Learning Fuzzy Rules for Controllers with Genetic Algorithms

T. Pal,[1,*] N. R. Pal,[1,†] M. Pal[2,‡]
[1]*Electronics and Communication Sciences Unit and* [2]*Economic Research Unit, Indian Statistical Institute, 203 B.T. Road, Calcutta 700108, India*

A genetic algorithm (GA)-based scheme for learning fuzzy rules for controllers, called an optimized fuzzy logic controller (OFLC) was proposed by Chan, Xie and Rad (2000). In this article we first analyze their OFLC and discuss some of its limitations. We also propose some modifications on an OFLC to eliminate those limitations. Then, for systems with symmetrical rule base we propose a new method to reduce the number of rules, which reduces the search space as well as the design time. We define a fitness function that reduces the number of rules maintaining the performance of the rule set. The trade-off between the number of rules and the performance can be decided by changing the parameters of our fitness function. We theoretically analyzed the properties of "one-step change mutation" and compared that with our mutation scheme. The proposed scheme, for the inverted pendulum problem can find rule sets containing $<5\%$ of all possible fuzzy rules, having good integral time absolute error (ITAE) and it takes only a few steps to balance the system over the entire input space. To show the superiority of our scheme, we compare it with other methods. © 2003 Wiley Periodicals, Inc.

1. 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.[1,2] One of the most important applications of fuzzy logic is in controller design.[3–8] Fuzzy logic controllers (FLCs) convert the linguistic control strategy into an automatic control strategy. Experience shows that the FLC yields results sometimes superior to those obtained by conventional control algorithms for its flexibility in representing the subjective nature of human decision making. In particular, the FLC appears very useful when the process is too complex to analyze by conventional quantitative technique or when the available sources of information are interpreted qualitatively, inexactly,

[*]On study leave from R.E. College, Durgapur, India. e-mail: tandra_v@isical.ac.in.
[†]Author to whom all correspondence should be addressed: e-mail: nikhil@isical.ac.in.
[‡]e-mail: mano@isical.ac.in.

or imprecisely. Thus, the FLC falls in between the conventional precise mathematical control and expert-like decision making.

FLCs model the human decision-making process with a collection of fuzzy *if-then* rules.[6] Successful design of a fuzzy control system depends on several factors such as choice of the rule set, membership functions, inferencing scheme, and the defuzzification strategy. Of these factors, selection of an appropriate rule set is the most important one. Sometimes, fuzzy control 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 essential step toward designing any successful FLC.

There have been several attempts both under supervised and self-organized paradigms for obtaining a good rule base. Some of these methods use neural networks[9–16] and others use genetic algorithms (GAs).[17–33] The rule base 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 an 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 an FLC with q inputs and one output, the total number of possible rules is p^{q+1}. It is not an easy task to determine a small subset of rules from such a large "rule space" that would be suitable for controlling the process.

Because fuzzy control systems are highly nonlinear with many input and output variables, GAs[34–36] often are used to optimize the control rules, which are capable of rapidly locating a near-optimal solution to difficult problems.

Karr[17] applied GAs for learning the membership functions of fuzzy controllers. Considering isosceles triangles for the membership functions, a GA with conventional binary coding was used to move and to expand or to shrink the base of each triangle. The fitness function minimized by GAs was defined by

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

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 weighting constants. Karr used four different initial conditions (Cases 1–4) for tuning the membership functions. Clearly, the performance of the system can be strongly influenced by the choice of 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[19] 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 is measured by $500 - T$, where T is the average time steps required by the controller to be sufficiently close to the set point. Thrift used an ordinary two-point crossover operation. The mutation operation can alter the allele value to its immediate upper level or immediate lower level or to a blank code. A blank code indicates existence of no rule corresponding to that combination of input fuzzy sets.

Nomura et al.[20] also used GAs 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. Homaifer and McCormick[24] 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 and Candel[22] first showed that a new fuzzy reasoning model (NFRM) controller outperforms the conventional FRM controller. To illustrate an application of the NFRM to the DC series motor, they used two expert-provided fuzzy relation matrices. Then, they showed that the performance of NFRM controllers can be enhanced by GAs to derive the optimal fuzzy relation matrices and fuzzy membership functions. The evaluation function used in GAs is

$$J = 1/(1 + e^2)$$

Here, the mean-square error e^2 is

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

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 showed that if domain knowledge is used in the initialization procedure, it is exploited by the GA leading to faster convergence and better rule base.

Herrara et al.[23] proposed a GA-based tuning method for the parameters of membership functions used to define fuzzy control rules. This method minimized a squared-error function defined in terms of the training data.

Homaifar and McCormick[24] presented a method for simultaneous design of membership functions and a rule set using GAs. The 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 through generation 30) and a refinement stage. In the evolution stage, the GA was used to find satisfactory controllers, i.e., controllers that moved in the correct direction. 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 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.

Lim et al.[31] also used the GA to learn fuzzy rules. It requires 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 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). The mutation operator also is modified because of the 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 is

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

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 poll 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.[27] proposed a method of optimizing different control parameters of a multi-input and multioutput fuzzy control system based on the GA. They used the Takagi-Sugeno model[18,37]:

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

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

$$y = \sum_{i=0}^{n} \alpha_i \cdot y_i \Bigg/ \sum_{i=0}^{n} \alpha_i$$

where $\alpha_i = \min(\mu_{A_1^i}(x_1), \ldots, \mu_{A_m^i}(x_m))$ and μ_j^i is a bell-shaped membership function. They used the GA to select the optimal values of the consequent as well as antecedent membership parameters. The fitness function used is

$$J = \sum_{i=1}^{n} \beta |x_i - x_i^d| \quad \text{or} \quad J = \sum_{i=1}^{n} \beta (x_i - x_i^d)^2$$

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

Carse et al.[25] presented a GA-based approach to design fuzzy controllers for multiple input-multiple output (MIMO) systems called Pittsburgh-style fuzzy classifier system (P-FCS1) based on the Pittsburgh model of learning classifier systems.[38,39] They used a Mamdani-Assilian[26]-type model with symmetrical triangular membership functions for both input and output variables. Each membership function is represented by a pair, center and width. 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 that tries to preserve the epistatical linkage between genes representing rules with overlapping fuzzy sets.

In Wong and Fan's[32] method, the complete rule base with n input variables and one output variable is expressed as

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

where x_i and u stand for input and output linguistic variables. The term $A_{(i,j)}$ 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. The $f(j_1, \ldots, j_n) = \langle a_1 j_1 + \cdots + a_n j_n \rangle$, 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 rule base 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 the 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.

