SOGARG: A Self-Organized Genetic Algorithm-Based Rule Generation Scheme for Fuzzy Controllers

Tandra Pal, *Member, IEEE*, and Nikhil R. Pal, *Senior Member, IEEE*

***Abstract*—This paper presents a self-organized genetic algorithm-based rule generation (SOGARG) method for fuzzy logic controllers. It is a three-stage hierarchical scheme that does not require *any* expert knowledge and input-output data. The first stage selects rules required to control the system in the vicinity of the set point. The second stage starts with the rules resulted from the first stage and extends its span of operation to the entire input space. Thus, the second stage ends up with a rulebase that can bring the system to its set point from almost all initial states of the input space. The third stage then refines the rulebase and reduces the number of rules in the rulebase. The first two stages use the same fitness function whose aim is only to acquire the controllability, but the last stage uses a different one, which attempts to optimize both the settling time and number of rules without compromising the controllability of the system. The mutation operations used in different stages are chosen to be different to make them consistent with the goals of different stages. The effectiveness of SOGARG is demonstrated using two control problems: the inverted pendulum and the truck back. For the inverted pendulum, rule sets contain only 16.6 rules on average, which is about 4.8% of all possible rules and it takes about 35 steps to control the system over the entire input space with an average integral time absolute error (ITAE) of 0.1019. For the truck back, we get on average 23 (6.7%) rules, and for this the average time steps and the average ITAE are, respectively, 40 and 71.42. SOGARG is found to be not sensitive to the changes in the parameters of the genetic algorithms and to the changes in the system parameters. To demonstrate the superiority of our method, we compare our results with that of Lim *et al.*, 1996 and Chan *et al.*, 2000.**

***Index Terms*—Controllability, fuzzy controllers, genetic algorithm (GA), robustness, self-organizing.**

I. Introduction

FUZZY LOGIC provides an effective means to capture the approximate, inexact nature of the real world. As systems become more and more complex, it becomes more difficult to describe them by precise mathematical models. Fuzzy logic can describe such complex systems with linguistic rules [1]–[3]. One of the most important applications of fuzzy logic is in controller design [5]–[10]. Fuzzy logic controllers (FLCs) convert the linguistic control strategy into an automatic control strategy. Experience shows that FLC yields results sometimes superior to those obtained by conventional control algorithms.

Manuscript received July 18, 2001; revised November 7, 2002 and April 17, 2003.

The authors are with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700108, India. (e-mail: tandra_v@isical.ac.in; nikhil@isical.ac.in).

Digital Object Identifier 10.1109/TEVC.2003.815377

Successful design of a rulebased fuzzy control system depends on several factors such as choice of the rule set, membership functions, inference mechanism, and the defuzzification strategy. Of these factors, selection of an appropriate rule set is more difficult because it is a computationally expensive combinatorial optimization problem. Sometimes for fuzzy controllers, rules are derived from human experts who have acquired their knowledge through experience. However, experts may not always be available; even when available extraction of an appropriate set of rules from the experts may be tedious, time consuming, and process specific. Thus, extraction of an appropriate set of rules or selection of an optimal or suboptimal set of rules from the set of all possible rules is an important and an essential step toward the design of any successful FLC.

There have been several attempts both under supervised and self-organized paradigms toward obtaining a good rulebase. Probably the first attempt to design the self-organizing fuzzy controller is due to Mamdani and his coworkers [4], [5]. The self-organizing controller (SOC) has to perform two tasks simultaneously: 1) to observe the environment while issuing the appropriate control actions and 2) to use the results of these control actions to improve them, i.e., to learn from them. In order to improve the control strategy, its performance is to be assessed. In Mamdani's approach, the performance is measured by the deviation of the actual response from the desired ones (obtained from a control engineer or an expert operator). This is expressed as a variable whose value gives a rough indication of the magnitude of the desired corrections required at the output. It is then translated into input corrections or reinforcements to the process. This is called credit assignment. Some knowledge of the process order and dead times is used to identify which past control outputs are responsible for the current poor performance, and these are used to define the correction procedure. This controller can modify a predefined set of rules, or it can start with no rules at all and learn its control policy as it goes.

Neural networks [12]–[19] and genetic algorithms (GAs) [26]–[51] have been used by several researchers for rule generation. The rulebase tuning has been attempted primarily in two ways: through tuning of membership functions of a given rule set and/or through selection of an "optimal" subset of rules from all possible rules. With increase in the number of input variables, the possible set of fuzzy rules increases rapidly. For instance, if each variable (both input and output) has p fuzzy subsets, then for a fuzzy logic controller with q inputs and one output, the total number of possible rules is p^{q+1}; of these at most a subset of p^q rules could be consistent. In our discussion, a rulebase with p^q rules will be called an *exhaustive rulebase* while a rulebase with rules fewer than p^q will be called a *reduced rulebase*. It is not an easy task to determine a small subset of rules from the set of all possible rulebases that would be suitable for controlling the process.

In this paper, we propose a method (SOGARG) to find an "optimal" rule set using genetic algorithms. It is a self-organized process, i.e., it does not require any human expert or explicit training data. The fitness function is very simple which tries to ensure that the controller operates over the entire input space with a good settling time and less number of rules in the rule set. There are three stages in the process of finding the optimal rule set. The first stage (Stage 1) yields a set of rules, which can control the system from any initial position near the set point and the second stage (Stage 2) enhances and/or modifies this rule set with a view to ensuring the controllability of the system from any initial position in the input space. The third stage (Stage 3) prunes and tunes the rule set produced by the previous stage. To demonstrate the effectiveness and superiority of SOGARG, we have implemented the inverted pendulum and truck-back control problems and compared our results with those of Lim *et al.* [34] and Chan *et al.* [42].

The rest of the paper is organized as follows. Section II contains a preliminary discussion on genetic fuzzy systems. Section III gives the philosophy and the architecture of SOGARG and Section IV describes how to implement SOGARG. Its performance evaluation along with sensitivity analysis is done in Section V. Section VI compares it with other methods. We end our report with conclusions in Section VII.

II. Genetic Fuzzy Systems: A Brief Review

Genetic algorithms (GAs) [20] are a probabilistic heuristic search process based on concepts of natural genetic systems. They are highly parallel and believed to be robust in searching global optimal solutions of complex optimization problems. They recombine structural information to locate new points in the search space with expected improved performance.

As fuzzy control systems are highly nonlinear with many input and output variables, GAs [20]–[23] are often used to optimize the control rules. Karr [26] applied GAs for learning the membership functions of fuzzy controllers. Considering isosceles triangles for the membership functions, GA, with conventional binary coding, was used to move and expand or shrink the base of each triangle. Karr used the fitness function

$$f = \sum_{i=\text{case } 1}^{\text{case } 4} \sum_{j=0 \text{ sec}}^{30 \text{ sec}} \left(w_1 x_{ij}^2 + w_2 \theta_{ij}^2\right)$$

where x and θ are the linear displacement of the cart and angular displacement of the pole, respectively, of a cart-pole system; and w_1 and w_2 are constants. Karr used four different initial conditions (case 1–case 4) for tuning the membership functions. Clearly, the performance of the system can be strongly influenced by the choice of the weights w_1 and w_2. Moreover, use of just four initial conditions may not result in a good set of membership functions to ensure the controllability of the system over the entire input domain.

Thrift [28] described the design of a two input-one output fuzzy controller for centering a cart-pole system. The alleles in the chromosome represented fuzzy sets on the output variable. The length of a chromosome was equal to the total number of combinations of input fuzzy sets. The fitness of an individual chromosome was measured by 500 $-T$, where T was the average time steps required by the controller to be sufficiently close to the set point. Thrift used ordinary two point crossover operation. The mutation operation could alter the allele value to its immediate upper level or immediate lower level or to a blank code. A blank code indicated existence of no rule corresponding to that combination of input fuzzy sets.

Nomura *et al.* [29] also used a GA to determine both the membership functions and an optimal set of rules for a single-input-single-output nonlinear system. For a single-input system the number of possible consistent rules is equal to the number of linguistic values defined on the input linguistic variable. Homaifar and McCormick [33] pointed out that the method may suffer from the constraint that the end points of a given fuzzy set are always located at the peaks of adjacent fuzzy sets.

Park *et al.* [31] first showed that a new fuzzy reasoning model (NFRM) controller can outperform the conventional fuzzy reasoning model (FRM) controller. To illustrate an application of NFRM to a dc series motor, they used two expert provided fuzzy relation matrices. Then, they showed that the performance of a NFRM controller could be enhanced using GA-based learning to derive optimal fuzzy relation matrices and fuzzy membership functions. The evaluation function used in GA was

$$J = \frac{1}{(1+e^2)}.$$

Here, the mean-square error e^2 is

$$e^2 = \frac{\sum_{i=1}^{N}(n_{ri} - n_{mi})^2}{\sum_{i=1}^{N} {n_{ri}}^2}$$

where n_{ri} is the actual speed for current i, n_{mi} is the corresponding GA tuned NFRM-produced value, and N is the number of discretization intervals of the speed. They demonstrated that if domain knowledge is used in the initialization procedure, it is exploited by the GA leading to faster convergence and better rulebase.

Herrera *et al.* [32] proposed a genetic algorithm-based tuning method for the parameters of membership functions used to define fuzzy control rules. This method relied on a set of input-output training data and minimized a squared-error function defined in terms of the training data.

Homaifar and McCormick [33] presented a method for simultaneous design of membership functions and the rule set using genetic algorithms. A GA has been used to determine the consequent fuzzy set of each possible rule and to tune the base lengths of the antecedent fuzzy sets. The peaks for the antecedent fuzzy sets and the definitions of the consequent fuzzy sets were kept unaltered. The information about the rule set and membership functions were encoded into a single chromosome. The computation of fitness was divided into two stages, an evolution stage (which lasted 30 generations) and a refinement stage. In the evolution stage, the GA was used to find satisfactory controllers. In the refinement stage, they attempted to minimize the time needed to bring the system to the set point. The fitness computation was done using a complex algorithm. This method did not need an expert's knowledge or training data and the number of rules in a chromosome was kept fixed to the number of all possible combinations of the input linguistic values, i.e., they looked for an exhaustive rulebase.

Lim *et al.* [34] described a GA-based method for learning fuzzy rules. It required no prior knowledge about the system's behavior. Given a set of linguistic values on the input and output variables, they derived a rule set having n fuzzy control rules through an adaptive learning, where n is a prespecified number. To satisfy the constraint that each chromosome must contain exactly n rules they used a special type of two-point crossover operator called positioned-aligned crossover (PAX). They also used a modification of the mutation operator due to n-rule constraint. If mutation results in $n+1$ rules, the value of a randomly chosen allele will be nullified. Similarly, if the mutation produces a chromosome with $n-1$ rules, a null allele will be altered. They used the inverted pendulum control problem for simulation and the fitness function used was

$$f = \sum_{i=1}^{d} \frac{bT_{\theta i} + (1-b)T_{ei}}{dT}.$$

Here, d denotes the total number of different initial conditions used for testing the chromosome. For the ith initial condition, $T_{\theta i}$ denotes the number of time steps that the pole retains itself within 1° from the vertical position and T_{ei} denotes the number of time steps elapsed before the pole falls or T_{ei} is equal to a prespecified value of $T(=200)$, the maximum number of time steps the controller is allowed to run. The fitness value of all chromosomes in the population are scaled using the following linear scaling function to avoid the effect of super individuals

$$F = k_1 f + k_2$$

where f and F are the fitness values before and after scaling, respectively. Suitable values of k_1 and k_2 are chosen so that $F_{\text{ave}} = f_{\text{ave}}$ and $F_{\max} = S_f \times F_{\min}$, where f_{ave} is the average fitness before scaling, F_{ave}, $F_{\max}$, and $F_{\min}$ are, respectively, the average, maximum and minimum fitness after scaling and S_f is the scaling factor.

Renhou *et al.* [36] proposed a method of optimizing different control parameters of a multi-input and multi-output fuzzy control system based on GA. They used the Takagi–Sugeno model [11], [27]

$$R_i : \text{if } \left(x_1 \text{ is } A_1^i\right) \text{ and } \left(x_2 \text{ is } A_2^i\right) \ldots \text{ and } \left(x_n \text{ is } A_n^i\right) \text{ then } \left(y_i = p_0^i + p_1^i x_1 + \cdots + p_n^i x_n\right)$$

$i = 1, 2, \ldots, m$. The final output y is computed as

$$y = \frac{\sum_{i=1}^{m} (\alpha_i \cdot y_i)}{\sum_{i=1}^{m} \alpha_i}$$

where α_i is the firing strength of the ith rule computed using the "min" operation. They employed GA to select the optimal values of the consequent as well as antecedent membership parameters. The fitness function used is

$$J = \sum_{k=1}^{N} \beta \left| y_k - y_k^d \right| \text{ or } J = \sum_{k=1}^{n} \beta \left(y_k - y_k^d \right)^2$$

where y_k and y_k^d are the output and the desired output (central point) of the process and N is the number of samples. They used a double inverted pendulum with six linguistic variables each having only two linguistic values. The number of rules in the rulebase remains the same as only the parameters of the rule set are optimized.

