, *Senior Member, IEEE*, and Yaobin Chen, *Member, IEEE*

***Abstract*— In this paper, evolutionary fuzzy systems are discussed in which the membership function shapes and types and the fuzzy rule set including the number of rules inside it are evolved using a genetic (evolutionary) algorithm. In addition, the genetic parameters (operators) of the evolutionary algorithm are adapted via a fuzzy system. Benefits of the methodology are illustrated in the process of classifying the iris data set. Possible extensions of the methods are summarized.**

***Index Terms*—Fuzzy expert systems, genetic algorithm, membership.**

## I. Introduction

### A. Background

FUZZY systems are being used successfully in an increasing number of application areas; they use linguistic rules to describe systems. These rule-based systems are more suitable for complex system problems where it is very difficult, if not impossible, to describe the system mathematically. One of the most important considerations in designing any fuzzy system is the generation of the fuzzy rules as well as the membership functions for each fuzzy set. In most existing applications, the fuzzy rules are generated by experts in the area, especially for control problems with only a few inputs. With an increasing number of variables, the possible number of rules for the system increases exponentially, which makes it difficult for experts to define a complete rule set for good system performance. An automated way to design fuzzy systems might be preferable.

There are many ways to attack this problem. A straightforward approach is to use clustering algorithms (like the c-means clustering algorithm, fuzzy c-means clustering algorithm, etc. [2]) or similar methods to partition the pattern space into many subspaces with or without overlaps among them, then map the center of each cluster into a rule according to the definitions of fuzzy variables [1], [24]. One disadvantage of this approach is that the extracted rules are independent of the membership functions so there is no guarantee that the fuzzy system obtained will have sufficiently good performance, especially for a complex system problem with a large number of input variables. In many cases, however, the system's performance can be improved by further tuning the membership functions and selecting suitable fuzzification and defuzzification methods.

The design of a fuzzy system can be formulated as a search problem in high-dimensional space where each point represents a rule set, membership functions, and the corresponding system's behavior. Given some performance criteria, the performance of the system forms a hypersurface in the space. Developing the optimal fuzzy system design is equivalent to finding the optimal location of this hypersurface. The hypersurface has the following characteristics.

- The hypersurface is *infinitely large* since the number of possible fuzzy sets for each variable is unbounded.
- The hypersurface is *nondifferentiable* since changes in the number of fuzzy sets are discrete and can have a discontinuous effect on the fuzzy system's performance.
- The hypersurface is *complex* and *noisy* since the mapping from a fuzzy rule set to its performance is indirect and dependent on the evaluation method used.
- The hypersurface is *multimodal* since different fuzzy rule sets and/or membership functions may have similar performance.
- The hypersurface is *deceptive* since similar fuzzy rule sets and membership functions may have quite different performances.

These characteristics seem to make evolutionary algorithms such as genetic algorithms (GA's) better candidates for searching the hypersurface than conventional methods such as hill-climbing search methods.

### B. Previous Approaches

GA's are commonly used evolutionary algorithms that provide a way to search poorly understood, irregular spaces. One of the key issues in the evolutionary design of fuzzy systems using GA's is their genotype representation; that is, what is encoded into the chromosomes.

Thrift [23] and Hwang and Thompson [11] encode all the rules into the chromosome while fixing the membership functions. Using several critical points to represent each membership function while using all the possible rules, Karr [15] and Karr and Gentry [16] use GA's to evolve these critical points; that is, to adjust the membership functions. Since in a fuzzy system the membership functions and rule set are codependent, they should be designed or evolved at the same time. Homaifar and McCormick [10] use GA's to tune the membership functions and evolve the rule set at the same time. The base length of each triangular membership function and all possible rules are encoded into the chromosomes. Similar to [10], Lee and Takagi [17] also encode membership

Manuscript received September 23, 1997; revised September 22, 1998. This work was supported by Delphi Engine and Energy Management.

The authors are with the Purdue School of Engineering and Technology, Indiana University/Purdue University Indianapolis, Indianapolis, IN 46202 USA.

Publisher Item Identifier S 1063-6706(99)02801-5.

functions and all the rules into the chromosome, but have a different way to encode the triangular membership functions. They restrict adjacent membership functions to fully overlap and also constrain one membership function to have its center resting at the lower boundaries of the input range. By using this coding, only $n-1$ membership function centers need to be encoded, where $n$ is the maximum number of partitions for a given dimension. The above-mentioned methods encode all possible rules into the chromosome. There are some drawbacks by doing so [3]: first, the computational efficiency associated with fuzzy logic is lost using a high number of rules and second, the robustness decreases with increasing the number of rules. This is especially true when the dimension of the inputs and the number of fuzzy sets for each input variable become great since the number of possible rules exponentially increases with the these numbers.

In most applications, not all possible rules need to be used; only a portion of the rules are needed. So only this portion of rules should be encoded into the chromosome and evolved. By doing so, the length of the chromosome will be reduced greatly and, therefore, will be suitable for larger problems. Karr [14] considers a very special case where the number of rules is provided by an expert, together with many complete rules forming the rule set and the antecedents for the remaining ones so only the consequent parts of the latter type need to be evolved and, therefore, need to be encoded in the chromosome. This is not the case for many applications. Most of the time, it will be difficult, if not impossible, to know *a priori* exactly how many rules are required to be included in the rule set; only a maximal number can be guessed or estimated.

It is better to encode the number of rules to be included in the rule set together with rules and/or membership functions into the chromosome to be evolved. There are several ways to do this. Lee and Takagi [18], [19] proposed encoding membership functions and fitness functions in chromosomes. Shimojima *et al*. [22] and Inoue *et al.* [12] defined membership functions for each rule and encoded effectiveness information for each rule and membership function. Shimojima *et al.* used fitness functions that encouraged minimizing the number of rules; Inoue *et al.* used a method they called ''forgetting.''

Due to the highly complex and nonlinear characteristic of the problem space, uniform distribution of the fuzzy sets is usually not optimal. The performance of a fuzzy classification system based on fuzzy if–then rules depends on the choice of a fuzzy partition. If a fuzzy partition is too coarse, the performance may be low. If a fuzzy partition is too fine, many fuzzy if–then rules cannot be generated because of the lack of training patterns in the corresponding fuzzy subspaces. For a problem, some parts of pattern space might require fine partition, while other parts require only coarse partition. Therefore, the choice of an appropriate fuzzy partition is important and difficult. To cope with this difficulty, Ishibuchi, Nozaki *et al.* [13] introduce the concept of distributed fuzzy if–then rules. They encode all fuzzy if–then rules corresponding to several different fuzzy partitions into a tri-value string $\{-1, 0, 1\}$ and apply GA's to remove the unnecessary rules from fuzzy if–then rules corresponding to the different fuzzy partitions. Since each possible rule for each subspace is coded into the chromosome, the length of the chromosome is very large when the number of input dimensions and/or of different partitions is large. Other ways to tackle the nonlinear distribution problem should be sought. A natural and better way is to employ nonlinear functions in addition to linear functions as membership functions. Natural choices are *Gaussian functions*, *sigmoid functions*, etc. Through inclusion of linear and nonlinear functions, the type of membership function for each fuzzy set will not be predetermined, but instead be evolved during the design process.