The GA has been used by Wong and Her[29] to eliminate unnecessary fuzzy sets to obtain a system with fewer rules. The rule structure is the same as that described in Ref. 32. Wong and Her used triangular, assymetrical membership functions with 50% overlap. Each chromosome contains the parameters that describe membership functions of the fuzzy sets of input and output variables. The fitness function used takes into account both the number of rules and the performance of the rule base.

Chan et al.[30] presented a GA-based optimized FLC (OFLC). They made some modifications on simple GAs (SGAs) to improve its performance. The rule base may be initialized with an expert specified suboptimal one to speed up the convergence. They used symmetrical rule tables, so the first half of the string is mirrored to the other half after crossover and mutation. The initialization of population is also done only on the first half. The fitness function is 1/(1 + ITAE), where ITAE is the integral time absolute error.

In this study, we first discuss some problems specially associated with the mutation operator of the OFLC of Chan et al.[30] and theoretically analyze its computational performance. Then, we present a new scheme improved and simplified OFLC (ISOFLC) with a symmetrical rule table, where we do not need to consider all the rules in the rule table. Instead, the GA derives an n-rule fuzzy system, where n is not fixed. The variable-length rule set will be able to grow or shrink according to the need of the problem. The lower the number of rules in the rule set, the lower would be the search space of the GA and the computation time to determine the output of the system also would be low. Our evaluation function itself keeps the balance between the number of rules and the performance of the rule base on the system. The performance of the scheme is shown on the inverted pendulum problem and compared with the work of Lim et al.[31] Some better theoretical properties of our scheme over that of Chan et al.[30] are discussed also.

2. BRIEF OVERVIEW OF GAs

GAs[34] are probabilistic heuristic search processes based on concepts of natural genetic systems. They are highly parallel and are 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.

GAs are capable of solving a wide range of complex optimization problems using three simple genetic operations (reproduction/selection, crossover, and mutation) on coded solutions (chromosomes/strings) for the parameter set, not the parameters themselves, in an iterative fashion. GAs consider several points in the search space simultaneously, which reduces the chance to converge to a local optima. They use the payoff or penalty (i.e., objective) function called the fitness function and do not need any other auxiliary information.

Let us consider the problem of optimizing a complex function having k parameters $p_1, p_2, \ldots, p_k$. To solve such a problem, GAs start with a set of initial strings/chromosomes $P = S_i$; $i = 1, 2, \ldots, N$ as the initial approximations of the parameter set. In GA literature, P is called a population. Each chromosome S_i represents a coded version of an approximate solution set $c_i = (a_{i1}, a_{i2}, \ldots, a_{ik})$. Usually, a binary chromosome of length $L_b = k \cdot l_b$ is taken as a string or chromosomal representation of an approximation. The substring comprising bits $(i - 1)*l_b$ through $i \cdot l_b$, $i = 1, 2, \ldots, k$, represents an approximation of the ith parameter. The population of the chromosomes then undergoes a sequence of three genetic operations to produce usually an improved population. These three operations are selection, crossover, and mutation. This process is repeated until some stopping criteria is reached.

3. OFLC OF CHAN ET AL.[30] AND SOME REMARKS

3.1. Chromosome Representation

In the work by Chan et al.[30] the input and output fuzzy sets are modeled by symmetrical and triangular membership functions. A chromosome represents a candidate solution of the problem, i.e., a rule set for the FLC. 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 (input) variables x_1 and x_2 and one consequent (output) variable, say y. Let the number of terms 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 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 consequences 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 the label of an output linguistic value to be used for a given rule. For example, the seven linguistic values (fuzzy sets) negative big (NB), negative medium (NM), negative small (NS), zero (Z), positive small (PS), positive medium (PM), and positive big (PB) of a linguistic variable are coded as 1, 2, 3, 4, 5, 6, and 7, respectively.

3.2. Symmetrical Rule Table, Mirror Action, and Initialization

The system is assumed to be symmetrical and hence the rule base is taken as symmetrical, which can balance the system from both directions to the set point. Symmetrical rule tables reduce the search space.

Because of the symmetrical nature of the system, 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. The mirror image of a site of a chromosome is given by another site at the same distance with respect to the line of symmetry in the opposite direction. The line of symmetry is located at (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 = (number of fuzzy subsets(l) + 1) − k; in the case of NS (3), the index of its image is $5 = (7 + 1) - 3$, i.e., PS.

Chan et al.[30] initialized the population using a random generator and/or used an expert-provided rule set. Because a good initial population may speed up the convergence, a population may be constructed based on the available information or common sense. However, they claimed that their algorithm also can optimize the controller with an empty rule base.

3.3. Fitness Function

The fitness function used by Chan et al.[30] is

$$\text{fitness} = 1/(1 + \text{ITAE}) \tag{1}$$

Here, ITAE of a rule set, for d initial conditions, is defined as

$$\text{ITAE} = \sum_{i=1}^{d} (\text{ITAE})_i$$

3.4. Reproduction

The reproduction module consists of parent selection, crossover, and mutation. First, a pair of parents is selected, and then *either* crossover *or* mutation is applied to produce two children. The process is repeated to produce a new population of children of the same size as that of the original population. Unlike simple GAs, where mutation follows crossover, this algorithm performs *only one of the two operations* on selected parents. If the crossover rate is 0.7 and the mutation rate is 0.3, then the parents have a 70% chance of performing crossover and a 30% chance of mutation.

3.4.1. *Parents Selection*

Roulette wheel selection[34] is used in the OFLC to select parents. 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.[34] The chance of a string to be selected as a parent thus becomes

$$p_i = (f_i + f_{\max}) \Big/ \left(\sum f_i + N{*}f_{\max} \right), \quad i = 1, 2, \ldots, N$$

where N is the size of population and $f_{\max}$ is the maximum fitness value of any chromosome in that generation.

3.4.2. *Crossover*

One-point crossover is used on the first half of chromosomes. After crossover, the first half of a child is mirrored to the second half.