Carse *et al.* [35] presented a GA-based approach to design fuzzy controllers called Pittsburgh-style fuzzy classifier system (P-FCS1) based on the Pittsburgh model of learning classifier systems [24], [25]. Each rule R_k, for an n-input m-output system is expressed as

$$R_k : \text{if } (x_1 \text{ is } X_{1k}) \text{ and } (x_2 \text{ is } X_{2k}) \ldots \text{ and } (x_n \text{ is } X_{nk}) \text{ then } (y_1 \text{ is } Y_{1k}) \text{ and } (y_2 \text{ is } Y_{2k}) \ldots \text{ and } (y_m \text{ is } Y_{mk})$$

where X_{ik} and Y_{jk} are symmetric triangular fuzzy sets defined on x_i and y_j. X_{ik} is defined by the center and width (x_{cik}, x_{wik}) and similarly Y_{jk} is defined by the center and width (y_{cik}, y_{wik}). The system learns both fuzzy rules and membership functions and they are encoded in a chromosome as real numbers. The number of rules in each rule set is allowed to vary under the action of different operators. They introduced a new crossover operator, which tries to preserve the epistatical linkage between genes representing rules with overlapping fuzzy sets.

Genetic algorithm is also used by Wong and Fan [39] to automatically generate the control rules and membership functions of a fuzzy logic controller. An exhaustive rulebase with n input variables and one output variable can be expressed as

$$R(j_1, \ldots, j_n) : \text{if } x_1 \text{ is } A_{(1,j_1)} \text{ and } \ldots \text{ and } x_n \text{ is } A_{(n,j_n)} \text{ then } u \text{ is } O_{f(j_1,\ldots,j_n)}$$

where x_i and u stand for input and output linguistic variables. $A_{(i \cdot j)}$ is the jth fuzzy set of the ith input linguistic variable and $f(j_1, \ldots, j_n)$ is an index function that decides a linguistic value of u. $f(j_1, \ldots, j_n) = \langle a_1 j_1 + \cdots + a_n j_n \rangle$, where $\langle b \rangle$ denotes the integer nearest to b. Each chromosome contains the encoded parameters a_i, m_i, m, D_i, s_i, and D. The control rulebase is decided by the parameters a_i; and the membership functions are decided by the parameters m_i, D_i, m, D, and s_i. Here, m_i, D_i, and s_i are the number, length of subdivisions, and width of fuzzy sets of ith input variable, and m and D are the number and length of fuzzy sets of the output variable. The performance evaluations (rise time, overshoot, and integral absolute error) are considered in their fitness function. Modeling of the index of output linguistic value of a rule using such a linear combination of the indices of the fuzzy sets used in the antecedent clause of the rule is not meaningful. This makes the learning task unnecessarily complex and may not result in good solutions.

Wong and Her [38] proposed a method based on GA that can eliminate unnecessary fuzzy sets (linguistic values) to obtain a fuzzy system with fewer rules. But they looked for an exhaustive rulebase. The rule structure is the same as that described in [39]. They used assymetric triangular membership functions. Each chromosome contained parameters to describe both input and output membership functions. The fitness function considered the number of rules, as well as the performance of the rulebase.

Bobbin and Yao [41] proposed an evolutionary algorithm to discover good rule structures and also the rule parameters for controlling a cart-pole system. Unlike conventional approaches authors here do not start with any expert defined partition of the state space. A novel degenerate tree rule structure is used which not only allows us to evolve useful rules but also find a good discrimination of the state space. Each gene here represents a simple rule. These rules (genes) are combined using a complex tree-like structure, where each rule to the right refines the rule to the left. The rules are linked in such a manner that each member of the population defines a complete rule system giving a complete solution to the problem. In addition to simple mutation operator, which adds or deletes rules, topological mutations, which modify the order in which rules are considered are also used here.

An optimized fuzzy logic controller (OFLC) is also produced by Chan *et al.* [42] using GA. They made some modifications on simple genetic algorithm to improve its performance. The rulebase may be initialized with an expert specified suboptimal one to speed up the convergence. They used symmetric rule tables, so the first half of the string is mirrored to the other half after crossover and mutation, which are done only on the first half. The fitness function is $1/(1 + \mathrm{ITAE})$, where ITAE is the integral time absolute error.

In this context, it may be worth mentioning a few other works on reduction and/or selection of rules by GAs [45]–[49] because Stage 3 of SOGARG also has the same objective. Ishibuchi *et al.* [45] formulated a GA-based rule selection method for classification problems with two objectives: to maximize the number of correctly classified training patterns and to minimize the number of selected rules. The fitness value of a rule set S is defined as

$$f(s) = W_NCP \cdot NCP(S) - W_S \cdot |S|$$

where W_{NCP} and W_S are constant positive weights assigned to the two objectives $NCP(S)$, the number of correctly classified training patterns and $|S|$, the number of the rules in S. They also derived a set of nondominated solutions employing different values to these weights.

In the rule reduction method proposed by Krone *et al.* [46], the number of rules and the classification error were considered in the fitness function. To cope with the problems that have high dimensional search spaces, a slightly higher classification error was accepted in favor of a further rule reduction. To preserve the number of covered data sets, the fitness is multiplied with a penalty factor (related to the data points not covered by the rule set). Instead of conventional top-down approach they used bottom-up initialization, i.e., they started with a population having a few rules. In such an approach, the evaluation is expected to take less computing time due to a low average number of rules in a chromosome. But this approach cannot be used in place of Stage 3 of SOGARG. Chin and Qim [47] used GAs to search for an optimal subset of rules from a given rulebase. They showed that the reconstructed controller has a better performance than the original one. Time-weighted integral of squared errors was used to measure the performance, and the overshoot and rise time were used to show the combined performance index. Cordon and Herrera [48] presented an evolutionary process for controllers. This method also has three stages: generation, simplification, and tuning. But it requires input-output data set making it a supervised one. Our method is an unsupervised one. Roubos and Setnes [49] proposed a four stage scheme for system modeling: 1) generation of an initial rulebase by fuzzy clustering of sampled data; 2) rule reduction (by pivoted-QR decomposition); 3) rulebase simplification by finding similar fuzzy sets; and 4) constrained real coded genetic optimization to exploit redundancy. This method is also a supervised one and is not comparable to our scheme.

III. The Philosophy and the Architecture of SOGARG

The proposed method, SOGARG, selects, in a self-organized manner, an optimal subset of rules from all possible rules for a fuzzy controller, which can bring the system to its set point within a short time keeping in view that the controller will operate over the entire input space. SOGARG is *totally unsupervised* and *no* input-output data is required. The self-organizing process attempts to achieve the following: 1) to reduce the number of rules; 2) to reduce the average time to reach the set point (settling time); 3) to ensure that the controller operates over the entire input space; and 4) to eliminate the necessity of training data or expert's knowledge.

Let us divide the input space into two regions, the vicinity of the set point (VS) and the rest. Let us call the entire input space VB, i.e., $\mathrm{VS} \subset \mathrm{VB}$. Whatever is the initial condition of the controller, the system has to pass through VS to reach the set point. Therefore, it is very important to get a good set of rules for controlling the system in VS. In the proposed scheme, we first try to extract rules for VS and then for VB. A good set of rules for VS is expected to expedite appropriate rule generation for VB, because, when the initial condition of the controller is away from the set point, the system can be brought into VS with a few steps in many ways. Once it is inside VS, the system will be controlled by the rules for VS.

The first stage selects rules for VS. We set the initial conditions of the system at different positions in VS. The rule generation is done by genetic algorithms using a fitness function which is proportional to the number of initial conditions for which the system is controllable. Let us call the rules, thus selected, as VS rules (VSR). Stage 1 is not interested in the rules that are responsible for controlling when the system is outside the area VS.

In the second stage, we consider initial conditions near the boundary of input space and attempt to enhance and modify VSR taking VSR as input and again using GAs in a self-organized manner so that we can attain a controllable system for all initial conditions near the boundary. If we take a reasonable number of initial conditions near the boundary, then the trajectories followed by the controller are expected to span the entire input space, VB. Hence, the rule set, thus obtained, is expected to have an adequate subset of rules to drive the system to the set

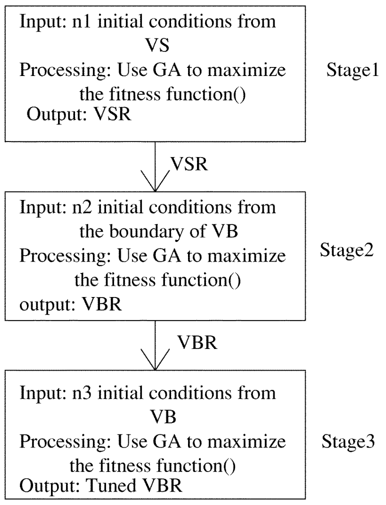

Input: n1 initial conditions from VS
Processing: Use GA to maximize the fitness function()
Output: VSR
Stage1
VSR
Input: n2 initial conditions from the boundary of VB
Processing: Use GA to maximize the fitness function()
output: VBR
Stage2
VBR
Input: n3 initial conditions from VB
Processing: Use GA to maximize the fitness function()
Output: Tuned VBR
Stage3

Fig. 1. Schematic description of SOGARG.

point from any initial conditions in VB. Let us call this rule set obtained after the second stage as VBR.

The output of the second stage, the rule set VBR, may contain some redundant and unnecessary rules. The third stage prunes and tunes the rule set VBR by screening out the unnecessary and redundant rules. It also modifies the rules, if necessary, to improve the performance of the system. The rule set pruning mechanism is designed, again using GAs, in such a manner that it reduces the number of rules and the settling time simultaneously. This is achieved using a fitness function different from the one used in the previous two stages. After the third stage, only the relevant rules remain in the rule set, which enables the controller to provide a better performance. There may be several rulebases resulting in similar performance. SOGARG, being a stochastic procedure, can lead to different rulebases in different runs with similar performance.

A schematic diagram of the proposed algorithm is shown in the Fig. 1, where the order of execution of different stages is very natural and logical. The hierarchical architecture of SOGARG is easy to interpret. SOGARG allows us to improve and refine the rulebase stage by stage.

The main advantage of SOGARG lies in its simplicity, closeness to human common sense/intuition and the fact that it does not require any data. Since, it has an incremental architecture, the system design time (computation time) is expected to be much less than other GA-based approaches. This is indeed reflected by our simulation results. Different fitness functions and mutation operations that are used in different stages of SOGARG are consistent with the goal of the corresponding stages.

We emphasize on automatic generation of a small rulebase without compromising the performance. The goodness of a fuzzy rulebased system not only depends on the number of rules in the rulebase but also on the cooperation between them. Specially, Stage 3 takes care more in this regard. It resolves the conflicts, if they exist, among the rules by deleting and/or changing some rules appropriately. If necessary, it also inserts rules to improve the performance. Moreover, the lower the number of rules, the more descriptive/interpretable the system is.

IV. Implementation of SOGARG

A. Chromosome Representation

A chromosome represents a candidate solution of the problem, i.e., a rule set for the fuzzy logic controller. The number of alleles (containing integer values) in a chromosome is equal to the number of distinguished antecedent clauses in the rules. Suppose, there are two antecedent variables x_1 and x_2 and one consequent variable, say y. Let the number of term sets corresponding to x_1, x_2, and y be m, n, and l, respectively. Then, there are $m \times n$ alleles in a chromosome, one for every possible combination of input fuzzy sets associated with the input variables x_1 and x_2. Each of these $m \times n$ antecedent clauses is represented by a unique position in every chromosome. For each antecedent clause there are l possible consequents corresponding to l output fuzzy sets, which make the total number of possible fuzzy rules $m \times n \times l$. The allele value at each location in a chromosome contains either the label of an output linguistic value to be used for a given rule or zero. In other words, if a_i represents the allele at position i, its nonzero value gives the consequent part (i.e., the label of the corresponding fuzzy set on the output variable y) of the rule which corresponds to the ith location of the chromosome. A chromosome containing an allele value zero at the ith position (i.e., $a_i = 0$) indicates that the rule set represented by the chromosome has not selected any rule with the ith antecedent clause. Hence, a chromosome can be represented as $c = \{a_i | a_i = r; r = 0, 1, \ldots, l$ and $i = 1, 2, \ldots, m \times n\}$.

B. Initial Population

The parameters required for the initialization of population are population size and lower limit (q) and the upper limit (Q) on the number of rules to be selected initially in a chromosome. For each chromosome, a random number is generated between q and Q which gives the initial number of rules in that chromosome. Positions of these rules in the chromosome are selected randomly. The allele value for each of these rules is also selected randomly from the set $\{1, 2, \ldots, l\}$. Thus, we are considering all possible fuzzy rules to get the initial population. The allele value in other positions are set to zero. Initially, the number of rules is restricted to 40%–50% of $m \times n$.