GA behavior is determined by the exploitation and exploration relationships kept throughout the run [8]. Given fixed settings for parameters such as crossover and mutation rates through the run, the GA may have its exploitation/exploration relationship (EER) disproportioned and produce a lack of diversity in the population [8]. Accordingly, GA parameter settings should be adapted through the run. Since the interaction between GA parameter settings and GA performance is complex and unknown, finding algorithms to achieve optimal adaptive parameter settings is very difficult, if not impossible. This suggests the use of fuzzy systems for adapting GA parameters. The main idea is to use a fuzzy system whose inputs are any combination of GA performance measures or current control parameters and whose outputs are GA control parameters [8]. Lee and Takagi [17] propose an automatic learning technique to design fuzzy rules for tuning GA's. Due to the heavy computation requirement, they first apply this technique to design a fuzzy system to tune a GA for solving the simple DeJong F1 function, then apply the obtained fuzzy system to tune the GA to solve other different and more complex problems. This technique is very similar to the meta GA of Grefenstette [7]. This implies that the robustness of the obtained fuzzy rules strongly depends on the problems to be solved and the performance measures used by the technique. As we know from the literature, there exist a lot of adaptive GA's to tackle the lack of diversity problem and a certain body of expertise, experience, and knowledge on GA's has become available as a result of empirical studies conducted over a number of years [9]. This human expertise and knowledge is very useful and should be the first choice for designing a fuzzy system to tune a GA to reach a suitable EER for avoiding premature convergence and improving GA behavior.

### C. Current Approach

In this paper, a GA-based method to evolve a fuzzy expert system is discussed. It not only can evolve the rule set (including the optimal number of rules inside the rule set), tune the membership functions, and evolve the membership function types, but also scales well and is, therefore, useful for large complex problems. In addition, a fuzzy expert system is designed from our experience and knowledge and is used to adapt the genetic parameters of the GA.

The paper is organized as follows. Section II describes the GA. Section III describes the fuzzy expert system. In Section IV, details are given on how to design the fuzzy system using a GA. In Sections II–IV, the implementations described are related to the simulation example of Section V. Example results are given in Section V, which demonstrate

that the method discussed in this paper is effective and efficient.

## II. Genetic Algorithms

GA's are search algorithms that reflect in a *primitive* way some of the processes of natural evolution including crossover, mutation, and survival of the fittest. They are analogous to neural networks' status as primitive approximations to biological neural processing. GA's provide powerful search mechanisms that can be used in optimization or classification applications. While stochastic in nature, GA's perform a highly effective search of the problem hyperspace, efficiently directing the search to promising regions. GA paradigms are effective in a wide variety of applications; they are not designed to solve only a narrow class of problems. GA's work with a *population* of points rather than a single point. Each ''point'' is a vector in hyperspace representing one potential (or candidate) solution to the optimization problem. A population is, thus, just an ensemble or set of hyperspace vectors. Each vector is called a *chromosome* in the population. The number of elements in each vector (chromosome) depends on the number of parameters in the optimization problem and the way to represent the problem. How to represent the problem as a string of elements is one of the critical factors in successfully applying a GA (or other evolutionary algorithm) to a problem.

GA paradigms do not require information that is auxiliary or related to the problem such as function derivatives, while many hill-climbing search paradigms, for example, require the calculation of derivatives in order to successfully explore the local maximum or minimum. So GA's can be applied to wider areas, especially those difficult for traditional hill-climbing methods. A typical series of operations carried out when implementing a GA paradigm is:

1) initialize the population;
2) calculate fitness for each chromosome in population;
3) reproduce selected chromosomes to form a new population;
4) perform crossover and mutation on the population;
5) loop to step 2) until some condition is met.

Initialization of the population is commonly done by seeding the population with random values. The fitness value is proportional to the performance measurement of the function being optimized. The calculation of fitness values is conceptually simple. It can, however, be quite complex to implement in a way that optimizes the efficiency of the GA's search of the problem space. It is this fitness that guides the search of the problem space.

It is not unusual for most (if not all) of the fitness values after, say, a few dozen to a few hundred generations, to be quite high. In cases where the fitness value can range from 0 to 1, for example, most or all of the fitness values may be 0.9 or higher. This lowers the differential between fitnesses that provides the impetus for effective reproduction, i.e., ensuring that higher fitness values have a significantly higher probability of reproduction. One way around this problem is to shift the fitness values in some manner.

After fitness calculation, the next step is reproduction. Reproduction comprises forming a new population, usually with the same total number of chromosomes, by selecting from members of the current population using a stochastic process that is weighted by each of their fitness values. The higher the fitness, the more likely it is that the chromosome will be selected for the new generation. One commonly used way is a ''roulette wheel'' procedure that assigns a portion of a roulette wheel to each population member where the size of the portion is proportional to the fitness value. This procedure is often combined with the *elitist strategy*, which ensures that the chromosome with the highest fitness is always copied into the next generation.

The next operation is called crossover. To many evolutionary computation practitioners, crossover is what distinguishes a GA from other evolutionary computation paradigms. Crossover is the process of exchanging portions of the strings of two ''parent'' chromosomes. An overall probability is assigned to the crossover process, which is the probability that given two parents, the crossover process will occur. This probability is often in the range of 0.65–0.80. The final operation in the typical GA procedure is mutation. Mutation consists of changing an element's value at random, often with a constant probability for each element in the population. The probability of mutation can vary widely according to the application and the preference of the person exercising the GA. However, values of between 0.001 and 0.01 are not unusual for mutation probability.

In the example simulation in this paper, the ''roulette wheel'' procedure with the *elitist strategy* [6] is used for reproduction, where the portions of the roulette wheel assigned to population members are proportional to the *shifted* fitness values [4]. The original fitness values are linearly *shifted* with the minimal fitness mapping to 0.1. The crossover operator used is two-point crossover with a default crossover probability of 0.75 [6]. The mutation operator used in this paper depends on our chromosome representation and will be explained later. Note that in our evolutionary fuzzy system described in Section IV, fuzzy rules can be used to adapt crossover probability and mutation rate.

## III. Fuzzy Expert Systems

Fuzzy logic provides a general concept for description and measurement. Most fuzzy logic systems encode human reasoning into a program to make decisions or control a system. Fuzzy logic comprises fuzzy sets, which are a way of representing nonstatistical uncertainty and approximate reasoning, which includes the operations used to make inferences in fuzzy logic. Unlike traditional Aristotelian two-valued logic, in fuzzy logic, fuzzy set membership occurs by degree over the range [0,1], which is represented by a membership function. It is this function that is the fuzzy set. The function can be linear or nonlinear. Commonly used are *left_triangle*, *right_triangle*, *triangle*, *Gaussian*, and *sigmoid* functions, as shown in Fig. 1. Definitions of these membership functions as used in this paper are as follows.