3.4.3. *Mutation*

Chan et al.[30] uses a “one-step change mutation scheme,”[33] which changes a randomly selected allele value to either its next or previous value with equal probability. Mutation is done on the first half of the chromosome and then it is mirrored to the second half. Let n_m be the number of mutations to be done on a chromosome. Then, the following steps realize the mutation:

(1) Select two chromosomes by parent selection
(2) While $n_m > 1$ do (3)–(5)
(3) Generate a random integer R_1 from 1 to (length of the chromosome $-$ 1)/2 to select which site is to be mutated

(4) Generate another random number R_2 in [0, 1] to determine whether to increase or decrease the rule output by one; if $R_1 \leqslant 0.5$, then we increase the allele value at site R_1 by 1 or decrease the allele value at site R_1 by 1; when the allele value at site R_1 is 1, it will not be decreased and when it is l, it will not be increased further
(5) $n_m = n_m - 1$
(6) Mirror the first half of the chromosome to the second half; similar action is applied to the second chromosome

3.5. Generation Selection

Generation selection is different from parent selection. It selects the next generation population. In the OFLC steady state without duplicates (SSWOD)[35] 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, the existing population and their N children, i.e., total $2N$ individuals are passed to the generation selection module to select the best-fit population of N individuals for the next generation.

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

3.6. Some Remarks on OFLC

3.6.1. Search Space Reduction

If the controller has k inputs and one output each with d linguistic values, then the total number of possible rule sets (search space) is d^{d^k}. For example, for a two-inputs and one-output controller with seven linguistic values for each input and output variable, the search space contains 7^{49} rule sets. For a symmetrical controller, there exists an equilibrium point. So, the search space is reduced to 7^{24} in Chan et al.'s[30] method.

The search space and the computation time in each generation can be reduced further if the evolutionary process allows deletion of bad and redundant rules. Chan et al.'s[30] method did not entertain this.

3.6.2. One-Step Change Mutation Scheme

Chan et al.[30] used a "one-step change mutation scheme,"[33] where the current consequent is changed to either the immediately preceding or the next one. Although, this scheme can improve local search, it can greatly influence the maintenance of diversity, particularly when the population is initialized without any prior knowledge or in a situation when the entire population has converged to a homogeneous one. In the first case, the rate of convergence is expected to be low and in the second case the whole population may get stuck to a nonoptimal/suboptimal solution. For these reasons, we think, Chan et al.[30] insisted on using prior knowledge about the system to get the initial population, and the OFLC requires checking whether a new offspring is transferred to the next generation.

As mentioned earlier, when there is no offspring in a new population, the number of mutations in a chromosome is increased by one in the next generation, and the whole population is replaced by another randomly generated population, except for the best chromosome, which is kept. This is nothing but a new run with added information and with more mutation probability to restore diversity. In their one-step change mutation scheme, the exploration property of the GA may get significantly affected. This will be more prominent if the initial population is not generated using expert knowledge, and then the performance of OFLC could be poor. Our analysis in the next section reveals that indeed it is the case.

4. ANALYSIS OF OFLC

Suppose we have a chromosome representing a rule base containing only one (actually two because it is symmetrical) bad rule at the rth site. The allele value at the site corresponding to the bad rule is l, which should be 1. Now, in Chan et al.'s[30] scheme, l can never be changed to 1 by a single mutation. Let us calculate the expected number of mutations (steps) required to change the value of the rth site from l to 1 and its variance. A random site out of n (here, $n = (M - 1)/2$, where M is the length of the symmetrical chromosome) is selected. If the rth site is selected, then we reduce the content of the rth site by 1, i.e., make it $(l - 1)$ with probability 1 (as maximum value of the content is l). The random selection of site is continued until we get the rth site again. Then, the content of the rth site is either decreased to $(l - 2)$ or increased to l from $(l - 1)$ with equal probability (i.e., 1/2). The whole procedure is continued until the content of the rth site becomes 1. We will now find the expected number of trials (i.e., choosing a site) necessary to get 1 in the rth site and its variance.

We call a trial successful if rth site is selected. Suppose L is the number of trials necessary to get J successes, where J is the number of successes needed to make the content of the rth site as 1; L is obviously a random variable. It follows a negative binomial distribution

$$P(L = l^* | J) = \binom{l^* - 1}{J - 1} \left(\frac{1}{n}\right)^J \left(1 - \frac{1}{n}\right)^{l^* - J}$$

The expectation and variance of $L|J$ can be derived as

$$E(L|J) = nJ, \quad V(L|J) = n(n-1)J$$

The J is also a random variable. Thus, the random variable L is nothing but the number of trials necessary to make the content of the rth site 1. We have to find out

$$E(L) \quad \text{and} \quad V(L)$$

We know that $E(L) = E(E(L|J)) = E(nJ) = nE(J)$ and $V(L) = E(V(L|J)) + V(E(L|J)) = E(n(n-1)J) + V(nJ) = n(n-1)E(J) + n^2V(J)$. Now, we shall find $E(J)$ and $V(J)$.

Consider a Markov chain[40] with l states with the following transition matrix:

$$\mathbf{P}_{l\times l} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & \cdots & 0 & 0 \\ & & & & \cdots & & \\ 0 & 0 & 0 & 0 & \cdots & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{l\times l}$$

The ijth element of the transition matrix $\mathbf{P}$ gives the probability of going to the jth state from the ith state. This is a special type of Markov chain known as a random walk model. State 1 is an absorbing state. Once the content of the rth site becomes 1, the process ends. If the content of the rth site becomes l, it becomes $l - 1$ by the next chance. The transition matrix can be partitioned in the following way:

$$\mathbf{P} = \begin{pmatrix} \mathbf{1} & \mathbf{O}' \\ \mathbf{R} & \mathbf{Q} \end{pmatrix}$$

where $\mathbf{O}' = (0, 0, \ldots, 0)$, $\mathbf{R}' = (1/2, 0, 0, \ldots, 0)$, and

$$\mathbf{Q}_{l-1\times l-1} = \begin{pmatrix} 0 & \frac{1}{2} & 0 & \cdots & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & \cdots & 0 & 0 \\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{l-1\times l-1}$$