We do not restrict the GA to derive a rule set having the number of rules only between q and Q. If we do, q and Q will be treated as constraints in the crossover and mutation operations and will make the process more complex. Instead, we allow the number of rules in the rule set to vary, such that the variable length rule set may be able to grow or shrink according to the need of problem.

The population size p is also an important parameter. A big population provides diversity of the chromosomes, but the process may take a long time to converge. On the other hand, with a very small population size all chromosomes may become identical within a few generations. Therefore, the population size is to be judiciously chosen depending on the problem at hand. In this investigation, we recommend a population size $p = 40$. However, we shall see later that changing the population size by 25% on either side does not alter the performance of the final rulebase. Stage 2 operates on the terminal population of Stage 1, while the initial population for the third stage consists of the 5 best chromosomes in the terminal population of Stage 2. SOGARG also has a provision to incorporate prior knowledge, if available, at the time of initialization of population in Stage 1. This makes the process faster.

C. Reproduction, Selection, and Crossover

Roulette wheel selection has been used to reproduce a mating pool of size 1.5 times the initial population size p. We increase the population size to 1.5 times the original size p in order to increase the diversity of the population. Let the size of the mating pool be denoted by s. The crossover and mutation operations are done on this population of size s. We select $s/2$ pairs of chromosomes randomly from the mating pool and each pair is then crossed over to generate a new pair of chromosomes. The random selection of pairs is done with replacement. So, the same string may involve in crossover operation more than once. One point crossover operation is done choosing the crossover point randomly. So, after crossover, a new pool of size s is generated for mutation operation. After mutation we select the best $p-1$ strings from the current pool and the best string of the previous population for the next cycle. We bring the best chromosome of the previous population following the Elitist strategy [20]. So, the size of the new generation is again p.

D. Mutation

We have used three mutation schemes for the three stages of the algorithm, Scheme I for Stage 1, Scheme II for Stage 2, and Scheme III for Stage 3. The mutation in each chromosome results in a modified chromosome *if and only if* the fitness is increased due to this operation, otherwise, the previous chromosome is restored. This is a kind of *induced or directed mutation* [22], [23].

Scheme I:

1) randomly selects a position of zero allele value and sets it to a randomly selected value in $\{1, 2, \ldots, l\}$;
2) randomly selects another position of nonzero allele value and changes it to a value in $\{0, 1, \ldots, l\}$.

Scheme II:

1) randomly selects a position of allele value zero and changes to a randomly selected value in $\{1, 2, \ldots, l\}$;
2) randomly selects a position of nonzero allele value and sets it to zero.

Scheme III:

1) randomly selects a position of nonzero allele value and sets it to zero.

The mutation operation in Scheme I consists of insertion of a rule [step 1)] and one deletion or modification or no change of rule [step 2)]. As the probability of deletion of a rule $(1/(l+1))$, i.e., change from a nonzero value to zero, is very low, minimization of the number of rules is not given much importance in this stage. The sole aim of this stage is to obtain a set of rules (may even be with some redundant rules) that can stabilize the system from any position near the set point. The major restriction on the number of rules for this stage is imposed at the time of the initialization of population.

In Scheme II of the mutation operation, the increase in the number of rules due to insertion of a rule [step 1)] is compensated by the deletion of a rule [step 2)]. Hence, the change in the number of rules in a chromosome occurs only during crossover. Since the nonzero allele values are randomly distributed over a chromosome, the crossover operation is not expected to change much the number of rules in a chromosome. This conforms to our experience also.

The function of the mutation operation in Stage 3, i.e., Scheme III is to remove the redundant rules from the rule set selected in Stage 2 without disturbing the controllability of the rule set represented by the chromosomes (we use the term controllability not in its strict mathematical sense but to indicate the capability of the system to reach the set point within a short time from any initial condition over its operating domain). We emphasize here that the directed mutation proposed does not disturb the evolutionary characteristic of the GA.

E. Fitness Function

Choosing an appropriate fitness function is the most important aspect of applying genetic algorithms to solve any problem. Genetic algorithms require the problem to be transformed in the form of optimizing a fitness function or an objective function. The simplicity of the objective function (fitness function) is one of the most important features of our algorithm. As mentioned earlier we want to minimize the number of rules, as well as the time required to stabilize.

We run the controller for different initial conditions (i.e., with different initial states). If the system attains the set point within a predetermined number of steps (T), we call the corresponding initial state as a "controllable state." Suppose we run the controller for N initial conditions. Define a variable cs_i such that

$$\begin{aligned} cs_i &= 1, && \text{if the } i\text{th initial condition is } a \\ & && \text{controllable state} \\ &= 0, && \text{otherwise.} \end{aligned}$$

Our fitness function for Stage 1 and Stage 2 is then defined as

$$f(R) = \sum_{i=1}^{N} cs_i$$

where R is the rule set associated with the chromosome under consideration.

The fitness increases when the controller can reach the set point for more initial positions. A higher value of the fitness function of the rule set represents its superiority. For Stage 1, the initial positions are selected from VS while for Stage 2, we consider positions near the boundary of VB.

In Stage 1, the proposed scheme does not pay any attention to the rules in the area outside the VS. The Stage 2 of the proposed algorithm modifies and/or enhances VSR to generate VBR. The third stage refines VBR with a view to achieving three things: 1) the capability of the system to attain the set point with initial conditions over the entire input domain, 2) reduction of the average settling time of the system, and 3) minimization of the number of rules in the rule set. In order to realize this, we propose the following fitness function:

$$f(R) = w_b * n_b + \frac{w_t}{t_{av}} + \frac{w_r}{|R|}$$

where n_b is the number of controllable states, t_{av} is the average settling time, $|R|$ is the number of rules in R, and w_b, w_t, and w_r are three nonnegative constants representing the relative importance of the three components of the fitness function. Usually, w_b should be much greater than w_t and w_r as controllability of the system is more important than settling time and number of rules. However, the values of the weights can be varied depending on the user's preferences or as required for a particular problem. If the designer wants to give more importance on the number of rules than on the average settling time, then w_r should be greater than w_t. In the present case, we have assigned more importance to the average settling time and, hence, w_t is greater than w_r. If there are more than one rule set which can control the system for the same number of initial positions (the same value of n_b), then the fitness function will prefer the rule set with the lowest average settling time. Similarly, if there are more than one rule set with the same value of n_b and t_{av}, then GA will bias the solution toward the rule set with the minimum number of rules.

F. Choice of Initial States for Computing Fitness

One of our criteria is that the controller must be able to operate over the entire input space. To satisfy this criterion we should use a sufficient number of initial states to compute the fitness of an individual chromosome. The more the number of the initial states, the more the degree of reliability of the extracted rule set. However, this will increase the design time of the controller. So, choice of initial states is an important factor for the proposed approach.

In the proposed method, the number of initial states to be used at different stages of the procedure depends on the domains and the number of fuzzy sets associated with the input variables. For the first stage, the initial positions should be chosen from a small domain around the set point (i.e.,VS) because our aim is to find rules, which are necessary to balance from positions near the set point. When the second stage begins, we already have a good set of rules, VSR, obtained from Stage 1, which can bring the system from VS to the set point. So, for the second stage, it is sufficient to consider the initial states from the boundary of the input space. These two stages together will give a high confidence in the reliability of the controller to operate over the entire input space.

The third and the final stage requires initial states distributed over the entire input space (VB) because it tunes and prunes the rule set resulted from the previous stage (Stage 2) with respect to the number of rules and average settling time. The number of initial states in this stage is the sum of those used in the previous stages and a few more initial states from the input space in between VS and the boundary. The increase in the number of initial states hardly changes the run time of the process because of its very small population size.

V. Performance Evaluation of SOGARG

The **inverted pendulum** and **truck-back** control systems are used to illustrate the effectiveness of SOGARG. These are two well-modeled control problems commonly used for demonstration of fuzzy control algorithms and development tools.

The inverted pendulum [54] is a pole of mass m supported through a hinge by a cart of mass M, where the pole motion is constrained to be on a vertical plane and the cart motion is constrained to be along a frictionless track in the horizontal X-direction. It is assumed that the pendulum mass is concentrated at the end of the rod and the rod is mass less.

In the simulation, the original nonlinear system of equations [54] is linearized for small pole angle θ. To simulate the fuzzy controller for the inverted pendulum system, following linear equation is used in a four-step Runge-kutta method to get the next state of the system. $\dot{\mathbf{X}} = A\mathbf{X} + Bu$, where $\mathbf{X}$ and u are, respectively, state vector and fuzzy control signal. A and B are two constant matrices. For simplicity, we have ignored the cart positioning part and constrained ourselves only to pole balancing.

The inverted pendulum is an unstable system in that it may fall over at any time unless a suitable control force is applied. Only when the cart is at rest, the pole is balanced in the vertical position, and the force u on the cart is zero, the system is stable. The objective of the control problem is to apply forces to the cart until the pole is balanced in the vertical position (i.e., $\theta = 0$ and $\dot{\theta} = 0$). Here, θ is the angular displacement, $\dot{\theta}$ is the angular velocity of the pole, and u is the force applied on the cart. θ and $\dot{\theta}$ are the input linguistic variables and u is the output linguistic variable.

The truck-back system [53] consists of a truck located somewhere on a grid (xy-plane) at a given angle ϕ to the horizontal. Suppose the grid size be 100 $\times$ 100. The objective is to take the truck from any initial position (x_i, y_i, ϕ_i) to the location of the loading dock ($x = 50$, $y = 50$) while making the truck vertical to the dock ($\phi = 90\,^\circ$). The controller provides a turning angle (output) θ that is used to move the wheels (and in turn the truck) at every time step. The equations of motion for the truck are

$$\begin{aligned} \phi\prime &= \phi + \theta \\ x\prime &= x + r(\phi\prime) \text{ and} \\ y\prime &= y + r(\phi\prime) \end{aligned}$$

where r is the fixed distance the truck backs at each time step, and $\phi\prime$, $x\prime$, and $y\prime$ are, respectively, the new truck angle, x-location, and y-location. For simplicity, it is assumed that the truck is sufficiently far from the loading dock in the y direction so that the y distance could be ignored. Therefore, x and ϕ are the input linguistic variables and θ is the output linguistic variable.

A. Computational Protocols

For both systems, each of the input and output linguistic variables has seven linguistic values or fuzzy sets: NB, NM, NS, Z, PS, PM, and PB. We used overlapped isosceles triangles as membership functions as shown in Fig. 2. All membership functions have equal base length. This is possibly the most natural and unbiased choice for the membership functions.

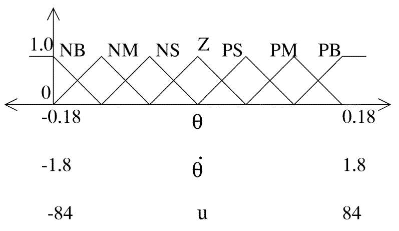


Fig. 2. Fuzzy membership functions of the linguistic values associated with input and output linguistic variables of the inverted pendulum.

TABLE I
CONSTRAINTS AND RANGES FOR (a) INVERTED PENDULUM AND (b) TRUCK BACK

Variables	Values/Ranges
l	$0.5\ m$
m	$0.1\ Kg$
M	$2\ Kg$
τ	$0.01\ sec$
θ	$-0.18\ to\ +0.18\ rad$
$\dot{\theta}$	$-1.8\ to\ +1.8\ rad/sec$
u	$-63\ to\ +63N$

(a)

Variables	Values/Ranges
r	$1.6\ m$
θ	$-30\,\text{deg}\ to\ +30\,\text{deg}$
ϕ	$-90\,\text{deg}\ to\ +270\,\text{deg}$
x	$0m\ to\ 100m$

(b)

The control rules are of the form

$$\text{if } (x_1 \text{ is } \{\text{NB}, \ldots, \text{PB}\}) \text{ and } (x_2 \text{ is } \{\text{NB}, \ldots, \text{PB}\}) \text{ then } (z \text{ is } \{\text{NB}, \ldots, \text{PB}\})$$

where x_1 and x_2 are the input linguistic variables and z is the output linguistic variable. Hence, the number of all possible fuzzy rules is $343 (= 7 \times 7 \times 7)$ and there are $49 (= 7 \times 7)$ alleles in a chromosome. The allele value is 1 for NB, 2 for NM, and so on.

The values of constants and ranges of different linguistic variables for inverted pendulum and truck-back systems are given in Table I. For the truck back, the rulebase is not a flat matrix (as in the case of inverted pendulum), because it is wrapped around to form a cylinder.

To simulate the inverted pendulum, we have used 16 initial positions for Stage 1 well around the set point ($\theta = 0, \dot{\theta} = 0$), 24 initial positions for Stage 2, selected from near the boundary of the input space, and 48 positions (16 used in Stage 1 + 24 used in Stage 2 + 8 new) for Stage 3. These initial positions are considered to evaluate the fitness of each chromosome representing a rule set. For each of these positions, the controller is simulated for $T = 100$ time steps. Thus, the maximum fitness value for Stage 1 could be 16 while that for Stage 2 could be 24.