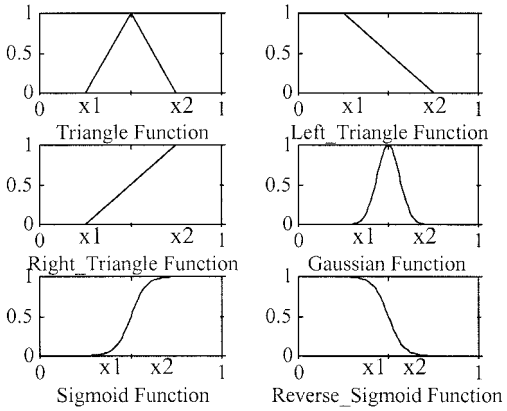



Fig. 1. Six commonly used membership functions.

*Left_triangle membership function*

$$f_{left\_triangle} = \begin{cases} 1, & \text{if } x < x_1 \\ \dfrac{x_2 - x}{x_2 - x_1}, & \text{if } x_1 \le x \le x_2 \\ 0, & \text{if } x > x_2 \end{cases}$$

*Right_triangle membership function*

$$f_{right\_triangle} = \begin{cases} 1, & \text{if } x < x_1 \\ \dfrac{x - x_1}{x_2 - x_1}, & \text{if } x_1 \le x \le x_2 \\ 0, & \text{if } x > x_2 \end{cases}$$

*Triangle membership function*

$$f_{triangle}(x) = \begin{cases} 0, & \text{if } x < x_1 \\ 2\dfrac{x - x_1}{x_2 - x_1}, & \text{if } x_1 \le x \le \dfrac{x_2 + x_1}{2} \\ 2\dfrac{x_2 - x}{x_2 - x_1}, & \text{if } \dfrac{x_2 + x_1}{2} < x \le x_2 \\ 1, & \text{if } x > x_2 \end{cases}$$

*Gaussian membership function*

$$f_{Gaussian}(x) = e^{-0.5y^2} \quad \text{where } y = \frac{8(x - x_1)}{x_2 - x_1} - 4$$

*Sigmoid membership function*

$$f_{sigmoid}(x) = \frac{1}{1 + e^{(-y+6)}} \quad \text{where } y = \frac{12(x - x_1)}{x_2 - x_1}$$

*Reverse_sigmoid membership function*

$$f_{reverse\_sigmoid}(x) = 1 - f_{sigmoid}(x).$$

Other definitions are possible, of course, but the authors have found these to be useful for a variety of problems. From the definitions, it can be seen that each membership function is determined by two values—the *start_point* $x_1$ and the *end_point* $x_2$.

Theoretically, each fuzzy variable can have many fuzzy sets with each having its own membership function, but commonly used are three, five, seven, or nine fuzzy sets for each fuzzy variable. Fig. 2 shows a fuzzy variable *temperature* having five fuzzy sets with triangular membership functions.

The general form of a Mamdani-type fuzzy rule in a fuzzy expert system is

If $x_1$ is $S_1$, and $x_2$ is $S_2, \ldots, x_k$ is $S_k$,
then $y_1$ is $T_1, \ldots,$ and $y_1$ is $T_1$

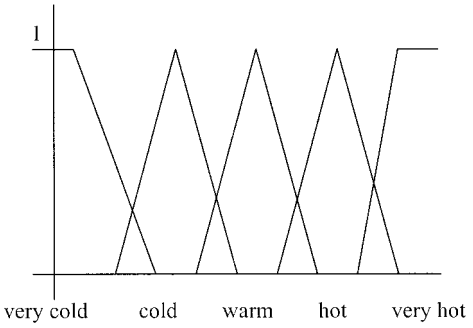



Fig. 2. Membership functions of variable temperature.

where each $y_i$ is the consequent (output) variable whose value is inferred, each $x_i$ is an input variable (an antecedent), and each $S_i$ and $T_i$ is a fuzzy set represented by a membership function. For simplicity, only Mamdani-type fuzzy rules are considered in this paper. A fuzzy system is defined if and only if its rule set and its membership functions associated with its fuzzy sets are defined. An example definition of a fuzzy system as used in this paper is given in List I.

The number ''5'' in the first line specifies the number of rules listed in the rule set. The next line contains the number of input fuzzy variables (4) followed by the number of output fuzzy variables (1). Next, the fuzzy sets for all input and output variables are defined. In accordance with the second line, we define four input and one output fuzzy variables. The next line, *input_one 5 0.4 1.0*, defines the first fuzzy input variable's name as *input_one*, specifies the variable's domain to have five fuzzy sets and defines the variable's dynamic range (domain) to be 0.4–1.0. The dynamic range is generally related to the variable's minimum and maximum, respectively, in the data set. In the next five lines, each line defines one of the fuzzy sets for the variable *input_one*. Two main kinds of membership functions are available: nonlinear and linear. The variable *input_one* is represented by the nonlinear fuzzy membership functions *Gaussian, Triangle, Triangle, Triangle* and *Gaussian* with the first value in the same row being *start_point* $x_1$ and the second being *end_point* $x_2$, respectively. The other three input variables, *input_two*, *input_three*, *and input_four*, are specified following the first input variable. Then comes the output variable, which is named output in the List I and is defined by five fuzzy membership functions over the domain [0,1]. The next five lines define the five fuzzy rules included in the rule set as specified by the number ''5'' in the first line, the meaning of which will be explained in the next section. All the fuzzy rules in a fuzzy expert system are (theoretically) fired in parallel. The fuzzy expert system works as follows.

1) Determine the fuzzy membership values activated by the inputs.
2) Determine which rules are fired in the rule set.
3) Combine the membership values for each activated rule using the AND operator.
4) Trace rule activation membership values back through the appropriate output fuzzy membership functions.
5) Utilize defuzzification to determine the value for each output variable.
6) Make decision according to the output values.