For an absorbing Markov chain, $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$ is the fundamental matrix. Let us denote $m = l - 1$. To compute $\mathbf{N}$, we use $\mathbf{LU}$-decomposition of $(\mathbf{I} - \mathbf{Q})$, i.e.,

$$\mathbf{I} - \mathbf{Q} = \mathbf{LU}$$

where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively, which can be computed from $(\mathbf{I} - \mathbf{Q})$ matrix as

$$\mathbf{L} = [L_{ij}]_{m\times m} = \begin{pmatrix} \frac{2}{2} & 0 & 0 & \cdots & 0 & 0 \\ \frac{-1}{2} & \frac{3}{4} & 0 & \cdots & 0 & 0 \\ 0 & \frac{-1}{2} & \frac{4}{6} & \cdots & 0 & 0 \\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & \frac{(m-1)+1}{2(m-1)} & 0 \\ 0 & 0 & 0 & \cdots & -1 & \frac{1}{m} \end{pmatrix}$$

i.e.,

$$L_{ij} = \begin{cases} \frac{i+1}{2i} & \text{if } i = j \text{ and } i \neq m \\ \frac{1}{m} & \text{if } i = j \text{ and } i = m \\ \frac{-1}{2} & \text{if } i-1 = j \text{ and } j \neq m \\ -1 & \text{if } i-1 = j \text{ and } j = m \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{U} = [U_{ij}]_{m\times m} = \begin{pmatrix} 1 & \frac{-1}{2} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \frac{-2}{3} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \frac{-3}{4} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ & & & & \cdots & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & \frac{-(m-1)}{m} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

i.e.,

$$U_{ij} = \begin{cases} 1 & \text{if } i = j \\ \frac{-i}{i+1} & \text{if } i+1 = j \\ 0 & \text{otherwise} \end{cases}$$

Thus,

$$\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1} = (\mathbf{LU})^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$$

Here,

$$\mathbf{L}^{-1} = [L_{ij}^{-1}]_{m\times m} = \begin{pmatrix} 2\times\frac{1}{2} & 0 & 0 & \cdots & 0 & 0 \\ 2\times\frac{1}{3} & 2\times\frac{2}{3} & 0 & \cdots & 0 & 0 \\ 2\times\frac{1}{4} & 2\times & 2\times\frac{3}{4} & \cdots & 0 & 0 \\ & & & \cdots & & \\ 2\times\frac{1}{m} & 2\times\frac{2}{m} & 2\times\frac{3}{m} & \cdots & 2\times\frac{m-1}{m} & 0 \\ 2\times 1 & 2\times 2 & 2\times 3 & \cdots & 2\times(m-1) & m \end{pmatrix}$$

i.e.,

$$L_{ij}^{-1} = \begin{cases} 2\times\frac{j}{i+1} & \text{if } i > j \text{ and } i < m \\ 2\times j & \text{if } i = m \text{ and } j \neq m \\ j & \text{if } i = j \text{ and } j = m \\ 0 & \text{otherwise} \end{cases}$$

and

$$\mathbf{U}^{-1} = [U_{ij}^{-1}]_{m\times m} = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{m-1} & \frac{1}{m} \\ 0 & 1 & \frac{2}{3} & \frac{2}{4} & \cdots & \frac{2}{m-1} & \frac{2}{m} \\ 0 & 0 & 1 & \frac{3}{4} & \cdots & \frac{3}{m-1} & \frac{3}{m} \\ & & & & \cdots & & \\ 0 & 0 & 0 & 0 & \cdots & 1 & \frac{(m-1)}{m} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

i.e.,

$$U_{ij}^{-1} = \begin{cases} \frac{i}{j} & \text{if } i \leqslant j \\ 0 & \text{otherwise} \end{cases}$$

So, we get

$$\mathbf{N} = [N_{ij}]_{m\times m} = \begin{pmatrix} 2 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 2 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 6 & \cdots & 6 & 6 & 3 \\ & & & \cdots & & & \\ 2 & 4 & 6 & \cdots & 2(m-2) & 2(m-2) & (m-2) \\ 2 & 4 & 6 & \cdots & 2(m-2) & 2(m-1) & (m-1) \\ 2 & 4 & 6 & \cdots & 2(m-2) & 2(m-1) & m \end{pmatrix} \tag{2}$$

i.e.,

$$N_{ij} = \begin{cases} 2j & \text{if } i \geqslant j \text{ and } j \neq m \\ 2i & \text{if } i < j \text{ and } j \neq m \\ i & \text{if } j = m \end{cases}$$

THEOREM. *Suppose n_{ij} is the total number of times that the process comes to state j starting from state i. Then, $\mathbf{E}((n_{ij})) = \mathbf{N}$ and $\mathbf{V}((n_{ij})) = \mathbf{N}_2 = \mathbf{N}(2\mathbf{N}_{dg} - \mathbf{I}) - \mathbf{N}_{sq}$, where $\mathbf{N}_{dg}$ is the matrix $\mathbf{N}$ after making all off diagonal elements zero and $\mathbf{N}_{sq}$ is the matrix where every element of $\mathbf{N}$ is squared.*[40] *We get, $\mathbf{N}_2 = [(N_2)_{ij}]$, where*

$$(N_2)_{ij} = \begin{cases} 2j(4j-1) - (2j)^2 & \text{if } i \geqslant j \text{ and } j \neq m \\ 2i(4j-1) - (2i)^2 & \text{if } i < j \text{ and } j \neq m \\ i(4j-1) - (i)^2 & \text{if } j = m \end{cases}$$

The set containing only state 1 is an ergodic set. The other states are transient states. Let $\mathbf{t} = (t_1, t_2, \ldots, t_m)^T$ be the vector giving the total number of steps (including the original position) that the process is in a transient state, where t_i = total number of steps needed to reach an ergodic set from the transient state i. In an absorbing chain, this is the time to absorption.

THEOREM. *$E(\mathbf{t}) = \boldsymbol{\tau}$ and $V(\mathbf{t}) = \boldsymbol{\tau}_2$, where $\boldsymbol{\tau} = \mathbf{N}\boldsymbol{\zeta}$ and $\boldsymbol{\tau}_2 = (2\mathbf{N} - \mathbf{I})\boldsymbol{\tau} - \boldsymbol{\tau}_{\text{sq}}$, $\boldsymbol{\zeta}$ is a column vector consisting of* 1*'s only and $\boldsymbol{\tau}_{\text{sq}}$ is the vector in which every element of $\boldsymbol{\tau}$ is squared.*[40]. *Thus, for our fundamental matrix* **N** *in Equation* 2, *we get*

$$\boldsymbol{\tau} = \begin{pmatrix} 1(2m-1) \\ 2(2m-2) \\ \vdots \\ i(2m-i) \\ \vdots \\ m(2m-m) \end{pmatrix}$$

Because we are interested in the last elements of $\boldsymbol{\tau}$ and $\boldsymbol{\tau}_2$, we give the results from $(\tau)_m$ and $(\tau_2)_m$ only, which are nothing but $E(J)$ and $V(J)$, respectively, i.e., expected number of steps needed to reach an ergodic state from state l and its variance.

$$\tau_m = m(2m - m) = m^2 \quad \textit{and} \quad (\tau_2)_m = \frac{m^2}{3}(2m^2 + 1)$$

So, we get

$$E(L) = nE(J) = nm^2 = n(l-1)^2$$

$$V(L) = n(n-1)m^2 + n^2\frac{m^2}{3}(2m^2+1)$$

$$= n(n-1)(l-1)^2 + n^2\frac{(l-1)^2}{3}(2(l-1)^2 + 1)$$

So, if $n = 24$ and $l = 7$, then

$$E(L) = 964 \quad \textit{and} \quad V(L) = 524{,}448$$

Thus, on average 964 *mutations* (*with a variance of* 524,448, *which is a very big number*) *may be required. Note that here we considered only one chromosome, so crossover operation was not considered.*

This analysis gives an indication that if expert knowledge is not available to initialize the population, the computational efficiencies of OFLC could be very poor. This can be improved if we replace the one-step change mutation by random mutation. We show it theoretically.