For the truck-back system, we have taken 11 initial positions for Stage 1, 19 for Stage 2, and 42 (11 used in Stage 1 + 19 used in Stage 2 + 12 new positions) for Stage 3. For each position, the controller is run for $T = 100$ time steps. The input positions used for inverted pendulum and truck back are shown in Table II. The choice of initial positions plays a vital role

TABLE II
INITIAL POSITIONS USED IN DIFFERENT STAGES OF SOGARG FOR (a) INVERTED PENDULUM AND (b) TRUCK BACK

	Initial Positions
Stage 1	(16 positions)
	(-0.09,-0.9)(-0.09,-0.3)(-0.09,0.3)
	(-0.09,0.9)(-0.03,-0.9)(-0.03,-0.3)
	(-0.03, 0.3)(-0.03,0.9)(0.03,-0.9)
	(0.03,-0.3)(0.03,0.3)(0.03,0.9)
	(.09,-.9)(.09,-.3)(.09,.3)(.09,.9)
Stage 2	(24 positions)
	(-0.18,-1.8)(-0.18,-1.2)(-0.18,-0.6)
	(-0.18,0)(-0.18,0.6)(-0.18,1.2)
	(-0.18,1.8)(0.18,-1.8)(0.18,-1.2)
	(0.18,-0.6)(0.18,0)(0.18,0.6)
	(0.18,1.2)(0.18,1.8)(-0.12,-1.8)
	(-0.06,-1.8)(0.0,-1.8)(0.06,-1.8)
	(0.12,-1.8)(-0.12,1.8)(-0.06,1.8)
	(0.0,1.8)(0.06,1.8)(0.12,1.8)
Stage 3	(48 positions)
	(-0.12,-1.2)(-0.12,0)(-0.12,1.2)
	(0,1.2)(0.12,1.2)(0.12,0)
	(0.12,-1.2)(0,-1.2) + position
	used in Stage 1 and Stage 2

(a)

	Initial Positions
Stage 1	(11 positions)
	(40,45)(50,45)(60,45)(40,90)
	(50,90)(60,90)(40,135)(50,135)
	(60,135)(37.5,90)(62.5,90)
Stage 2	(19 positions)
	(0,-90)(25,-90)(50,-90)(75,-90)
	(100,-90)(0,0)(25,0)(50,0)
	(75,0)(100,0)(0,180)(25,180)
	(50,180)(75,180)(100,180)
	(0,90)(25,90)(75,90)(100,90)
Stage 3	(42 positions)
	(12.5,-45)(37.5,-45)(62.5,-45)
	(87.5,-45)(12.5,225)(37.5,225)
	(62.5,225)(87.5,225)(12.5,45)
	(12.5,135(87.5,45)(87.5,135)
	+ positions used in Stage 1
	and Stage 2

(b)

in achieving the controllability of the controller over the input space. Such choices are implicit human knowledge incorporated in the systems.

The constants used in the fitness function are

$$w_b = 1, w_t = 0.1, \text{ and } w_r = 0.01$$

giving more emphasis on the controllability criterion. However, the weights may be varied depending on user's choices or as required by the system considered.

We used the following parameters for the simulation of GAs:

$$\begin{aligned} \text{Population size} &= 40 \text{ for Stage 1} \\ &= 30 \text{ for Stage 2} \\ &= 5 \text{ for Stage 3.} \\ \text{Mutation rate} &= 1 \text{ per chromosome} \\ \text{Crossover rate} &= 100\% \\ \text{Number of generations} &= 15 - 25 \text{ for Stage 1 and Stage 2} \\ &= 20 \text{ for Stage 3.} \end{aligned}$$

The number of rules in the initial population is 15–20 *for inverted pendulum and* 20–25 *for truck back.*

For the inverted pendulum, the pole is considered balanced if -0.001 rad $\leq \theta \leq 0.001$ rad and -0.01 rad/s $\leq \dot{\theta} \leq 0.01$ rad/s for five successive steps within 100 time steps. And for the truck-back system, the truck is considered controlled if 49 m $\leq x \leq 51$ m and $89° \leq \phi \leq 91°$ for five successive steps within 100 time steps.

We generated $1369 (= 37 \times 37)$ samples (initial states) uniformly distributed over the product space $\theta \times \dot{\theta}$ for the inverted pendulum and $3636 (= 101 \times 36)$ samples (initial states) uniformly distributed over the product space $x \times \phi$ which are used to examine the controllability and performance of the rule sets extracted by SOGARG.

The simulation terminates if *either* the maximum number of generations (for Stage 3) is attained *or* the fitness attains a value close to the desired one.

B. Results

Keeping all computational protocols the same, we have implemented SOGARG with several initial populations to find an optimal rule set. We report here only two typical cases, Instance-1 and Instance-2, for inverted pendulum and one case Instance-3, for the truck back.

The results are shown in the form of decision tables. Each table has four subtables: (a) a typical rule set from the initial population of Stage 1; (b) the rule set obtained by Stage 1; (c) the rule set obtained by Stage 2; and (d) the final and optimal rule set produced in Stage 3.

1) Results of the Inverted Pendulum: The results are shown in Tables III and IV, each of which has four subtables. The optimal rule set obtained finally contains only about 5% of all possible fuzzy rules.

For both Instance-1 and Instance-2, none of the chromosomes in the initial population of Stage 1 could balance the system for any of the 16 initial positions. Tables III(a) and IV(a) show two typical rulebases from the initial population of Instance-1 and Instance-2, respectively, just to illustrate the evolutionary characteristic of the SOGARG process.

TABLE III
RULE SETS FOR INSTANCE-1 OF INVERTED PENDULUM. (a) A TYPICAL RULE SET FROM THE INITIAL POPULATION OF STAGE 1. (b) RULE SET OBTAINED FROM STAGE 1. (c) RULE SET OBTAINED FROM STAGE 2. (d) FINAL RULE SET OBTAINED FROM STAGE 3

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	3	0	0	0	0	2	0
NM	0	2	0	2	0	0	6
NS	0	2	7	0	0	0	6
Z	3	4	0	0	0	7	0
PS	0	0	0	0	0	0	3
PM	0	1	0	5	0	4	6
PB	0	0	0	7	0	2	6

(a)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	0	0	2	0	0	0	0
NM	6	0	4	1	0	0	7
NS	6	3	1	1	5	5	0
Z	0	0	4	4	6	6	0
PS	0	4	0	6	0	0	0
PM	6	1	0	0	0	5	7
PB	0	0	7	0	3	0	5

(b)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	2	0	2	0	2	0	0
NM	1	0	4	1	0	0	4
NS	0	0	**1**	**1**	4	0	0
Z	0	2	0	**4**	**6**	6	0
PS	3	0	5	**6**	**0**	0	0
PM	6	1	7	0	0	5	7
PB	0	0	7	0	7	0	5

(c)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	2	0	2	0	2	0	0
NM	1	0	* 0	* 0	0	0	* 0
NS	0	0	1	1	4	0	0
Z	0	2	0	4	6	6	0
PS	* 0	0	5	6	0	0	0
PM	* 0	* 0	7	0	0	* 0	7
PB	0	0	7	0	7	0	5

(d)

TABLE IV
RULE SETS FOR INSTANCE-2 OF INVERTED PENDULUM. (a) A TYPICAL RULE SET FROM THE INITIAL POPULATION OF STAGE 1. (b) RULE SET OBTAINED FROM STAGE 1. (c) RULE SET OBTAINED FROM STAGE 2. (d) FINAL RULE SET OBTAINED FROM STAGE 3

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	5	0	3	0	0	0	0
NM	5	4	4	0	0	3	1
NS	0	0	0	3	0	5	0
Z	0	0	0	0	0	0	1
PS	0	0	2	0	1	0	3
PM	3	0	0	0	5	0	5
PB	0	0	0	2	0	0	0

(a)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	1	0	6	2	0	2	0
NM	2	2	0	0	7	0	0
NS	0	0	2	3	5	0	0
Z	1	0	6	4	5	7	5
PS	0	0	0	0	5	0	0
PM	4	0	5	0	0	0	0
PB	4	0	6	1	0	0	6

(b)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	1	1	0	2	0	0	0
NM	2	1	0	0	0	0	0
NS	0	0	**2**	**3**	**5**	0	0
Z	1	0	**6**	**4**	**5**	7	0
PS	0	0	6	**0**	**5**	0	0
PM	4	2	5	5	0	0	1
PB	4	0	6	0	7	0	6

(c)

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	1	* 0	** 1	* 0	** 3	0	0
NM	* 0	* 0	0	0	** 3	0	0
NS	0	0	2	3	* 0	0	0
Z	1	0	* 0	4	5	* 0	** 5
PS	0	0	* 0	** 6	5	0	0
PM	* 0	* 0	5	* 0	0	0	* 0
PB	* 0	0	6	0	7	0	7

(d)

In order to determine the effectiveness of the proposed scheme we have examined each of the 1369 initial conditions using the rule set reported in Table III and Table IV. For Instance-1, the rule set in Table III(a) could balance only in three cases out of 1369 points. The rule set obtained after Stage 1 (Table III(b) with 24 rules) can bring the system to its set point for 1011 cases.

Note that, a proper subset of rules, VSR is capable of balancing not only the initial states in the vicinity of the set point, VS, but also a large number of initial states outside the VS. This establishes the appropriateness of Stage 1 as a first step of the proposed method to expedite the whole process.

Stage 2 enhances the rule set resulting in 1369 controllable states with 25 rules. Comparing the Tables III(b) and III(c), we find that most of the rules which are responsible for controlling the system in VS are present in both the Tables (shown in bold face), while the rules in Table III(b) which are supposed to control the system beyond VS are changed in Table III(c). This observation conforms the objective of Stage 2.

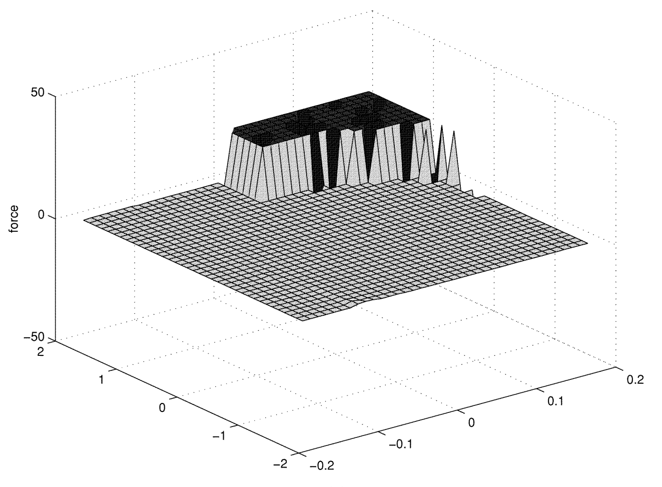
Finally, the refinement phase (Stage 3) further tunes the rule set selecting only 18 rules [Table III(d)], which enables the system to reach the set point for all 1369 initial conditions. An inspection of Table III(c) and III(d) reveals that Stage 3 deletes deletesrules. The deleted rules are indicated by asterisks $(^*)$ in Table III(d).

Table IV reveals that the case is similar for Instance-2. However, Table IV(c), i.e., the rule set produced by Stage 2, unlike the previous case of Instance-1, can bring the system to its set point for 1341 initial positions out of 1369. Hence, in Stage 3, five new rules are added [indicated by double asterisk $(^{**})$] besides the deletion [indicated by asterisk $(^*)$] of 13 rules to produce the rule set in Table IV(d) that can balance for all 1369 points. This demonstrates that SOGARG can evolve to a small but good set of rules as the process goes through Stage 1, Stage 2, and Stage 3.

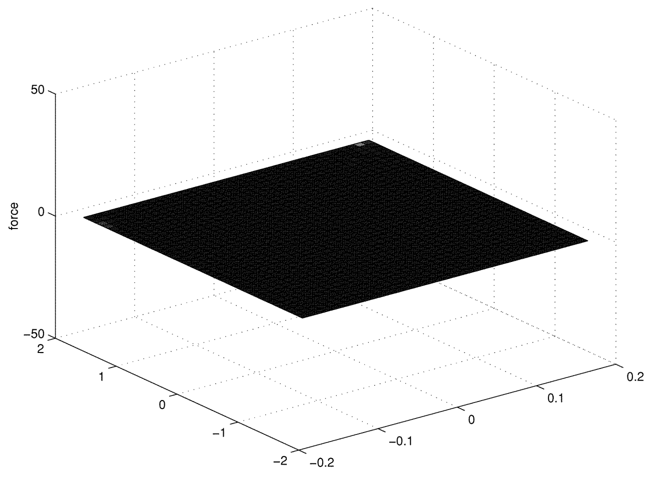
In order to further ascertain the quality of the rules extracted by SOGARG, we have plotted the forces suggested by the controller after a fixed number of time steps for each of the 1369 initial positions for the rule sets obtained by different stages of Instance-1. We call such surfaces as *force surfaces*. Fig. 3(a) and (b) depicts the *force surfaces* for the rule sets (Tables III(b) and III(c), respectively) produced by Stage 1 and Stage 2 after 50 time steps. Figs. 4(a), (b), and 5 are the same for the final rule set [Table III(d)] after 10, 20, and 30 time steps, respectively.