LIST I
AN EXAMPLE OF A FUZZY EXPERT SYSTEM

```
5
4       1
input_one       5       0.4     1.0
        Gaussian        0.4075  0.625
        Triangle        0.61    0.70
        Triangle        0.65    0.75
        Triangle        0.72    0.85
        Gaussian        0.745   0.925
input_two       5       0       0.6
        Gaussian        0.015   0.285
        Gaussian        0.105   0.295
        Gaussian        0.15    0.32
        Triangle        0.305   0.495
        Triangle 0.24   0.5325
input_three     5       0       1
        Gaussian        0.075   0.4
        Triangle        0.175   0.52
        Triangle        0.25    0.42
        Triangle        0.35    0.625
        rightTriangle   0.475   0.9375
input_four      5       0       0.4
        leftTriangle    0.04    0.22
        Triangle        0.11    0.25
        Triangle        0.15    0.28
        Traingle        0.30    0.32
        Gaussian        0.25    0.37
output  5       0       1
        Triangle        0.075   0.45
        Triangle        0.375   0.48
        Triangle        0.43    0.52
        Triangle        0.50    0.55
        Sigmoid         0.65    0.85
-2      -3      0       4       5
3       1       5       -3      1
1       -3      -2      -2      -3
-4      -1      1       3       1
-3      -1      3       -3      -3
```

Determination of the fuzzy membership values activated by the inputs is often called fuzzification. Each input may activate one or more fuzzy sets of that input variable according to the definitions of the fuzzy membership functions. Only the rules with at least one antecedent set activated are said to be fired by the inputs. The **AND** operator is typically used to combine the membership values for each fired rule to generate the membership values for the fuzzy sets of output variables in the *consequent* part of the rule. Since there may be several rules fired in the rule sets, for some fuzzy sets of the output variables there may be different membership values obtained from different fired rules. There are many ways to combine these values. One commonly used way is to use the **OR** operator, that is to take the maximum value as the membership value of that fuzzy set. Next, a defuzzification method is used to produce a single scalar value for each output variable. A common way to do the defuzzification is called the *clipped center of gravity* or *centroid* method. Then according to the output values, some decisions can be made to solve the problem. For example, for the $n$-class classification problem, the output variable range can be divided into $n$ evenly distributed parts, then the input pattern belongs to class $I$ if the inferred output value is located inside the $i$th part. This is the approach taken for the iris data set classification system described in this paper.

## IV. EVOLUTIONARY FUZZY SYSTEMS

### A. Representation

When designing a fuzzy system using a GA, the first important consideration is the representation strategy, that is how to encode the fuzzy system into the chromosome. A fuzzy system is specified only when the rule set and the membership function associated with each fuzzy set are specified. To completely represent a fuzzy system, each chromosome must encode all the needed information about the rule set and the membership functions as shown in List I. For the purpose of discussion, assume that we have a fuzzy system like the one shown in List I with four input variables and one output variable, and that each variable has five fuzzy sets representing the linguistic descriptions: *very low, low, medium, high,* and *very high*. We can use the integers 1–5 to represent each of

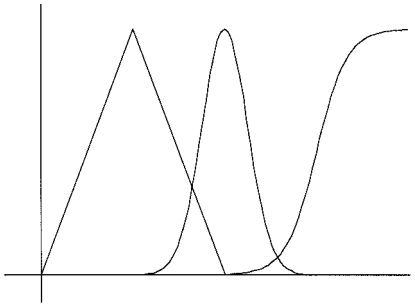


Fig. 3. Three uniformly distributed membership functions.

these five terms, use the integer 0 to represent the absence of a term, and use a minus sign ''$-$'' to encode the term ''not.'' For example, $-1$ means ''not very low'' while 1 means ''very low.'' In this way, a fuzzy rule can be completely represented by eleven integers. For example, the rule if *input_1 is not low, input_2 is not medium, and input _4 is high, then output is very high* can be encoded as $-2-3045$. (Note, only Mamdani-type rules are considered here for simplicity.) If the rule set includes 20 rules, then an integer string of length 100 can represent the rule set completely.

In this paper, a total of six types of functions (defined in the last section) are used as the membership function candidates; each is represented by an integer from 1 to 6. A membership function in our example is completely determined by three values: the *start_point* $x_1$, the *end_point* $x_2$, and the function type value. In order to have a homogeneous chromosome, integers are chosen to represent the *start_point* $x_1$ and the *end_point* $x_2$ instead of real values. Assume for the variable $x$ that its dynamic range is [*a, b*] and that it has $n$ fuzzy sets. If the fuzzy membership functions are uniformly distributed over the range with half-way overlap as shown in Fig. 3, then the center point $c_i(i = 1, \cdots, n)$ of the *i*th membership function is located at

$$c_i = a + i * \text{step} \qquad i = 1, \cdots, n, \quad \text{where} \quad \text{step} = \frac{b-a}{n+1}.$$

We constrain the *start_point* $x_1^i$ of the *i*th membership function to vary only between $c_{i-1}$ and $c_i$, and the *end_point* $x_2^i$ of the *i*th membership function can vary only between $c_i$ and $c_{i+1}$. Assume an integer $s(s = 0, \cdots, 10)$ is used to represent $x_1^i$ and $x_2^i$, then $x_1^i$ and $x_2^i$ can be calculated from the integer $s$ using the following formula:

$$x_1^i = i * \text{step} - \frac{\text{step} * (10 + s)}{2 * 10} + a$$
$$x_2^i = i * \text{step} + \frac{\text{step} * (10 + s)}{2 * 10} + a, \qquad i = 1, \cdots, n.$$

For an unknown fuzzy system, we generally have no idea how many rules should be included in the rule set before the system is designed. A maximum acceptable number can be guessed and/or given, however. Within the maximum number constraint, the number of fuzzy rules in the rule set should also be evolved. Assume for our example system that the maximum acceptable number is 30, then the total length (in integers) of the chromosome representing the system is

$$1 + 5 * (5 * (2 + 1)) + 5 * 30 = 226$$

and the system can be represented as

$$s_1 s_2 s_3 s_4 \cdots s_{14} s_{15} s_{16} s_{17} \cdots s_{76} s_{77} s_{78} s_{79} s_{80} s_{81} \cdots$$
$$s_{222} s_{223} s_{224} s_{225} s_{226}$$

where $s_1$ represents the number of rules varying between 1 and 30, $s_2$, $s_3$ represent the start point and end point for the first fuzzy set of the first input variable and can vary between zero and ten, $s_4$ represents the membership function type for the first fuzzy set of the first input variable and can vary between one and six, $s_5$ to $s_{76}$ encode the remaining fuzzy membership functions (start point, end point, type), $s_{77}$ to $s_{81}$ represent the first fuzzy rule and $s_{222}$ to $s_{226}$ represent the last possible rule. Since $s_1$ specifies how many possible rules are encoded in the chromosome, only the first $s_1$ rules are used to form the rule set, but it may be that not each of them is feasible. Each possible rule is therefore checked to see whether it represents a feasible rule or not. A rule without a nonzero antecedent or consequent part is not a feasible rule and will not be included in the rule set. For example, assume we have a rule encoded as 12320. This has no nonzero consequent part, so it will not be included in the rule set as a rule and the number of feasible rules will be $s_1 - 1$. If all $s_1$ possible rules are infeasible (this mostly happens at the beginning of the GA run), then this chromosome contains no feasible rules, does not form a usable fuzzy system, and is assigned a small (around 0.0001) positive random value as its fitness value.