In random mutation, an allele is replaced with a random number between 1 to l. Thus, the allele value can go to 1 from l in a single mutation (one step), which is not at all possible in OFLC. We now calculate the expected number of mutations (steps) required to change the value of the rth site from l to 1 and its variance for random mutation. Let us define a trial as randomly selecting one of the sites and then randomly altering its value. The probability that the rth site will be selected and its content will be changed to 1 is $p = (1/n)(1/l)$ and $q = 1 - p$. Thus, p is the probability that the content of the site will be changed to 1 in one step and

q represents the probability that the content of the site will not be changed to 1 in one step. Now, the content of rth site can change to 1 in i steps, $i = 1, 2, \ldots, \infty$. So, the expected number of trials needed is

$$E(L) = \sum_{i=1}^{\infty} i \cdot p \cdot q^{i-1} = \frac{p}{(1-q)^2} = \frac{1}{p} = n \cdot l$$

and

$$V(L) = \left(\sum_{i=1}^{\infty} i^2 \cdot p \cdot q^{i-1} \right) - (E(L))^2 = \frac{2-p}{p^2} - \frac{1}{p^2} = \frac{1-p}{p^2}$$
$$= n^2 \cdot l^2 - n \cdot l = n \cdot l(n \cdot l - 1)$$

Assuming the same values of n $(=24)$ and l $(=7)$, we get $E(L) = 168$ and $V(L) = 28{,}056$. Thus, on average, 168 mutations (with variance 28,056) can change the value of a particular site from l to 1, and for OFLC, it is 964, where variance is also very high (524,448).

5. PROPOSED SCHEME: ISOFLC

We shall illustrate our method keeping in mind the inverted pendulum problem that has two input variables and one output variable. An OFLC does not allow rule deletion. Consequently, the rule base will always have $m \times n$ rules, where m and n are the number of fuzzy sets associated with the two input variables. The scheme proposed here, called the ISOFLC, entertains rule deletion also and thereby reduces the search space drastically. The execution time for each generation also will be greatly reduced because for evaluation of fitness of each chromosome it will process fewer than $m \times n$ rules. To achieve this, we need some modifications and changes in the chromosome representation, initial population, fitness function, and in the mutation operation as described next.

5.1. Chromosome Representation

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. If a chromosome contains an allele value of zero at position i, then in that rule base there will be no rule with the antecedent clause corresponding to the ith site.

5.2. 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 the chromosome. Positions of these rules in the chromosome also 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.

We do not restrict the GA to derive a rule base having number of rules between q and Q. Then, q and Q will be treated as constraints in the learning process. 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 the problem. So, we use the same crossover operation of OFLC in ISOFLC.

5.3. Fitness Function

We like to get a rule base with fewer rules but without compromising the performance of the system. The rule base should be able to bring the system to the target state with lower ITAE. So, we introduce a new fitness function that uses number of rules as well as ITAE. The simplified fitness function that can decide on the trade-off between the number of rules and the quality of performance is

$$\text{fitness} = \frac{1}{w_1 \cdot n_r + w_2 \cdot \text{ITAE}} \tag{3}$$

Here, n_r is the number of rules in the rule set and ITAE is already defined in Equation 1. The w_1 and w_2 are the weights associated with the number of rules and the ITAE, respectively. We have used a convex combination of n_r and ITAE, i.e., $w_1 + w_2 = 1$.

5.4. Mutation

We have used random mutation. If the number of linguistic terms associated with the output variable is l, in this mutation scheme, an allele is randomly replaced with a random number between zero to l. So, it can

(i) *Insert* a rule by selecting a site of zero allele value and setting it to a randomly selected nonzero value in $1, 2, \ldots, l$
(ii) *Delete* a rule by selecting a site of nonzero allele value and setting it to zero
(iii) *Modify* a rule by selecting a site of nonzero allele value and setting it again to a randomly selected different nonzero value in $1, 2, \ldots, l$

This mutation scheme can maintain the diversity of the population. It also helps the learning process to get a rule base having fewer rules because of its rule deletion property.

5.5. Initial States

The rule base should be such that the controller is able to operate over the entire input space. To satisfy this criterion, we use a sufficient number of initial states distributed over the entire input space to compute the fitness of an individual chromosome. The more the number of initial states, the more will be the degree of

Table I. Initial positions used.

Initial positions
(−0.18, −1.8)(−0.18, 1.8)(0.18, −1.8)(0.18, 1.8)
(−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)
(−0.06, −0.6)(−0.06, 0.6)(0.06, −0.6)(0.06, 0.6)
(−0.03, 0)(0, −0.3)(0.03, 0)(0, 0.3)

reliability on the extracted rule set; but it will increase the design time of the controller. So, a trade-off is made by the designer. Initial states used in ISOFLC are shown in Table I.

The proposed scheme selects, in a self-organized manner, an "optimal" subset of rules from all possible rules, which attempts to bring the system to its set point within a short time (measured by the number of time steps), keeping in view that the controller will operate over the entire input space. ISOFLC attempts to achieve the following:

(i) To minimize the number of rules (considering the variable n_r in the fitness function and allowing deletion in the mutation)
(ii) To minimize the average time to reach the set point, i.e., settling time (by using ITAE to evaluate the fitness)
(iii) To ensure that the controller operates over the entire input space (by proper choice of initial states)

6. SIMULATION RESULTS OF ISOFLC

We have implemented the proposed method (ISOFLC) on the inverted pendulum problem with several initial populations. In most cases we get a rule base having few rules and low ITAE within a few generations without any guidance from human experts.