From these figures, it is easy to make the following observations.

1) The rule set obtained from Stage 1 balances the pendulum for the initial positions in the vicinity of the set point, VS, as well as for some positions outside the VS in 50 time steps [shown in Fig. 3(a)].
2) The rule sets obtained from Stage 2 and Stage 3 drive the pendulum to the equilibrium from all the initial conditions spanned over the entire input space in 50 and 30 time steps, respectively (Figs. 3(b) and 5). Although both rule sets in Tables II(c) and II(d) balance the pendulum from all initial conditions, the number of rules in the tuned rule set obtained by Stage 3 is less than that obtained by Stage 2. Thus, Stage 3 removes some redundant rules from the rule set obtained by Stage 2 without degrading the performance of the controller.

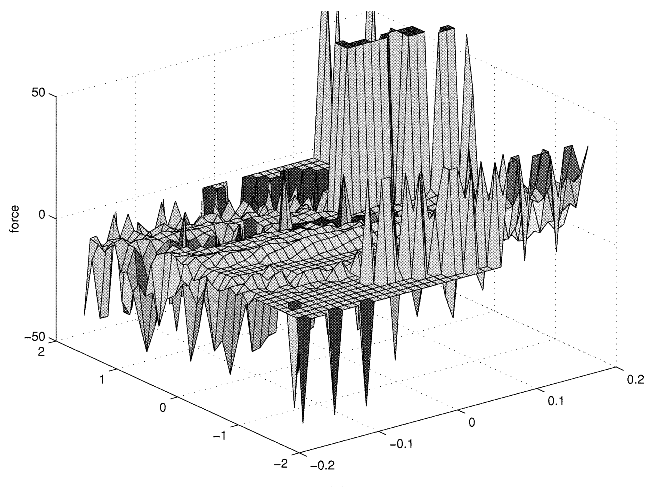


(a)

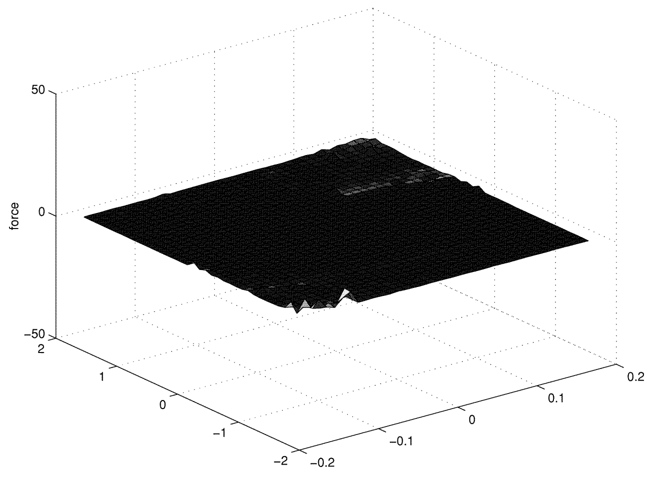


(b)

Fig. 3. Force surfaces after 50 time steps for the rule sets obtained from (a) Stage 1 and (b) Stage 2.



(a)



(b)

Fig. 4. Force surface for the rule set obtained from Stage 3 after (a) 10 time steps (b) 20 time steps.

3) The gradual improvement of the *smoothness* of *force-surfaces* shown in Figs. 4(a), (b), and 5 clearly indicates that with the passage of time the controller moves the pendulum gradually toward the set point and it attains the same only within 30 time steps for all 1369 initial conditions. Fig. 4(a) shows that for most of the initial conditions, after 10 steps, the rulebase in Fig. 3(d), could not bring the system near the set point, while Fig. 4(b) indicates that after 20 time steps the same rulebase can bring the pendulum near the equilibrium for almost all initial conditions. Finally, Fig. 5 suggests that for all initial conditions, 30 time steps are sufficient for the rulebase to bring the system to the set point.

Table V summarizes the results. It shows that for Instance-1, the rule set obtained after Stage 2 could balance all the 1369 initial positions, whereas, for Instance-2, it can balance for 1341 positions. However, the final rule sets obtained after Stage 3 can balance the pendulum for all cases. This is true for both Instance-1 and Instance-2. Moreover, the average settling time required with the final rule set is less than that required by the rule set obtained after Stage 2. The number of rules in the final rule set is also less than that obtained after Stage 2. This shows the importance of the fitness function used in Stage 3.

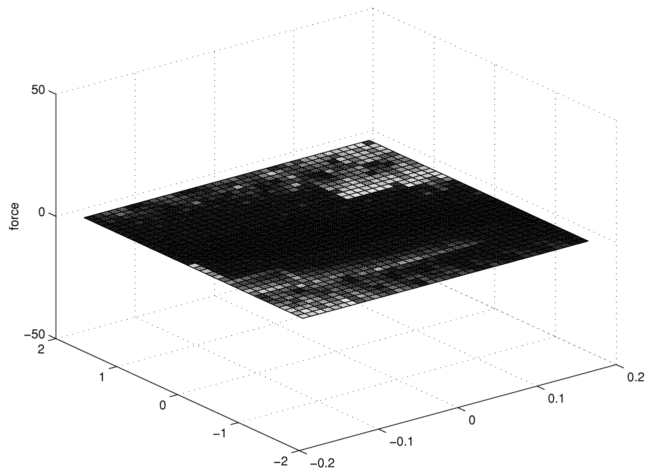


Fig. 5. Force surface after 30 time steps with the rule set obtained from Stage 3.

Study of robustness: Now we empirically study the robustness of the controller designed by SOGARG. We applied 25% perturbation in the system parameter values l, M, and m (rod length, cart mass, and pole mass) of the inverted pendulum. We compare the response curves of the system without and with deviation from the original value of l, M and m in Figs. 6(a),

TABLE V
PERFORMANCES OF DIFFERENT STAGES OF INSTANCE-1 AND INSTANCE-2 FOR INVERTED PENDULUM

		Initial	Stage 1	Stage 2	Stage 3
1	No. of rules	19	24	25	18
	Avg. no. of time steps	* N.A.	* N.A.	34.6	31.97
	Avg. ITAE	* N.A.	* N.A.	0.164	.1036
	No. of balanced positions among 1369	3	1011	1369	1369
	Failure (%)	99.78	26.15	0	0
2	No. of rules	17	23	24	16
	Avg. no. of time steps	* N.A.	* N.A.	* N.A.	41.81
	Avg ITAE	* N.A.	* N.A.	* N.A.	.1311
	No. of balanced positions among 1369	1	873	1341	1369
	Failure (%)	99.92	36.23	2.04	0

* N.A. means not applicable

(b), and (7a), respectively. Fig. 6(a) and (b) reveal that our controller is quite robust with respect to significant changes in l, M and m. Fig. 7(a) shows that the system can even smooth out the effect of 50% perturbation in the pole mass m. We applied 25% and 50% gust loading when the system is being stabled and near to the set point ($\theta = -0.0001$, $\dot{\theta} = 0.002$) at time step 50. The corresponding response curves are shown in Fig. 7(b). To see the effect of measurement noise we applied Gaussian disturbance at every step of the process. The bandwidth $(= 3 \times \sigma)$ of the Gaussian distribution was taken as 5% and 15% of the total domain of the respective input variables. The corresponding response curves are shown in Fig. 8(a) and (b). All the curves are shown only for initial position $\theta = 0.18$ and $\dot{\theta} = 1.8$. The behavior is similar for other initial conditions. Figs. 6–8 clearly reveal that the controller for the inverted pendulum problem is quite robust to the neglected dynamics and noise of the system.

2) Results of the Truck Back: A typical result of the truck-back system is shown in Table VI in the form of a decision table, having four subtables. The optimal rule set obtained contains only 20 rules, i.e. about 6% of all possible fuzzy rules.

Table VI(a) shows a typical rule set from the initial population of Instance-3 while Table VI(b)–(d) are the rulebases, respectively, obtained after the three stages of SOGARG. Out of 22 rules in Table VI(b) 6 rules are deleted, 13 are retained and 3 rules are modified in Table VI(c). Some (here, ten) new rules, as expected, are also added in Table VI(c). Table VI(d) retains 19 of the 26 rules, deletes 6 rules [indicated by asterisks (*)] and modifies just 1 rule [indicated by bullet (●)].

Table VII shows the performance of rule sets obtained from different stages. It shows that the rule sets obtained after Stage 2

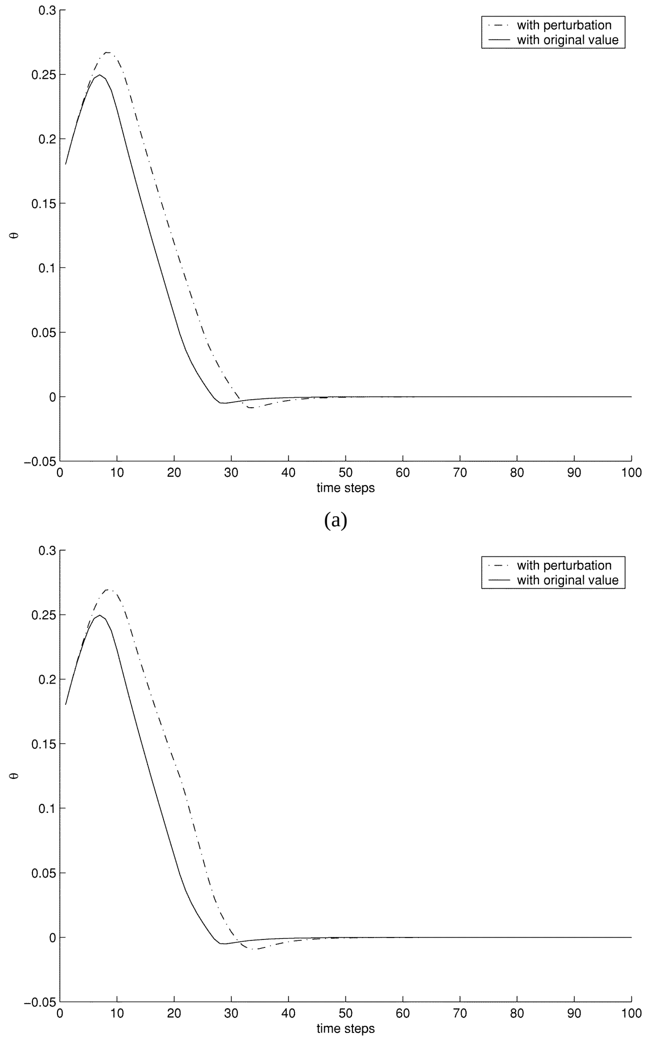


Fig. 6. Comparison of system response curves when (a) rod length l is changed by 25% from the original value and (b) the cart mass M is changed by 25% from the original value.

and Stage 3 could balance all 3636 initial positions of the truck back. The average settling time required with the final rule set is less than that required with the rule set obtained after Stage 2. Note that, the number of rules in the final rule set is less than that obtained after Stage 2.

Study of robustness: We also studied the robustness of the controller for the truck-back system. Fig. 9(a) shows the response curves of the truck-back controller for 25% change in r from the original value.

Figs.9(b) and 10(a) reveal that the system can easily smooth out the effect of 25% gust loading applied when the system is being stabled at $x = 49.9$ and $\phi = 87.72$ at time step 50. The controller can also stand the changes in θ (output) for gust loading as shown in Fig. 10(b), but Fig. 11(a) and (b) shows that the truck-back system *itself* is not so robust to measurement noise even for just 1.25%. We say this, because around the set point the oscillations are almost the same as shown in Fig. 11(a) and (b) for the rule sets obtained from SOGARG and the standard one, from which the input-output data were generated. The initial position considered was $x = 15$ m and $\phi = -30°$.

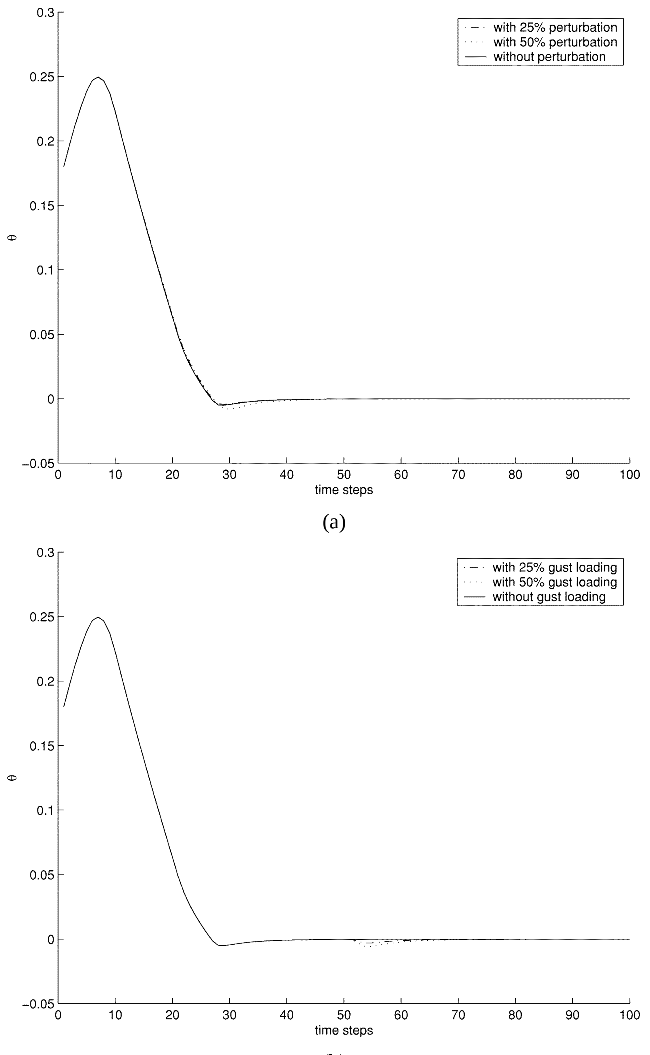


Fig. 7. Comparison of system response curves when (a) the pole mass m is changed by 25% and 50% from the original value and (b) gust loading is 25% and 50% of the total domain.