### B. Fitness Function

The next important consideration following the representation is the choice of the fitness function. The genotype representation encodes the problem into a string while the fitness function measures the performance of the system. To find a good fitness measurement for a system is quite important for evolving practical systems using GA's. Unlike traditional gradient-based methods, GA's can be used to evolve systems with any kind of fitness measurement functions including those that are nondifferentiable, discontinuous, etc. Finding a good fitness measurement can make it easier for the GA to evolve a useful system. How to define the fitness measurement function for a system to be evolved is problem dependent. For prediction and estimation problems, a commonly used function is a mean-square error or absolute difference error related function

$$E = \frac{1}{N} \sum_{i=1}^{N} (o_i - t_i)^2$$
$$E = \frac{1}{N} \sum_{i=1}^{N} |o_i - t_i|$$

where $N$ is the number of training data, and $o_i$ and $t_i$ are the *i*th obtained and target outputs, respectively. Assume the maximum error is $E_{\max}$; then one possible fitness function is

$$F = E_{\max} - E.$$

These kind of error functions reflect absolute error; that is, the error is dependent only on the difference between the obtained

output and target output. For example, for a target output of ten and obtained output of nine, and for a target output of two and obtained output of one, the same absolute error of one results and each will have the same contribution to the error function. But nine is generally a better estimation of ten than one is of two. Using the above error functions to calculate the fitness, the obtained system will thus have better accuracy for large target outputs than for small ones. To have the system have similar accuracy for any target output, relative error functions can be introduced. Examples of these functions are

$$E = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{o_i - t_i}{t_i}\right)^2$$

$$E = \frac{1}{N}\left|\frac{o_i - t_i}{t_i}\right|.$$

For classification problems, the above error functions are generally not good candidates. If class **A** is misclassified as class **B**, the same error results as when **A** is misclassified as any other class. Normally, for classification problems, the number of misclassified classes and the number of correctly classified classes are used in the error function and fitness functions. This approach is used in the problem addressed later in this paper. Other requirements for the system can also be encoded into the fitness function. For example, if we prefer a fuzzy system with a low number of rules, then the rule number can be encoded as a factor of the fitness normally through summation.

### C. Mutation Operator

From the encoding methods illustrated, we see that the chromosome representing the fuzzy expert system is integer based instead of binary based. Each element in a chromosome has an integer range according to which component or parameter it is representing in the original fuzzy system. For example, $s_1$ has a integer range from 1 to 30 to encode the number of rules inside the rule set; $s_{77}$ has a integer range from $-5$ to 5 to encode the selected fuzzy set; etc. The mutation operator used is thus a little different than that used in a binary encoding. Each time an element is chosen to be mutated, it is increased or decreased by one randomly inside its range. The mutation rate used is fixed at 0.01 if it is not adapted during the GA run.

### D. Adaptive Genetic Algorithm

Crossover and mutation are two critical operators. Although they are potentially disruptive, they facilitate an efficient search and guide the search into new regions. Crossover facilitates exploration, while mutation facilitates exploitation of the space. The probabilities of crossover and mutation are often held constant for the entire run of a GA, although this approach will not produce optimal results in many cases. They can be varied during the run, often starting out by running the GA with a relatively higher value for crossover and lower value for mutation, then tapering off the crossover value and increasing the mutation rate toward the end of the run, ending with values of, say, one half and twice the initial values, respectively.

LIST II
DEFINITION OF THE FUZZY SYSTEM THAT ADAPTS GA PARAMETERS

| | | | | |
|---|---|---|---|---|
| 8 | | | | |
| 3 | 2 | | | |
| BF | 3 | 0.0 | 1.0 | |
| | leftTriangle | | 0.0 | 0.7 |
| | Triangle | | 0.5 | 0.9 |
| | rightTriangle | | 0.7 | 1.0 |
| UN | 3 | 0 | 20 | |
| | leftTriangle | | 0 | 6 |
| | Triangle | | 3 | 9 |
| | rightTriangle | | 6 | 12 |
| VF | 3 | 0.0 | 0.2 | |
| | leftTriangle | | 0.0 | 0.12 |
| | Triangle | | 0.1 | 0.14 |
| | rightTriangle | | 0.12 | 0.2 |
| MR | 3 | 0.005 | 0.1 | |
| | leftTriangle | | 0.005 | 0.015 |
| | Triangle | | 0.01 | 0.02 |
| | rightTriangle | | 0.015 | 0.1 |
| CR | 3 | 0.4 | 0.9 | |
| | leftTriangle | | 0.48 | 0.65 |
| | Triangle | | 0.55 | 0.75 |
| | rightTriangle | | 0.65 | 0.83 |
| 1 | 0 | 0 | 1 | 3 |
| 2 | 1 | 0 | 1 | 3 |
| 2 | 2 | 0 | 2 | 2 |
| 0 | 3 | 2 | 3 | 1 |
| 3 | 1 | 0 | 1 | 3 |
| 3 | 2 | 0 | 2 | 2 |
| 0 | 3 | 1 | 3 | 1 |
| 0 | 3 | 3 | 1 | 3 |

Since the process to vary these two parameters to obtain good performance is unknown, it is unclear how to vary the parameters during the run. Normally, they are changed linearly. From experience, we know when the fitness is high, e.g., at the end of the run, low crossover rate and high mutation rate are often preferred. Also, when the best fitness is stuck at one value for a long time, the system is often stuck at a local minimum in a local neighborhood, so the system should probably concentrate on exploiting rather than exploring; that is, the crossover rate should be decreased and mutation rate should be increased. A similar situation exists for the variance of the fitnesses of the population. When variance is low, mutation should be emphasized, while when variance is high, crossover should be stressed. According to this kind of knowledge, we develop a fuzzy system to adjust the crossover and mutation rates with best fitness (BF), number of generations for unchanged best fitness (UN), and variance of fitnesses (VF) as the input variables, and mutation rate (MR) and crossover rate (CR) as output variables. For simplicity, each variable has three fuzzy sets: *low*, *medium*, and *high*. The definition of the fuzzy system that adapts GA parameters is given in List II.

Eight fuzzy rules are used to adjust the mutation and crossover rates. For clearness, the linguistic descriptions of these eight rules are listed below.

- If BF is *low*, then MR is *low* and CR is *high*.
- If BF is *medium* and UN is *low*, then MR is *low* and CR is *high*.
- If BF is *medium* and UN is *medium*, then MR is *medium* and CR is *medium*.

- If UN is *high* and VF is *medium*, then MR is *high*, and CR is *low*.
- If BF is *high* and UN is *low*, then MR is *low* and CR is *high*.
- If BF is *high* and UN is *medium*, then MR is *medium* and CR is *medium*.
- If UN is *high* and VF is *low*, then MR is *high* and CR is *low*.
- If UN is *high* and VF is *high*, then MR is *low* and CR is *low*.

Other rules and definitions of membership functions are possible, of course. These were selected based upon the experience of the authors for the example described in the next section. They have also proven useful for other problems.