The inverted pendulum problem is a well-modeled control problem commonly used for establishing fuzzy control algorithms and development tools. The inverted pendulum 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 the horizontal X direction. When the cart is at rest, the pole is balanced in the vertical position and the force u on the cart is zero and then the system is in equilibrium. 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 $\dot{\theta} = 0$). The control rules are of the form

$$\text{if } (\theta \text{ is } \{\text{NB}, \ldots, \text{PB}\}) \quad \text{and} \quad (\dot{\theta} \text{ is } \{\text{NB}, \ldots, \text{PB}\}) \quad \text{then} \quad (u \text{ is } \{\text{NB}, \ldots, \text{PB}\})$$

Here, θ is the angular displacement, $\dot{\theta}$ is the angular velocity of the pole, and u is the force applied on the cart. The terms θ and $\dot{\theta}$ are the input linguistic variables and u is the output linguistic variable. For simplicity, we have ignored the cart-positioning part and constrained ourselves only to pole balancing.

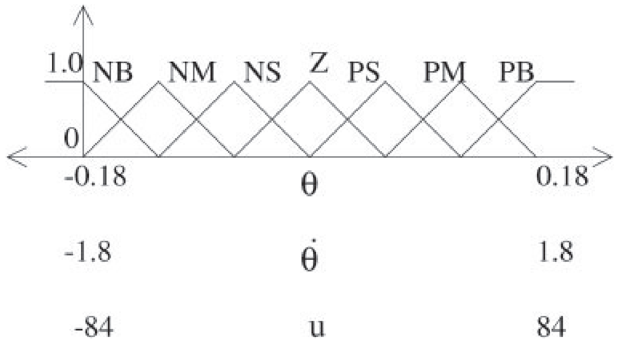

Figure 1. Fuzzy membership functions of the linguistic values associated with input linguistic variables θ and $\dot{\theta}$ and output linguistic variable u.

We have used the following computational protocols: pole length, 0.5 m; pole mass, 0.1 kg; cart mass, 2 kg; the maximum allowable angular deviation, 0.18 rad; and angular velocity, 1.8 rad/second. Each of the linguistic variables θ, $\dot{\theta}$, and u has seven linguistic values or fuzzy sets: NB, NM, NS, Z, PS, PM, and PB. We used isosceles triangles as membership functions with 50% overlap with the neighboring triangles. The membership functions of the input and output linguistic variables are shown in Fig. 1. All membership functions have equal base length. This is possibly the most natural and unbiased choice for the membership functions.

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. In our simulation, each chromosome is tested with the 20 different initial positions, shown in Table I, each of which is simulated for $T = 200$ time steps. Different initial conditions test the rule set for its ability to control different situations. The choice of initial positions plays a vital role in achieving the stability of the controller over the entire input space.

Parameters of the GAs:

$$\text{Population size} = 50$$

$$\text{Mutation rate} = 0.3$$

$$\text{Crossover rate} = 0.7$$

$$\text{Maximum number of generations} = 50 \text{ (for proposed scheme)}$$

$$= 100 \text{ (for OFLC)}$$

Number of rules in a chromosome in the initial population (in ISOFLC) is taken in the range of 7–12 (generated randomly). The pole is considered to be balanced if $\theta \leqslant 0.001$ rad, and $\dot{\theta} \leqslant 0.01$ rad/second within 100 time steps. The termination condition for the simulation is set to be either the maximum number of generations or the fitness does not increase in five successive iterations. We generated 1369 ($=37 \times 37$) samples (initial states) uniformly distributed over the product space

Table II. Results of ISOFLC but with the fitness function of Chan et al.[30]

	$\dot{\theta}$						
θ	NB	NM	NS	Z	PS	PM	PB
A rule set containing 17 rules (ITAE = 1.6751)							
NB	2	0	1	0	0	0	0
NM	1	3	0	0	1	0	0
NS	0	0	0	1	0	0	0
Z	3	0	2	4	6	0	5
PS	0	0	0	7	0	0	0
PM	0	0	7	0	0	5	1
PB	0	0	0	0	7	0	6
A rule set containing 19 rules (ITAE = 1.6361)							
NB	2	1	0	1	0	0	0
NM	1	0	3	0	0	0	0
NS	0	5	0	2	0	0	0
Z	0	1	3	4	5	7	0
PS	0	0	0	6	0	3	0
PM	0	0	0	0	5	0	7
PB	0	0	0	7	0	7	6

$\theta \times \dot{\theta}$, which are used to examine the stability and performance of the rule set extracted.

To show the effectiveness of our fitness function in Equation 3, we implemented the proposed scheme but with the fitness function of Chan et al.[30] described in Equation 1. We made several simulations. Two of them are shown in Table II having 17 and 19 rules and their corresponding ITAEs are 1.6751 and 1.6361, respectively. Using our fitness function in Equation 3, with $w_1 = 0.3$ and $w_2 = 0.7$ (say, Case I), we get rule sets having much less ITAE within 50 generations. Here, we report only two of them in Table III. Although they have the same number of rules as in Table II, their ITAEs are much less, 0.8759 and 0.7250, respectively. Increasing w_1 to 0.5, i.e., $w_1 = w_2 = 0.5$ (say, Case II), ISOFLC, within 25 generations, settles to rule sets having fewer rules but with slightly higher ITAEs. We report only two of them in Table IV. Table IV shows a rule set with 9 rules and an ITAE = 6.5314 and 11 rules with an ITAE = 4.4657.

To ascertain the quality of the rule sets extracted by ISOFLC, we have examined each of the 1369 (=37 × 37) initial conditions over the entire input space. Table V shows, in most cases, an inverse relation between the number of rules and the quality (measured by ITAE) of a rule base produced by ISOFLC, and all rule bases considered in the Table V are able to bring the system to the target state. So, by choosing a proper combination of the values of w_1 and w_2, a trade-off between the number of rules and the performance (in terms of ITAE) can be decided as required by the user.

Table VI shows the performance comparison of the results obtained by ISOFLC (for both Case I and Case II) and OFLC of Chan et al.,[30] averaged over 10 runs. Note that OFLC used an exhaustive rule set, i.e., $m \times n$ (=49) rules in the rule set, whereas the average number of rules in the rule set selected by ISOFLC is only 18 and 11.4 in Cases I and II, respectively.

Table III. Results of ISOFLC with $w_1 = 0.3$, $w_2 = 0.7$.