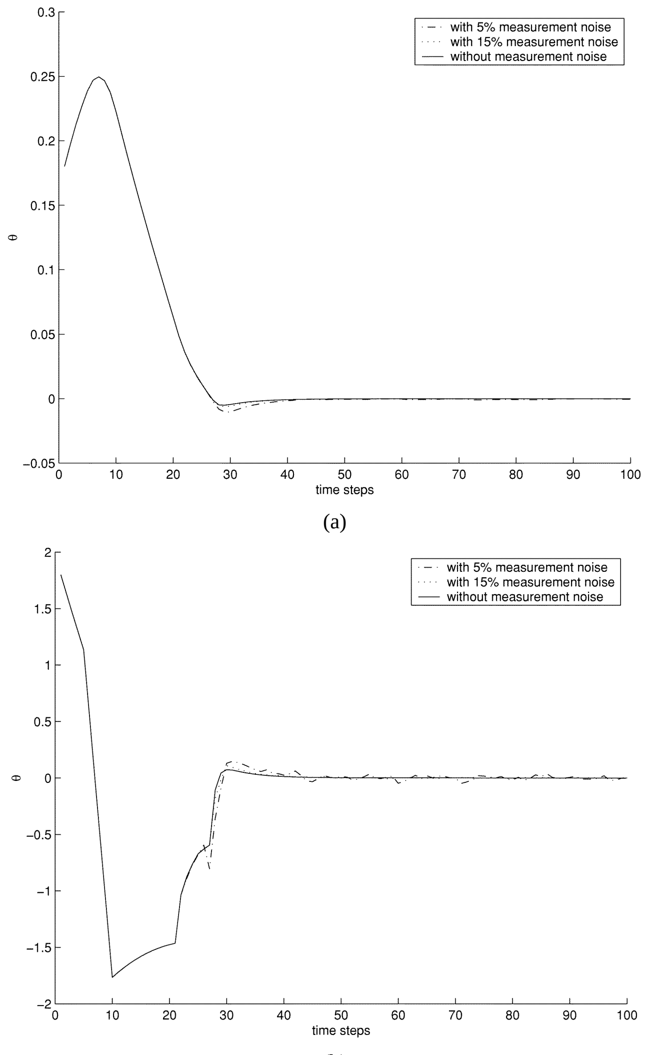


Fig. 8. Comparisons of response curves when bandwidths of measurement noise are 5% and 15% of the total domain of (a) θ and (b) $\dot{\theta}$.

The behavior for other initial conditions is similar. The response curves in Figs. 9–11 for the rule set in Table VI(d) demonstrate the robustness of the controller to neglected dynamics and noise of the system.

C. Sensitivity of the System on the Parameters of GA

Let us now see how sensitive is the proposed system with respect to changes in GA parameters: population size, mutation, and crossover rates. We consider these for the inverted pendulum system.

Population Size: If we change the population size by $\pm 25\%$ in Stage 1 and Stage 2, there is no noticeable change in the final performance. However, the number of generations required (to obtain the maximum fitness) in Stage 1 and Stage 2 vary accordingly. This is shown in the Table VIII. We keep the population size of Stage 3 fixed, because it is comparatively very small, which is equal to five. If the population size is increased, the number of generations required in Stage 1 and Stage 2 do not exceed 20 for any run, and when the population size is decreased, seven out of ten runs require more than 20 generations. In some cases, the number of rules increased in Stage 1 and Stage 2, but were compensated in Stage 3. The reason behind this is that Stage 1 and Stage 2 give importance only on the controllability and not on the number of rules in the rule set. On the other hand, Stage 3 tries to reduce the number of rules maintaining the controllability of the system, because the fitness of Stage 3 depends also on the number of rules.

Crossover: We change the crossover rate from 100% to 90% and 80% and the effect on the final results is shown in Table IX. In this case, there is also not much change in the performance of the final rulebase. As expected, a reduction in crossover rate marginally increases the required number of generations.

Mutation: Our original mutation rate was one per chromosome. We change this mutation rate by $\pm 20\%$, i.e., from 1 to 0.8 and 1.2 per chromosome. The effect of the variation of mutation rates on the final results is shown in Table X. Here also, practically there is no noticeable change in the performance of the final rulebase. There is only a minor change in the required number of generations in Stage 1 and Stage 2.

Stopping Criteria: In Stage 3 of our proposed method, the simulation terminates if the maximum number of generations

TABLE VI
RULE SETS OF INSTANCE-3 OF TRUCK BACK.
(a) A TYPICAL RULE SET FROM THE INITIAL POPULATION OF STAGE 1.
(b) RULE SET OBTAINED FROM STAGE 1. (c) RULE SET OBTAINED FROM STAGE 2. (d) FINAL RULE SET OBTAINED FROM STAGE 3

θ / x	NB	NM	NS	Z	PS	PM	PB
NB	0	1	6	0	1	0	5
NM	2	6	0	0	0	6	0
NS	0	3	1	6	0	0	0
Z	4	0	0	0	3	0	5
PS	0	0	5	7	0	0	0
PM	2	0	0	0	0	2	6
PB	1	4	6	0	0	1	0

(a)

θ / x	NB	NM	NS	Z	PS	PM	PB
NB	0	0	0	0	1	1	5
NM	2	6	3	0	0	6	0
NS	0	7	0	2	1	0	0
Z	4	4	0	0	3	5	5
PS	0	0	5	7	0	0	0
PM	2	0	0	6	0	2	6
PB	0	0	0	0	0	4	0

(b)

θ / x	NB	NM	NS	Z	PS	PM	PB
NB	0	0	0	2	1	1	5
NM	6	7	3	0	0	0	0
NS	0	7	0	1	0	1	0
Z	0	0	5	4	3	5	5
PS	5	6	5	7	0	0	0
PM	2	6	0	0	0	2	0
PB	0	6	6	6	0	4	0

(c)

θ / x	NB	NM	NS	Z	PS	PM	PB
NB	0	0	0	2	1	1	* 0
NM	6	* 0	3	0	0	0	0
NS	0	7	0	1	0	1	0
Z	0	0	5	4	3	* 0	5
PS	5	6	* 0	7	0	0	0
PM	• 6	* 0	0	0	0	2	0
PB	0	6	6	6	0	* 0	0

(d)

specified is achieved. In Stage 1 and Stage 2, the simulation stops when the fitness value becomes equal to the number of

TABLE VII
PERFORMANCES OF RULE SETS OBTAINED FROM DIFFERENT STAGES FOR INSTANCE-3 OF TRUCK BACK

	Initial	Stage 1	Stage 2	Stage 3
No. of rules	22	21	26	20
Avg. no. of time steps	* N.A.	* N.A.	49.94	40.51
Avg. ITAE	* N.A.	* N.A.	78.5	61.8
No. of balanced positions among 1369	34	2571	3636	3636
Failure (%)	99.06	29.29	0	0

* N.A. means not applicable

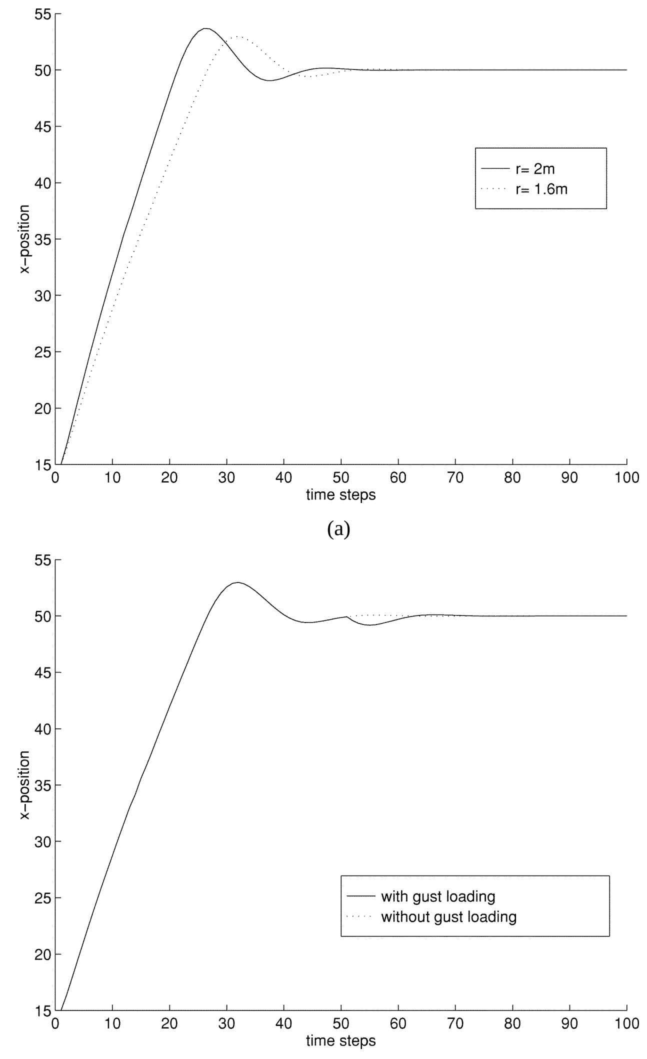
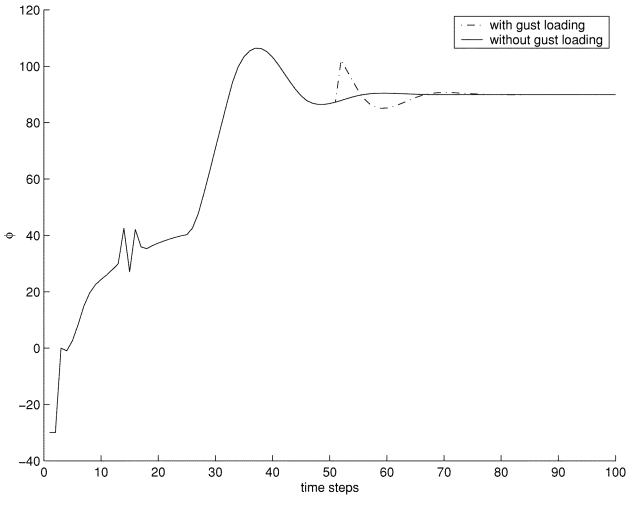
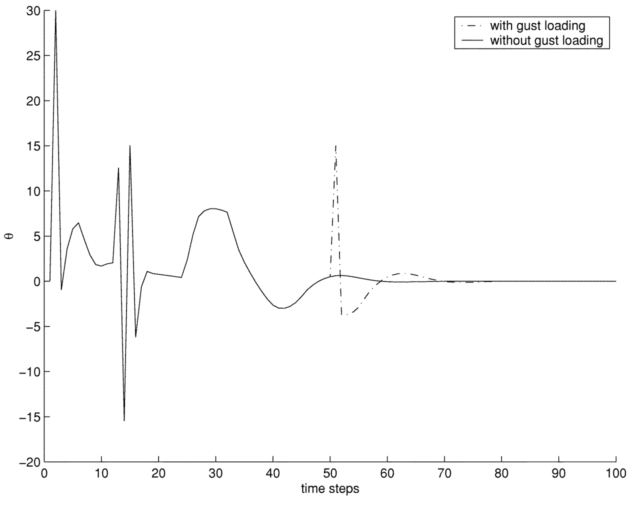


Fig. 9. Comparison of system response curves for x when (a) r is changed by 25% from the original value and (b) 25% gust loading is applied.

initial positions (specified) or the number of generations reaches a specified maximum value. For all three stages, we specify a large value (100) for the maximum number of generations. In all our simulations it is never attained. Changing the value even by

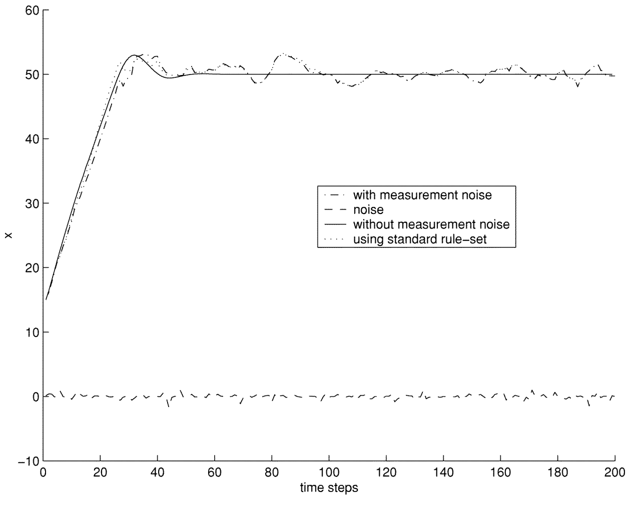


(a)

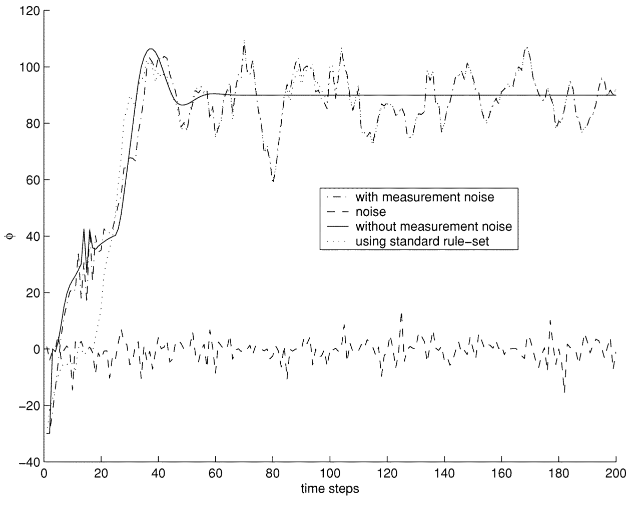


(b)

Fig. 10. Comparison of system response curves for (a) ϕ and (b) θ when 25% gust loading is applied.