## V. Simulation and Discussion

The implementation of the evolutionary fuzzy system is written in C++ and compiled using the Borland C++ 4.5 compiler. An example system has been evolved for the classification of the well-known iris data set, which consists of feature measurements for iris flowers measured by Anderson and popularized by Fisher [5]. It consists of 150 four-dimensional vectors representing 50 plants of each of three species *iris setosa, iris versicolor,* and *iris virginica*

$$x_i = (x_{i1}, x_{i1}, x_{i3}, x_{i4}), \qquad i = 1, \cdots, 150$$

where $x_{i1}$ is the sepal length, $x_{i2}$ is the sepal width, $x_{i3}$ is the petal length, and $x_{i4}$ is the petal width [5]. All of the attribute values have been normalized into real numbers in the range [0,1]. The problem is to discriminate the species according to the feature vectors. Thus, this is a three-class classification problem. For this problem, the fitness chosen is the percent correct.

Assuming each variable has three fuzzy sets associated with triangular membership functions uniformly distributed over [0,1], we originally used learning vector quantization (LVQ) [4] to extract fuzzy rules from the data set. First, LVQ was used to cluster the iris data into 16 subclasses. Using the centers of the 16 subclasses and the definitions of the fuzzy sets, 16 rules were formed. Since there were three occurrences of the same rule for two out of the 16 rules, all but one was eliminated due to the fuzzy operators used in this paper. Also there were three rules which were merged into one rule by using zero to represent the absence of a term. A fuzzy system with ten rules resulted. This system gave 17 errors out of 150 patterns classified.

Next, we adjusted the membership functions' shapes manually. By trial-and-error, the best system we evolved gave 11 errors. This is still not acceptable for this data set compared with most other classification methods. Finally, more efforts were made to adjust the membership functions' shapes and types manually. We finally got the best system we thought we could obtain using this approach. This system gave seven errors out of 150 patterns classified. This result is acceptable for some purposes, but we were not satisfied with it, and were thus motivated to evolve the fuzzy system using a GA.

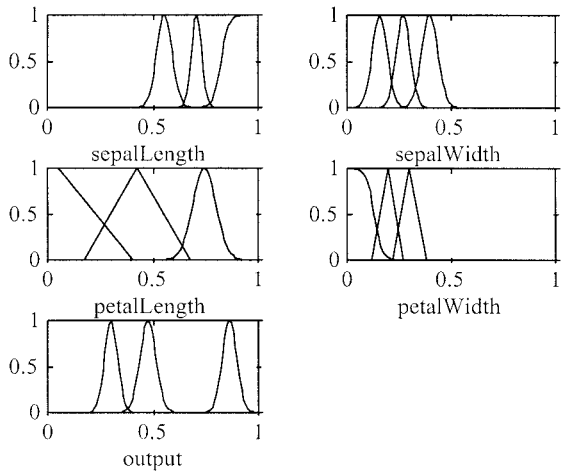



Fig. 4. Membership functions obtained for iris data.

Simulations were done to design a fuzzy classification system using the methods discussed previously for the iris data set. Three fuzzy sets were associated with each input variable. In one of the runs, a fuzzy system with four rules was evolved, which yields only three misclassifications. This is a good result compared with results obtained by most other classification methods. The four rules are as follows with the membership functions as shown in Fig. 4.

1) *If sepal length is high and sepal width is not low and petal length is high and petal width is low then output is high.*
2) *If sepal width is not medium and petal length is not high and petal width is high then output is not high.*
3) *If sepal length is not medium and sepal width is high and petal length is not medium and petal width is medium then output is medium.*
4) *If sepal length is not high and petal length is low then output is low.*

.Since the classification system has been developed using fuzzy rules, we can explain each classification and understand the relationship between the inputs (the four features of an iris flower) and the output (the iris flower's specie). This is an important advantage to evolving a fuzzy rule system.

Note the output representation. There is a single output with three fuzzy sets. The centroid method of defuzzification is used. The classifications are arbitrarily assigned to outputs zero, one, and two (low, medium, and high) in the order they appear in the data set. Although good results were obtained, it was noted that most of the time errors were due to class 1 being misclassified as class 2, or vice versa. It is possible that even better results could have been obtained by reordering the data so that classes 1 and 2 were not adjacent (results could also have been worse; reordering was not tried). It should, therefore, be noted that when using this output representation, ordering of output classes may be significant.

Since several operations were involved in evolving the final system, we attempted to (at least approximately) isolate the operations by looking at the effect of each by itself. The goal was to ensure that each was indeed contributing to improved

TABLE I
THE FUZZY RULE SET WAS EVOLVED USING FIXED MEMBERSHIP FUNCTIONS, AND WITH FIXED CROSSOVER AND MUTATION RATES (NO FUZZY SYSTEM TO ADAPT CROSSOVER AND MUTATION)

| Errors | Generation | | | | | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 18 | 16 | 18 | 13 | 18 | 21 | 15 | 9 | 10 | 22 | 16 |
| 8 | 23 | 16 | 20 | 13 | 20 | 24 | 17 | 9 | 10 | 22 | 17.4 |
| 7 | 23 | 16 | 30 | 22 | 28 | 24 | 28 | 27 | 12 | 25 | 23.5 |
| 6 | 23 | 23 | 76 | 186 | 44 | 61 | 83 | 83 | 18 | 25 | 62.2 |
| 5 | 47 | 28 | 81 | 236 | 73 | 78 | 181 | / | 18 | 255 | 110.8 |
| 4 | 215 | 60 | 428 | 527 | 137 | 84 | 492 | / | 18 | 481 | 271.4 |
| 3 | /* | / | / | / | / | / | 676 | / | / | / | 676# |

* *"/" means that the system could not find the solution within 1000 generations*
\# *the average of total generations for only those found within 1000 generations*

TABLE II
THE FUZZY RULE SET WAS EVOLVED USING FIXED MEMBERSHIP FUNCTIONS WITH A FUZZY EXPERT SYSTEM ADDED TO ADAPT THE CROSSOVER AND MUTATION RATES

| Errors | Generations | | | | | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 13 | 4 | 6 | 2 | 8 | 18 | 20 | 21 | 13 | 19 | 12.4 |
| 8 | 24 | 7 | 11 | 2 | 9 | 33 | 27 | 25 | 21 | 22 | 18.1 |
| 7 | 140 | 7 | 15 | 13 | 9 | 33 | 30 | 48 | 53 | 27 | 36.5 |
| 6 | 163 | 8 | 22 | 16 | 26 | 54 | 30 | 97 | 53 | 34 | 50.3 |
| 5 | 176 | 118 | 66 | 41 | 63 | 54 | 30 | 117 | 59 | 34 | 75.8 |
| 4 | 692 | 118 | 241 | 205 | 92 | 62 | 51 | 194 | 237 | 623 | 251.5 |
| 3 | /* | 544 | / | / | 665 | 167 | 732 | / | 350 | / | 491.6# |

* *"/" means that the system could not find the solution within 1000 generations*
\# *the average of total generations for only those found within 1000 generations*