	$\dot{\theta}$						
θ	NB	NM	NS	Z	PS	PM	PB
A rule set containing 17 rules (ITAE = 0.8759, average time steps = 24.1)							
NB	1	0	1	0	1	0	0
NM	0	2	0	1	0	0	0
NS	0	0	0	1	0	0	0
Z	0	2	3	4	5	6	0
PS	0	0	0	7	0	0	0
PM	0	0	0	7	0	0	0
PB	0	0	7	0	7	0	7
A rule set containing 19 rules (ITAE = 0.7250, average time steps = 23)							
NB	1	0	1	1	0	0	0
NM	0	1	0	0	0	2	0
NS	0	1	0	1	0	0	0
Z	2	0	2	4	6	0	6
PS	0	0	0	7	0	7	0
PM	0	6	0	0	0	7	0
PB	0	0	0	0	7	7	7

7. COMPARISON OF OUR RESULTS WITH LIM ET AL.'S WORK HAVING THE SAME GOAL

7.1. Some Remarks on Lim et al.'s[31] Method

Given fixed domains and symmetrical triangular membership functions for each input and output variables, Lim et al.[31] described a learning process based on

Table IV. Results of ISOFLC with $w_1 = w_2 = 0.5$.

	$\dot{\theta}$						
θ	NB	NM	NS	Z	PS	PM	PB
A rule set containing 9 rules (ITAE = 6.5314)							
NB	1	1	0	1	0	0	0
NM	0	0	0	0	0	0	0
NS	0	0	0	2	0	0	0
Z	0	0	2	4	6	0	0
PS	0	0	0	6	0	0	0
PM	0	0	0	0	0	0	0
PB	0	0	0	7	0	7	7
A rule set containing 11 rules (ITAE = 4.4657)							
NB	1	1	0	1	0	0	0
NM	0	0	0	0	0	0	0
NS	0	0	0	2	0	0	0
Z	0	0	2	4	6	0	0
PS	0	0	0	6	0	0	0
PM	0	0	0	0	0	0	0
PB	0	0	0	7	0	7	7

Table V. Number of rules and the corresponding ITAE and fitness of 10 different rule sets generated by ISOFLC for Cases I and II.

Case I $w_1 = 0.3$, $w_2 = 0.7$			Case II $w_1 = 0.5$, $w_2 = 0.5$		
No. of rules	ITAE	Fitness	No. of rules	ITAE	Fitness
15	0.9083	0.19	9	6.5314	0.1288
17	0.8759	0.17	9	6.8155	0.1264
17	0.9006	0.17	11	4.0702	0.1328
17	1.0559	0.17	11	4.1371	0.1322
17	1.1708	0.16	11	4.2023	0.1316
19	0.7250	0.16	11	4.5851	0.1284
19	0.8873	0.17	13	2.0513	0.1328
19	0.9530	0.15	13	2.1703	0.1318
19	1.0296	0.15	13	2.4203	0.1296
21	1.0406	0.14	13	2.7528	0.1270

GA to derive fuzzy control rules. We have already described their algorithm briefly in Section 1:

(1) They did not consider symmetrical rule base. So, the search space is large but flexibility is more.
(2) Number of rules in the rule base is fixed. An appropriate choice of the number of rules requires at least some knowledge of the underlying problem complexity, which may not always be available. Modifications of genetic operators such as crossover and mutation and other operators such as rule creation and rule deletion were required in their algorithm just to keep the number of rules in chromosome fixed. Fixing the number of rules may act as a constraint on the learning ability of the system.
(3) The fitness function of Lim et al.[31] is

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

For a test run with the ith initial condition, $T_{\theta i}$ denotes the number of time steps the pole remains at within 1° from the vertical position and T_{ei} denotes the number of time steps elapsed before the poll falls. Fitness function is suitable *only* for the inverted pendulum. Lim et al.[31] set $b = 0.6$ in their simulation. For each of these tests, the

Table VI. Performance comparison of ISOFLC and OFLC, averaged over 10 runs.

	ISOFLC		
	Case I	Case II	OFLC
No. of rules	18	11.4	49 (fixed)
Average no. of time steps	25.2	42	28
Average ITAE	0.9547	3.9732	1.0609
No. of balanced positions among 1369	1369	1369	1369
Failure (%)	0	0	0

Table VII. A typical rule set containing 20 rules obtained by the method of Lim et al.[31]

	$\dot{\theta}$						
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

cart-pole system is simulated until the pole falls or a prespecified value of T $(=200)$ time steps is reached.

We implemented the algorithm of Lim et al.[31] on our cart-pole system. Table VII 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 6 for Lim et al.'s[31] method, because their rule set is not able to set the cart-pole system to that precision. The pole is considered balanced if $\theta \leqslant 0.005$ for at least five time steps. This may be caused by higher importance on T_{ei} in the fitness function, which did not care for the set point. Table VIII compares ISOFLC with Lim et al.'s[2] method, averaged over 10 runs. We repeated our experiment 10 times and the average number of rules was 18 with an average ITAE of 0.9547 in Case I and these are, respectively, 11.4 and 3.9732 in Case II, whereas the average ITAE of Lim et al.'s[2] method is about 300 with 20 rules.

8. CONCLUSIONS

Chan et al.[30] proposed an algorithm based on the GA, called OFLC, to derive an optimal rule set. They used existing knowledge of the system to increase the speed of optimization. We first addressed a few limitations of the OFLC, partic-

Table VIII. Performance comparisons of ISOFLC and Lim et al.'s[31] method (averaged over 10 runs).

	ISOFLC		
	Case I	Case II	Lim et al.[31]
No. of rules	18	11.4	20 (fixed)
Average no. of time steps considering only balanced positions	25.2	42	40[a]
Average ITAE	0.9547	3.9732	300
No. of balanced positions among 1369	1369	1369	1311[a]
Failure (%)	0	0	3.5[a]

[a]For Lim et al.'s[31] method, the balanced condition is taken as $\theta \leqslant 0.005$ for at least five time steps.

ularly, the problems that may arise because of one-step change mutation. We then suggested modifications to avoid these problems by random mutation. We analyzed the theoretical properties of one-step change mutation and random mutation. We also proposed a new scheme, called ISOFLC to reduce the number of rules in the symmetrical rule set. Our fitness function allows the user to decide on the trade-off between cost (number of rules) and quality (ITAE). Given the required precision (with respect to balancing), our fitness function is able to generate a small rule set, which can bring the system to the set point from anywhere over the domain of input variables. The superiority of our scheme is established by comparing it with Chan et al.'s method[30] and also with Lim et al.'s[31] method.