(a)



(b)

Fig. 11. Comparison of system response curves of (a) x and (b) ϕ when 1.25% measurement noise is applied both in x and ϕ simultaneously.

30% does not change the performance as the required number of generations is always much less than the maximum value. However, if we reduce the number of generations to a very low value, the performance is likely to degrade.

VI. Comparisons With Other Methods

There have been several attempts to extract human interpretable rules for the cart-pole system [34], [41]. In Section II, we have discussed the methods by Lim *et al.* [34] and by Bobbin and Yao [41]. Here, we compare the performance of SOGARG with that of Lim *et al.* [34] because Lim *et al.*'s method is more similar in spirit with that of SOGARG. In addition, we also compare our method with the method of Chan *et al.* [42]. For a fair comparison, while realizing the methods in [34] and in [42], we use the same 48 initial positions as used in Stage 3 of SOGARG [Table II(a)] for the inverted pendulum. Similarly, the same 42 initial positions shown in Table II(b) are used for the truck-back system.

A. Comparison With the Method of Lim et al. [34]

Given fixed domains and symmetric triangular membership functions for each input and output variable, Lim *et al.* [34] described a learning process based on GA to derive R fuzzy control rules. We have already described their algorithm briefly in Section II. Here are a few more relevant details.

1) The number of rules in the rulebase is fixed. It can be viewed as a constraint on the learning process and requires at least some knowledge of the underlying problem complexity, which may not be known *a priori*. Modifications of genetic operations such as crossover and mutation, and other operations such as rule creation and rule deletion were required in their algorithm to keep the number of rules in the chromosome fixed. These modifications may not help the learning process or improve the quality (with respect to controllability) of the rule set.
2) The fitness function of Lim *et al.* [34] is

$$f = \sum_{i=1}^{d} \frac{bT_{\theta i} + (1-b)T_{ei}}{dT}.$$

TABLE VIII
EFFECTS OF VARIATIONS IN THE POPULATION SIZE ON THE SYSTEM PERFORMANCE

	population size in Stage 1 and 2		
	changed by 25%		original
	increase	decrease	
No. of rules	16.2	18.3	16.6
Avg. no. of time steps	37.7	40.45	35.44
Avg. ITAE	0.084	0.1025	0.1019
Controllability	100%	100%	100%
No. of generations required in Stage 1 and 2	13-19	15-27	15-25

TABLE IX
EFFECTS OF VARIATIONS IN THE CROSSOVER RATE ON THE SYSTEM PERFORMANCE

	crossover rate		
	changed to		original
	90%	80%	100%
No. of rules	17.22	19.3	16.6
Avg. no. of time steps	40.7	40.0	35.44
Avg. ITAE	0.1	0.99	0.1019
Controllability (%)	100	100	100
No. of generations required in Stage 1 and Stage 2	15-28	17-30	15-25

For a test with the ith initial condition, $T_{\theta i}$ denotes the number of time steps the pole remains within 1° from the vertical position and T_{ei} denotes the number of time steps elapsed before the pole falls. They used $b = 0.6$ in their simulation. We use the same value. For each of these tests, the cart-pole system is simulated until the pole falls or the prespecified value of $T (= 200)$ time steps is reached.

We implemented Lim *et al.*'s [34] method on our inverted pendulum. Table XI(a) shows a result of their method having 20 rules, which can only balance 1321 states among 1369. We had to relax the balancing condition mentioned in Section V-A for Lim *et al.*'s [34] method, because their rule set is not able to drive the cart-pole system to that precision. We considered the pole balanced, if $\theta \leq \pm 0.005$ for at least five time steps. This may be due to the importance of T_{ei} in the fitness function. Table XI(b) shows a comparison of our result with theirs, averaged over ten runs. We repeated our experiment ten times and the average number of rules found is 16.6 with an average ITAE of 0.1019, whereas the average ITAE of Lim *et al.*'s [34] method is about 14.52.

We emphasize the fact that Lim *et al.*'s [34] algorithm is only applicable to the cart-pole system. It cannot be implemented for the truck-back system for comparison purpose.

TABLE X
EFFECTS OF VARIATIONS IN THE MUTATION RATE ON THE SYSTEM PERFORMANCE

	mutation rate		
	changed to		original
	0.8	1.2	1
No. of rules	17.2	15.2	16.6
Avg. no. of time steps	42.5	40.9	35.44
Avg. ITAE	0.13	0.097	0.1019
Controllability (%)	100	100	100
No. of generations required in Stage 1 and Stage 2	20-30	13-22	15-25

TABLE XI
(a) A RULE SET CONTAINING 20 RULES OBTAINED BY THE METHOD OF Lim *et al.* [34]. (b) PERFORMANCE COMPARISON OF SOGARG AND Lim *et al.*'s METHOD [34], AVERAGED OVER TEN RUNS

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	3	0	1	0	5	2	0
NM	0	0	0	2	6	6	0
NS	0	2	0	0	3	0	0
Z	1	6	1	0	6	0	6
PS	0	0	0	7	0	5	0
PM	0	0	6	5	0	6	0
PB	0	0	0	0	0	0	6

(a)

	SOGARG	Lim et al.[34]
No. of rules	16.6	20 (fixed)
Avg. no. of time steps (considering only balanced positions)	35.44	40*
Avg. ITAE	0.1019	14.52
No. of balanced positions among 1369	1369	1311*
Failure (%)	0	3.5*

* For Lim et al.'s [34] method, the balanced condition is taken as $\theta \leq \pm 0.005$ for at least 5 successive time steps.

(b)

B. Comparison With the Method of Chan et al. [42]

We briefly discussed Chan *et al.*'s [42] method in Section II. The input and output fuzzy sets are modeled by symmetric and triangular membership functions. A chromosome looked for an exhaustive rule set, i.e., $m \times n$ $(= 49)$ rules in the rule set.

TABLE XII
(a) A RULE SET GENERATED BY OFLC AFTER 50 GENERATIONS, HAVING AVERAGE ITAE $= 0.0851$ AND AVERAGE TIME STEPS $= 45$. (b) PERFORMANCE COMPARISONS OF SOGARG WITH OFLC, AVERAGED OVER TEN RUNS FOR INVERTED PENDULUM

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	1	1	1	1	2	1	6
NM	1	2	2	1	1	6	6
NS	1	3	1	2	6	2	3
Z	2	1	4	4	4	7	6
PS	5	6	2	6	7	5	7
PM	2	2	7	7	6	6	7
PB	2	7	6	7	7	7	7

(a)

	SOGARG		OFLC
No. of rules	16.6	20.8	49 (fixed)
Avg. no. of time steps	35.44	25.52	28
Avg. ITAE	0.1019	0.0535	0.053
No. of balanced positions among 1369	1369	1369	1369
Failure (%)	0	0	0

(b)

TABLE XIII
(a) A RULE SET CONTAINING 20 RULES GENERATED BY SOGARG, HAVING AVERAGE ITAE $= 0.0534$ AND AVERAGE TIME STEPS $= 26.8$ FOR INVERTED PENDULUM. (b) A RULE SET GENERATED BY OFLC AFTER 100 GENERATIONS, HAVING AVERAGE ITAE $= 85.7$ AND AVERAGE TIME STEPS $= 49.02$ FOR TRUCK BACK

$\dot{\theta}$ / θ	NB	NM	NS	Z	PS	PM	PB
NB	1	1	0	2	2	0	0
NM	1	0	0	0	0	0	0
NS	0	0	1	2	4	0	0
Z	0	1	3	4	5	5	0
PS	0	0	4	6	0	7	0
PM	0	0	0	0	0	0	7
PB	0	0	7	0	7	0	7

(a)

ϕ / x	NB	NM	NS	Z	PS	PM	PB
NB	6	4	3	3	2	1	1
NM	7	3	4	4	1	2	2
NS	2	7	1	2	2	2	2
Z	6	5	5	4	3	3	2
PS	6	6	6	6	7	1	6
PM	6	6	7	4	4	5	1
PB	7	7	6	5	5	4	2

(b)

The system is assumed to be symmetric and hence the rulebase is taken as symmetric. So, the second half of a chromosome is taken as the mirror image of the first half of that chromosome with respect to the line of symmetry located at $(\text{length of chromosome} + 1)/2$. The mirror image of allele value NS (3) is PS (5). The index, d, of the fuzzy set representing the mirror image of a fuzzy set with the index k is governed by the relation, $d = (\text{number of fuzzy subsets } (l) + 1) - k$.

Chan *et al.* [42] initialized the population using a random generator and/or used an expert provided rule set. First, a pair of parents is selected, then *either* the crossover *or* the mutation operation is applied to produce two children. So, if the crossover rate is $0 < \alpha < 1$, then the mutation rate is $(1-\alpha)$. The process is repeated to produce a new population of children of the same size as that of the original population. To avoid the effect of super individuals, the maximum fitness of a generation is added to the fitness value of each chromosome for linearization of fitness values.

Both crossover and mutation are done on the first half of chromosomes. The first half of a child is then mirrored to the second half. Chan *et al.* [42] used "one step change mutation scheme," which changes a randomly selected allele value to either its next or previous value with equal probability. When the allele value at the site is one, it will not be decreased, also when it is l, it will not be increased further. Although the "one step change mutation scheme" of Chan *et al.* [42] can improve local search, it can greatly influence the maintenance of population diversity.

In the OFLC, steady state without duplicates (SSWOD) [21] is used to select the best fit individuals between the parents and children for the new generation. If the population size is N, then the reproduction module will produce N children using mutation and crossover. Next, from the existing population and their N children, i.e., from the total $2N$ individuals, the best fit population of N individuals is obtained for the next generation.

If none of the new N offspring is selected for the next generation, then the number of mutations to be done on a chromosome is increased by one for the next generation, and the whole population is replaced by another randomly generated population, except for the best string, which is kept in the new population.

Inverted Pendulum: A rule set generated by OFLC after 50 generations is shown in Table XII(a). Table XII(b) compares the performance of SOGARG and OFLC of Chan *et al.* [42], averaged over ten runs. Note that OFLC used an exhaustive rule set, i.e., $m \times n \ (= 49)$ rules in every rule set. When we take $w_r = 0.01$ and 15–20 rules in the initial population as specified in the Section V-A, the average number of rules in the rule sets produced by SOGARG is only 16.6. In this case though, the number of rules is about 1/3 of that produced by OFLC, average ITAE and number of time steps are comparable to those produced by OFLC. Reducing the weight w_r to 0.007 and increasing the number of rules in the initial population to 20–25, the average number of rules in the rule sets produced by SOGARG is increased to 20.8, which is still much less than half

TABLE XIV
PERFORMANCE COMPARISON OF SOGARG WITH OFLC, AVERAGED OVER TEN RUNS FOR TRUCK BACK

	SOGARG	OFLC
No. of rules	23	49 (fixed)
Avg. no. of time steps (considering only balanced positions)	40	46.7
Avg. ITAE	71.42	103
No. of balanced positions among 3636	3636	3636
Failure (%)	0	0

of that produced by OFLC. In this case, the average ITAE and average number of time steps are comparable to those of OFLC. A typical rule set generated by SOGARG with the changed values is shown in Table XIII(a).