TABLE III
THE FUZZY EXPERT SYSTEM INCLUDING MEMBERSHIP FUNCTIONS WAS EVOLVED, BUT WITHOUT FUZZY RULES TO ADAPT THE CROSSOVER AND MUTATION RATES (CROSSOVER AND MUTATION HELD CONSTANT)

| Errors | Generations | | | | | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 24 | 16 | 73 | 17 | 21 | 50 | 94 | 6 | 15 | 24 | 34 |
| 8 | 82 | 16 | 73 | 17 | 21 | 50 | 94 | 6 | 17 | 24 | 40 |
| 7 | 82 | 16 | 74 | 36 | 35 | 62 | 100 | 6 | 22 | 47 | 48.2 |
| 6 | 82 | 19 | 90 | 51 | 41 | 65 | 102 | 6 | 89 | 47 | 59.2 |
| 5 | 158 | 21 | 96 | 109 | 41 | 77 | 676 | 31 | 89 | 61 | 135.9 |
| 4 | 158 | 43 | 100 | 155 | 43 | 91 | 678 | 589 | 264 | 63 | 218.4 |
| 3 | 162 | 186 | 310 | 462 | 111 | 129 | 702 | 978 | 289 | 140 | 346.9 |

system performance. The emphasis of the simulations is on convergence speed rather than generalization. All 150 patterns were, therefore, used for system training. A division into training and test sets would be required to examine the system's ability to generalize. (The question of generalization is addressed in Shi and Eberhart [21].)

In all experiments, the number of generations needed to reach a specified number of errors (out of 150) within 1000 generations was determined for each of ten runs. These numbers appear in Tables I–IV. The right-most column of each table is the average number of generations needed to achieve the error level listed in the left-most column. If more than 1000 generations were required, the average is for only those found within 1000 generations.

First, a fuzzy expert system was evolved, but the membership functions were fixed (linear triangular functions with uniform distribution over the range) and the crossover and mutation rates were held constant during the run (no fuzzy rules were used to adapt crossover and mutation). The results are shown in Table I.

Next, the fuzzy rule set was evolved with fixed membership functions, but the fuzzy system (shown in List II) was used to adapt the crossover and mutation rates of the GA. The results are given in Table II.

Next, the fuzzy rule set and membership functions were evolved, but crossover and mutation rates were held constant during each run (no fuzzy expert system was used to adapt the crossover and mutation rates). The results are given in Table III.

Comparing Table I with Tables II–IV, we can see that the system with fixed membership functions and without a fuzzy system to adapt crossover and mutation rates requires the most generations on average and only one of the ten runs was successful in evolving a system with the best performance

TABLE IV
THE ENTIRE SYSTEM WAS EVOLVED INCLUDING MEMBERSHIP FUNCTIONS WITH A FUZZY EXPERT SYSTEM TO ADAPT THE CROSSOVER AND MUTATION RATES

| Errors | Generations | | | | | | | | | | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | 10 | 9 | 12 | 26 | 14 | 11 | 5 | 29 | 15 | 8 | 13.9 |
| 8 | 10 | 11 | 26 | 45 | 14 | 11 | 5 | 29 | 15 | 23 | 18.9 |
| 7 | 18 | 11 | 26 | 53 | 14 | 22 | 5 | 30 | 15 | 30 | 22.3 |
| 6 | 26 | 13 | 26 | 93 | 24 | 29 | 10 | 30 | 15 | 35 | 30.1 |
| 5 | 86 | 83 | 30 | 119 | 30 | 43 | 87 | 45 | 17 | 59 | 59.9 |
| 4 | 105 | 264 | 30 | 126 | 38 | 47 | 105 | 47 | 199 | 98 | 105.9 |
| 3 | 129 | 272 | 56 | 154 | 50 | 119 | 113 | 109 | 310 | 122 | 143.2 |

(three errors) within 1000 generations. Furthermore, it fails in one out of ten runs to evolve a system with four or five misclassifications.

Comparing Tables II and IV, we can see that for relatively higher numbers of errors, they take similar numbers of generations to get the same performance. In order to get better performance, however, the system with fixed membership functions requires many more generations and half of the runs fail to evolve a system with the best performance within 1000 generations.

Comparing Tables III and IV, we can see that by using a fuzzy expert system to adapt the crossover and mutation rates, significantly fewer generations are required to get the same performance compared to holding crossover and mutation constant. Therefore, it appears that a fuzzy expert system can be a good way to adapt GA operators. We have used this approach for other projects, and are continuing to explore it. Several approaches might lead to improved performance, such as the use of nonsymmetric membership functions. It would be interesting to compare this approach with other evolutionary computation paradigms such as particle swarm optimization [4] and modular evolutionary fuzzy expert systems [21], but such comparisons are beyond the scope of this paper.

## VI. CONCLUSION

In this paper, evolutionary fuzzy expert systems have been discussed in which the membership function shapes and types and the fuzzy rule set, including the number of rules inside the rule set, are evolved using a GA. (Other evolutionary algorithms such as particle swarm optimizers or evolutionary programming can also be used instead of a GA.) In addition, the genetic parameters of the evolutionary algorithm are adapted via a fuzzy expert system. Benefits of the methodology have been illustrated by a classifier for the iris data set. It appears that the method described could be useful for a wide range of classification and diagnostic problems.

In this paper, evolutionary fuzzy expert systems have been discussed in which the membership function shapes and types and the fuzzy rule set, including the number of rules inside the rule set, are evolved using a GA. (Other evolutionary algorithms such as particle swarm optimizers or evolutionary programming can also be used instead of a GA.) In addition, the genetic parameters of the evolutionary algorithm are adapted via a fuzzy expert system. Benefits of the methodology have been illustrated by a classifier for the iris data set. It appears that the method described could be useful for a wide range of classification and diagnostic problems.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for encouragement and helpful comments. Portions of this paper were adapted from *Computational Intelligence PC Tools* (Boston, MA: APP, 1996, chs. 5, 6).