References

1. Klir GJ, Yuan B. Fuzzy sets and fuzzy logic—theory and applications. Englewood Cliffs, NJ: Prentice Hall PTR; 1995.
2. Zadeh LA. Fuzzy logic and approximate reasoning. Syntheses 1975;30:407–428.
3. Lee CC. Fuzzy logic in control system: Fuzzy logic controller—Part I. IEEE Trans Syst Man Cybern 1990;20:404–418.
4. Lee CC. Fuzzy logic in control system: Fuzzy logic controller—Part II. IEEE Trans Syst Man Cybern 1990;20:419–435.
5. Driankov D, Hellendoorn H, Reinfrank M. An introduction to fuzzy control. New York: Springer-Verlag; 1993.
6. Yamakawa T. A fuzzy logic controller. J Biotechnol 1992;24:1–32.
7. Takagi T, Sugeno M. Fuzzy identification of systems and its applications to modeling and control. IEEE Trans Syst Man Cybern 1985;15:116–132.
8. Sugeno M. Industrial applications of fuzzy control. Amsterdam: Elsevier Science; 1985.
9. Lin C, Lee CSG. Neural-network–based fuzzy logic control and decision system. IEEE Trans Comput 1991;40:1320–1336.
10. Shann JJ, Fu HC. A fuzzy neural network for rule acquiring on fuzzy control system. Fuzzy Sets Syst 1995;71:345–357.
11. Pal NR, Pal T. On rule pruning using fuzzy neural networks. Fuzzy Sets Syst 1999;106: 335–347.
12. Benitez JM, Blanco A, Requena I. An empirical procedure to obtain fuzzy rules using neural networks. In: Proc VII IFSA World Congr. Sao Paulo, Brazil; 1995. pp 663–666.
13. Ishibuchi H, Fujioka R, Tanaka H. Neural networks that learn from fuzzy if-then rules. IEEE Trans Fuzzy Syst 1993;1:85–97.
14. Lee K, Kwang D, Wang HL. Int J Uncertain Fuzziness Knowl Based Syst 1994;2:265–277.
15. Li CC, Wu CJ. Generating fuzzy rules for a neural fuzzy classifier. In: Proc 3rd IEEE Int Conf on Fuzzy Systems (FUZZ-IEEE '94). June 26–29, Orlando, FL; 1994. pp 1719–1724.
16. Yao S, Wei C, He Z. Evolving fuzzy neural networks for extracting rules. In: Proc 5th IEEE Int Conf on Fuzzy Systems (FUZZ-IEEE '96). New Orleans, LA; 1996. pp 361–367.
17. Karr CL. Design of an adaptive fuzzy logic controller using a genetic algorithm. In: Proc 4th Int Conf on Genetic Algorithms. San Mateo, CA; 1991. pp 450–457.
18. Carr CL. Genetic algorithm for fuzzy logic controller. AI Exp 1991;2:22–23.
19. Thrift P. Fuzzy logic synthesis with genetic algorithm. In: Proc 4th Int Conf on Genetic Algorithms. San Mateo, CA; 1991. pp 509–513.
20. Nomura H, Hayashi I, Wakami N. A self tuning method of fuzzy control by genetic algorithm. In: Proc Int Fuzzy Syst Intell Contr Conf (IFSICC '92). 1992. pp 236–245.

21. Karr CL, Gentry EJ. Fuzzy control of pH using genetic algorithms. IEEE Trans Fuzzy Syst 1993;1:46–53.
22. Park D, Kandel A, Langholz G. Genetic-based new fuzzy reasoning models with application to fuzzy control. IEEE Trans Syst Man Cybern 1994;24:39–47.
23. Herrera F, Lozano M, Verdegay JL. Tuning fuzzy logic controllers by genetic algorithm. Int J Approx Reason 1995;12:299–315.
24. Homaifar A, McCormick E. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. IEEE Trans Fuzzy Syst 1995;3:129–139.
25. Carse B, Fogarty TC, Munro A. Evolving fuzzy rule based controllers using genetic algorithms. Fuzzy Sets Syst 1996;80:273–293.
26. Mamdani EH, Assilian S. An experiment in linguistic synthesis with a fuzzy logic controller. Int J Man Mach Stud 1974;7:1–13.
27. Renhou L, Yi Z. Fuzzy logic controller based genetic algorithms. Fuzzy Sets Syst 1996;83:1–10.
28. Gurocak HB. A genetic-algorithm-based method for tuning fuzzy logic controllers. Fuzzy Sets Syst 1999;108:39–47.
29. Wong C-C, Her S-M. A self-generating method for fuzzy system design. Fuzzy Sets Syst 1999;103:13–25.
30. Chan PT, Xie WF, Rad AB. Tuning of fuzzy controller for an open-loop unstable system: A genetic approach. Fuzzy Sets Syst 2000;111:137–152.
31. Lim MH, Rahardja S, Gwee BH. A GA paradigm for learning fuzzy rules. Fuzzy Sets Syst 1996;82:177–186.
32. Wong C-C, Fan C-S. Rule mapping fuzzy controller design. Fuzzy Sets Syst 1999;108: 253–261.
33. Kinzel J, Klawonn F, Fruse R. Modifications of GA for designing and optimising fuzzy controller. In: Proc IEEE Int Conf on Computational Intelligence. June 27–29, Orlando, FL; 1994. pp 28–32.
34. Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Reading: Addison-Wesley; 1989.
35. Davis L. Handbook of genetic algorithms. Reinhold: Van Norstrand; 1991.
36. Bhandari D, Pal NR, Pal SK. Directed mutation in genetic algorithms. Inf Sci 1994;79: 251–270.
37. Sugeno M, Nishida M. Fuzzy control of model cat. Fuzzy Sets Syst 1985;16:103–113.
38. Nakaoka K, Furuhashi T, Uchikawa Y. A study on apportionment of credits of fuzzy classifier system for knowledge acquisition of large scale systems. In: Proc 3rd IEEE Int Conf on Fuzzy Systems. June 26–29, Piscataway, NJ; 1994. pp 1797–1800.
39. Parodi A, Bonelli P. A new approach of fuzzy classifier systems. In: Proc 5th Int Conf Genetic Algorithms. Los Atlos, CA. Morgan Kaufmann Publisher, San Mateo, CA; 1993. pp 223–230.
40. Kemeny JG, Snell JL. Finite Markov chains. New York: D. Van Nostrand Company, Inc.; 1960.