Truck Back: An exhaustive rule set generated by OFLC after 100 generations is shown in Table XIII(b). Table XIV shows the performance comparison of rule sets obtained by SOGARG and OFLC of Chan *et al.* [42], averaged over ten runs. It reveals that SOGARG outperforms OFLC in this case.

VII. CONCLUSION

We proposed a new method for extraction of a near optimal rule set for a fuzzy logic controller. It is a genetic algorithm-based self-organized scheme. This method consists of three stages. The first stage attempts to extract a rule set with a view to enabling the system to control in the vicinity of the set point. In the second stage, this rule set is enhanced and modified to account for the entire input space. Finally, the last stage fine tunes the rule set through modification of existing rules and/or deletion of redundant rules and/or addition of new rules. We have used different objective functions and different mutation schemes for different stages which are consistent with the objectives of different stages. The effectiveness of the proposed scheme is demonstrated using two examples: inverted pendulum and truck back. We compared our method with the methods of Lim *et al.* [34] and Chan *et al.* [42]. Our method performed better according to our results. We have also empirically demonstrated that our method is quite robust against neglected dynamics and noise (gust loading and measurement noise). Our system is also robust against significant changes in the parameters of GAs. We emphasize that, with a proper choice of membership functions, the number of rules may be reduced further.

Investigation needs be done in order to formulate a guideline for selection of the three weights w_b, w_t, and w_r used in the fitness function of Stage 3. The choice of these weight values has a significant effect on the final solution, especially on rule selection. So, one simple method may be to employ various values of weights and the final solution could be selected from a set of solutions by the user depending on his/her preference. For this purpose, coevolutionary GAs using a second population, which evolves the weights [52] may be used. Further work also needs to be done to test SOGARG on a wide range of different problems in addition to the two used in this paper, so that a better understanding of its strength and weakness can be obtained.

REFERENCES

[1] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic—Theory and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[2] L. A. Zadeh, "Fuzzy logic and approximate reasoning," *Syntheses*, vol. 30, pp. 407–428, 1975.
[3] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man Mach. Stud.*, vol. 7, pp. 1–13, 1974.
[4] E. H. Mamdani, T. Procyk, and N. Baaklini, "Application of fuzzy logic controller design based on linguistic protocol," in *Discrete Systems and Fuzzy Reasoning*, E. H. Mamdani and B. R. Gaines, Eds. London, U.K.: Queen Mary College, University of London, 1976, pp. 125–149.
[5] T. J. Procyk and E. H. Mamdani, "A linguistic self-organizing process controller," *Automatica*, vol. 15, pp. 15–30, 1979.
[6] C. C. Lee, "Fuzzy logic in control system: fuzzy logic controller—part I and II," *IEEE Trans. Syst. Man Cybern.*, vol. 20, pp. 404–435, Mar.-Apr. 1990.
[7] D. Driankov, H. Hellendoorn, and M. Reinfrank, *An Introduction to Fuzzy Control*. New York: Springer-Verlag, 1993.
[8] T. Yamakawa, "A fuzzy logic controller," *J. Biotechnologies*, vol. 24, pp. 1–32, 1992.
[9] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst. Man Cybern.*, vol. 15, pp. 116–132, 1985.
[10] M. Sugeno, *Industrial Applications of Fuzzy Control*. Amsterdam, The Netherlands: Elsevier, 1985.
[11] M. Sugeno and M. Nishida, "Fuzzy control of model cat," *Fuzzy Sets Syst.*, vol. 16, pp. 103–113, 1985.
[12] C. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, pp. 1320–1336, Dec. 1991.
[13] J. J. Shann and H. C. Fu, "A fuzzy neural network for rule acquiring on fuzzy control system," *Fuzzy Sets Syst.*, vol. 71, pp. 345–357, 1995.
[14] N. R. Pal and T. Pal, "On rule pruning using fuzzy neural networks," *Fuzzy Sets Syst.*, vol. 106, pp. 335–347, 1999.
[15] J. M. Benitez, A. Blanco, and I. Requena, "An empirical procedure to obtain fuzzy rules using neural networks," in *Proc. VII IFSA World Congress, 95*, Sao Paulo, Brazil, 1995, pp. 663–666.
[16] H. Ishibuchi, R. Fujioka, and H. Tanaka, "Neural networks that learn from fuzzy if-then rules," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 85–97, May 1993.
[17] K. Lee, D. Kwang, and H. L. Wang, "A fuzzy neural network model for fuzzy inference and rule tuning," *Int. J. Uncertainty, Fuzziness Knowledge-Based Syst.*, vol. 2, pp. 265–277, 1994.
[18] C. C. Li and C. J. Wu, "Generating fuzzy rules for a neural fuzzy classifier," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE '94)*, Orlando, FL, 1994, pp. 1719–1724.
[19] S. Yao, C. Wei, and Z. He, "Evolving fuzzy neural networks for extracting rules," in *Proc. 5th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE '96)*, New Orleans, LA, 1996, pp. 361–367.
[20] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[21] L. Davis, *Handbook of Genetic Algorithms*. Reinhold: Van Nostrand, 1991.
[22] H. J. Muller, *Studies in Genetics—Selected Papers*. Bloomington, IN: Indiana Univ. Press, 1962.
[23] D. Bhandari, N. R. Pal, and S. K. Pal, "Directed mutation in genetic algorithms," *Inform. Sci.*, vol. 79, pp. 251–270, 1994.
[24] K. Nakaoka, T. Furuhashi, and Y. Uchikawa, "A study on apportionment of credits of fuzzy classifier system for knowledge acquisition of large scale systems," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systems*, Piscataway, NJ, 1994, pp. 1797–1800.
[25] A. Parodi and P. Bonelli, "A new approach of fuzzy classifier systems," in *Proc. 5th Int. Conf. Genetic Algorithms*, Los Altos, CA, 1993, pp. 223–230.
[26] C. L. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 450–457.
[27] C. L. Carr, "Genetic algorithm for fuzzy logic controller," *AI Expert*, vol. 2, pp. 22–23, 1991.

[28] P. Thrift, "Fuzzy logic synthesis with genetic algorithm," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 509–513.
[29] H. Nomura, I. Hayashi, and N. Wakami, "A self tuning method of fuzzy control by genetic algorithm," in *Proc. Int. Fuzzy Systems Intell. Contr. Conf. (IFSICC '92)*, 1992, pp. 236–245.
[30] 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.
[31] D. Park, A. Kandel, and G. Langholz, "Genetic-based new fuzzy reasoning models with application to fuzzy control," *IEEE Trans. Syst. Man Cybern.*, vol. 24, pp. 39–47, Jan. 1994.
[32] F. Herrera, M. Lozano, and J. L. Verdegay, "Tuning fuzzy logic controllers by genetic algorithm," *Int. J. Approximate Reasoning*, vol. 12, pp. 299–315, 1995.
[33] 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.
[34] M. H. Lim, S. Rahardja, and B. H. Gwee, "A GA paradigm for learning fuzzy rules," *Fuzzy Sets Syst.*, vol. 82, pp. 177–186, 1996.
[35] B. Carse, T. C. Fogarty, and A. Munro, "Evolving fuzzy rule based controllers using genetic algorithms," *Fuzzy Sets Syst.*, vol. 80, pp. 273–293, 1996.
[36] L. Renhou and Z. Yi, "Fuzzy logic controller based genetic algorithms," *Fuzzy Sets Syst.*, vol. 83, pp. 1–10, 1996.
[37] H. B. Gurocak, "A genetic-algorithm-based method for tuning fuzzy logic controllers," *Fuzzy Sets Syst.*, vol. 108, pp. 39–47, 1999.
[38] C.-C. Wong and S.-M. Her, "A self-generating method for fuzzy system design," *Fuzzy Sets Syst.*, vol. 103, pp. 13–25, 1999.
[39] C.-C. Wong and C.-S. Fan, "Rule mapping fuzzy controller design," *Fuzzy Sets Syst.*, vol. 108, pp. 253–261, 1999.
[40] J. Kinzel, F. Klawonn, and R. Fruse, "Modifications of GA for designing and optimising fuzzy controller," in *IEEE Int. Conf. Computational Intelligence*, 1994, pp. 28–32.
[41] J. Bobbin and X. Yao, "Evolving rules for nonlinear control," in *New Frontier in Computational Intelligence and its Applications*, M. Mohammadian, Ed. Amsterdam, The Netherlands: IOS Press, 2000, pp. 197–202.
[42] P. T. Chan, W. F. Xie, and A. B. Rad, "Tuning of fuzzy controller for an open-loop unstable system: a genetic approach," *Fuzzy Sets Syst.*, vol. 111, pp. 137–152, 2000.
[43] H.-X. Li and H. B. Gatland, "A new methodology for designing a fuzzy logic controller," *IEEE Trans. Syst. Man Cybern.*, vol. 25, pp. 505–512, Mar. 1995.
[44] O. Cordon, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic Fuzzy Systems. Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. Singapore: World Scientific, 2001, vol. 19, Series Advances in Fuzzy Systems-Application and Theory.
[45] H. Ishibuchi, T. Murata, and I. B. Turksen, "Single-objective and two-objective genetic algorithms for selection linguistic rules for pattern classification problems," *Fuzzy Sets Syst.*, vol. 89, pp. 135–150, 1997.
[46] A. Krone, P. Krause, and T. Slawinski, "A new rule reduction method for finding interpretable and small rule bases in high dimensional search spaces," in *Proc. 9th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE'2000)*, vol. 2, San Antonio, TX, 2000, pp. 694–699.
[47] T. C. Chin and X. M. Qim, "Genetic algorithms for learning the rule base of fuzzy logic controller," *Fuzzy Sets Syst.*, vol. 97, no. 1, pp. 1–7, 1998.
[48] O. Cordon and F. Herrera, "A three-stage evolutionary process for learning descriptive and approximate fuzzy logic controller knowledge bases from examples," *Int. J. Approximate Reasoning*, vol. 17, no. 4, pp. 369–407, 1997.
[49] B. H. Roubos and M. Setnes, "Compact fuzzy models through complexity reduction and evolutionary optimization," in *Proc. 9th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE'2000)*, vol. 2, San Antonio, TX, USA, 2000, pp. 762–767.
[50] T. Pal, N. R. Pal, and S. DebRay, "A self-organized rule generation scheme for fuzzy logic controllers," in *Proc. 9th IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE'2000)*, vol. 1, San Antonio, TX, 2000, pp. 13–18.
[51] T. Pal, "Evolutionary approaches to rule extraction for fuzzy logic controllers," in *Advances in Soft Computing, (Lecture Notes Series in Artificial Intelligence)*, vol. 2275, N. R. Pal and M. Sugeno, Eds. New York: Springer-Verlag, 2002, pp. 425–432.
[52] C. A. Peña-Reyes and M. Sipper, "Fuzzy CoCo: a cooperative-coevolutionary approach to fuzzy modeling," *IEEE Trans. Fuzzy Syst.*, vol. 9, pp. 727–737, Oct. 2001.
[53] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[54] K. Ogata, *Modern Control Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

Tandra Pal (M'02) received the B.Sc. degree in physics, the B.Tech. degree in computer science from Calcutta University, Calcutta, India, and the M.E. degree in computer science from Jadavpur University, Calcutta, India, in 1986, 1991, and 1993, respectively.

Since 1994, she has been a Lecturer in Regional Engineering College, Durgapur. Currently, she is a Visiting Research Fellow in the Electronics and Communication Sciences Unit of the Indian Statistical Institute, Calcutta. Her research interests includes fuzzy systems including fuzzy control, neural networks, and genetic algorithms.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree (honors) in physics, the M.S. degree in business management, from the University of Calcutta, Calcutta, India, in 1978 and 1982, respectively, and the M.Tech. and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, India, in 1984 and 1991, respectively.

Currently, he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute, Calcutta. His research interest includes image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, evolutionary computation, and fuzzy logic controllers. He has coauthored a book titled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing* (Norwell, MA: Kluwer, 1999), coedited two volumes titled *Advances in Pattern Recognition and Digital Techniques*, ICAPRDT99 (Narosa, 1999), and *Advances in Soft Computing*, AFSS 2002 (New York: Springer-Verlag, 2002), and edited a book titled *Pattern Recognition in Soft Computing Paradigm* (Singapore: World Scientific, 2001).

Dr. Pal is an Associate Editor of the *International Journal of Fuzzy Systems*, *International Journal of Approximate Reasoning*, IEEE TRANSACTIONS ON FUZZY SYSTEMS, and IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS-B. He is an Area Editor of *Fuzzy Sets and Systems*. He is a Member of the Editorial Advisory Board of the *International Journal of Knowledge-Based Intelligent Engineering Systems*, and a Steering Committee Member of the journal, *Applied Soft Computing*, Elsevier Science. He is an Independent Theme Chair of the World Federation of Soft Computing and a Governing Board Member of the Asia Pacific Neural Net Assembly. He was the Program Chair of the 4th International Conference on Advances in Pattern Recognition and Digital Techniques, December 1999, Calcutta, India, and the General Chair of the 2002 AFSS International Conference on Fuzzy Systems, Calcutta, 2002.