## REFERENCES

[1] S. Abe and M. S. Lan, "A method for fuzzy rules exaction directly from numerical data and its application to pattern classification," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 18–28, Feb. 1995.
[2] J. C. Bezdek, "On the relationship between neural networks, pattern recognition and intelligence," *Int. J. Approx. Reasoning*, vol. 6, no. 2, pp. 85–107, 1992.
[3] O. Cordon, F. Herrera, and M. Lozano, "On the bidirectional integration of fuzzy logic and genetic algorithms," in *2nd Online Workshop Evolutionary Computat. (WEC2)*, Nagoya, Japan, 1996, pp. 13–17.
[4] R. C. Eberhart, R. W. Dobbins, and P. K. Simpson, *Computational Intelligence PC Tools.* Boston, MA: APP, 1996.
[5] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugen.*, vol. 7, pp. 179–188, 1936.
[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.
[7] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 122–128, 1986.
[8] F. Herrera and M. Lozano, "Adaptive genetic algorithms based on fuzzy techniques," in *Proc. 6th Int. Conf. Inform. Processing Management Uncertainty Knowledge-Based Syst. (IPMU)*, Granada, Spain, July 1996, pp. 775–780.
[9] R. Hinterding, Z. Michalewicz, and A., E. Eiben, "Adaptation in evolutionary computation: A survey," in *Proc. IEEE Int. Conf. Evolutionary Computat.*, Indianapolis, IN, Apr. 1997, pp. 65–69.
[10] A. Homaifar and E. McCormick, "Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 129–139, May 1995.
[11] W. R. Hwang and W. E. Thompson, "Design of intelligent fuzzy logic controllers using genetic algorithms," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Orlando, FL, June 1994, pp. 1383–1388.
[12] H. Inoue, K. Kamei, and K. Inoue, "Automatic generation of fuzzy rules using hyper-elliptic cone membership function by genetic algorithms," in *Proc. 7th IFSA World Congress*, Prague, Czech Republic, June 1997, vol. II, pp. 383–388.
[13] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 260–270, Aug. 1995.
[14] C. L. Karr, "Applying genetics," *AI Expert*, vol. 6, no. 3, pp. 38–43, 1991.
[15] ——— "Adaptive control with fuzzy logic and genetic algorithms," in *Fuzzy Sets, Neural Networks, and Soft Computing*, R. R. Yager and L. A. Zadeh, Eds. New York: Van Nostrand Reinhold, 1993.
[16] C. L. Karr and E. J. Gentry, "Fuzzy control of pH using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 46–53, Feb. 1993.
[17] M. A. Lee, H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proc. Int. Conf. Genetic Algorithms*, Urbana-Champaign, IL, July 1993, pp. 76–83.

[18] _____ "Integrating design stage of fuzzy systems using genetic algorithms," in *Proc. 2nd IEEE Int. Conf. Fuzzy Syst. (FUZZ/IEEE)*, San Francisco, CA, Apr. 1993, vol. I, pp. 612–617.
[19] _____ "Embedding *a priori* knowledge into fuzzy system design method based on genetic algorithms," in *Proc. 5th IFSA World Congress (IFSA)*, Seoul, Korea, July 1993, pp. 1293–1296.
[20] Z. Michalewicz, *Genetic Algorithms + Data Structures + Evolution Programs*, 3rd ed. Berlin, Germany: Springer-Verlag, 1996.
[21] Y. Shi and R. C. Eberhart, "Modular evolutionary fuzzy expert systems," in *Proc. IEEE Int. Conf. Evolutionary Computat. (ICEC)*, Anchorage, AK, May 1998, pp. 387–391.
[22] K. Shimojima, T. Fukuda, and Y. Hasegawa, "RBF-fuzzy system with GA based unsupervised/supervised learning method," in *Proc. Int. Joint Conf. 4th IEEE Int. Conf. Fuzzy Syst./2nd Int. Fuzzy Eng. Symp. (FUZZ/IEEE-IFES)*, Yokohama, Japan, Mar. 1995, vol. I, pp. 253–258.
[23] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms (ICGA)*, San Diego, CA, July 1991, pp. 509–513.
[24] L. X. Wang and J. M. Mendel, "Generating fuzzy rules from numerical data with application," Univ. Southern California, Los Angeles, SIPI Rep. 169, 1991.

**Yuhui Shi** (SM'98) received the Ph.D. degree in electrical engineering (specialty in computational intelligence) from Southeast University, China, in 1992.

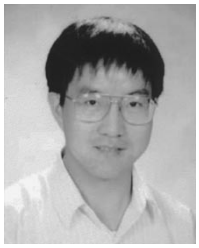
Since then, he has worked at several universities including the Department of Radio Engineering, Southeast University, Nanjing, China, the Department of Electrical and Computer Engineering, Concordia University, Montreal, Canada, the Department of Computer Science, Australian Defence Force Academic, Canberra, Australia, the Department of Computer Science, Korean Advanced Institute of Science and Technology, Taejon, Korea, and the Department of Electrical Engineering, Purdue School of Engineering and Technology, Indianapolis, IN. He is currently with Electronics Data Systems, Inc., Kokomo, IN, as an Applied Specialist. He has published approximately 20 papers and organized several special sessions in his areas of interest and has been a referee for several journals. His main interests include artificial neural networks, evolutionary computation, and fuzzy logic systems and their industrial applications.

Dr. Shi was a copresenter of the tutorial, *Introduction to Computation Intelligence*, at the 1998 WCCI Conference, Anchorage, AK, and presented the tutorial, *Evolutionary Computation and Fuzzy Systems*, at the 1998 ANNIE Conference, St. Louis, MO.

**Russell Eberhart** (SM'89) received the Ph.D. degree from Kansas State University, Manhattan, KS, in 1972.

He is Associate Dean for Research of the Purdue School of Engineering and Technology, and Professor of Electrical Engineering, at Indiana University Purdue University Indianapolis, Indianapolis, IN. He is also Executive Director of the Advanced Vehicle Technology Institute, Indianapolis, IN. He pioneered the use of neural networks for a number of biomedical engineering applications, including electroencephalogram (EEG) spike detection. He is the codeveloper of the particle swarm optimization concept. He has authored over 100 technical papers. He is coeditor of *Neural Network PC Tools* (New York: Academic, 1990), now in its fourth printing. He is coauthor of *Computational Intelligence PC Tools* (New York: Academic, 1996) and is at work on a new book, *Swarm Intelligence*. His area of personal research is in applications of neural networks, evolutionary algorithms, and computational intelligence systems.

Dr. Eberhart is Past President of the IEEE Neural Networks Council and Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He is past Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS. He has been active in IEEE conference activities, recently serving as the General Co-Chairman of the 1997 IEEE International Conference on Evolutionary Computation.



**Yaobin Chen** (S'85–M'88) received the B.S. degree from Nanjing Institute of Technology, Nanjing, China, in 1982, and the M.S. and Ph.D. degrees from Rensselaer Polytechnic Institute, Troy, NY, in 1986 and 1988, respectively, all in electrical engineering.

From 1988 to 1990, he was with the faculty of George Washington University, Washington, DC. Since 1990 he has been with the Purdue School of Engineering and Technology, Indiana University Purdue University Indianapolis (IUPUI), IN, where he is currently an Associate Professor of electrical engineering. He is a recipient of a U.S. patent. He has more than 50 technical publications in refereed journals and conference proceedings. He is currently an associate editor of the *International Journal Intelligent Automation and Soft Computing* and the *International Journal of Intelligent Control and Systems*. His areas of interest include real-time motion and motor control, neural networks and fuzzy systems and their applications, and modeling and control of hybrid and electric vehicles.

Dr. Chen was awarded the Research Initiation Award in 1991 from the National Science Foundation. He received the Teaching Excellence Recognition Award from Indiana University in 1998.