Parallel Distributed Hybrid Fuzzy GBML Models With Rule Set Migration and Training Data Rotation

Hisao Ishibuchi, *Senior Member, IEEE*, Shingo Mihara, and Yusuke Nojima, *Member, IEEE*

***Abstract*—We propose a parallel distributed model of a hybrid fuzzy genetics-based machine learning (GBML) algorithm to drastically decrease its computation time. Our hybrid algorithm has a Pittsburgh-style GBML framework where a rule set is coded as an individual. A Michigan-style rule-generation mechanism is used as a kind of local search. Our parallel distributed model is an island model where a population of individuals is divided into multiple islands. Training data are also divided into multiple subsets. The main feature of our model is that a different training data subset is assigned to each island. The assigned training data subsets are periodically rotated over the islands. The best rule set in each island also migrates periodically. We demonstrate through computational experiments that our model decreases the computation time of the hybrid fuzzy GBML algorithm by an order or two of magnitude using seven parallel processors without severely degrading the generalization ability of obtained fuzzy rule-based classifiers. We also examine the effects of the training data rotation and the rule set migration on the search ability of our model.**

***Index Terms*—Fuzzy rule-based classifiers, genetics-based machine learning, parallel distributed algorithms, training data rotation, training data stratification.**

I. Introduction

POPULATION-BASED evolutionary algorithms have a number of advantages over point-based search methods such as global search ability, adaptability to uncertain environments, and applicability to large-scale optimization problems [1]–[5]. Applications of evolutionary algorithms to machine learning have often been referred to as genetics-based machine learning (GBML) [1]. A number of GBML algorithms were proposed for machine learning, knowledge extraction, and data mining [6]–[11]. Recently, evolutionary multiobjective optimization was also used to handle conflicting objectives such as "accuracy and complexity" in classifier design [12]–[17].

GBML algorithms are divided into two categories depending on their coding mechanisms [11]. In one category, a rule set is coded as a string and handled as an individual. A single population consists of a number of rule sets. This category of GBML algorithms is referred to as the Pittsburgh approach. Pittsburgh-style GBML algorithms search for the best rule set with respect to a prespecified fitness function. In the other category, a single rule is coded as a string and handled as an individual. A single population consists of a number of rules. This category is subdivided into the Michigan approach and iterative rule learning (IRL). In the Michigan approach, a population of rules is viewed as a single rule-based classifier. The final population after the execution of a Michigan-style GBML algorithm is usually handled as an obtained rule-based classifier. In an IRL-style GBML algorithm, a rule-based classifier is designed by its multiple runs where a single rule is obtained from each run. For details of these three approaches, see [11].

The main difficulty in the application of Pittsburgh-style GBML algorithms to large-scale classification problems is their heavy computation load. This is because a number of rule sets are to be evaluated in each generation during their execution. Various techniques have been discussed for the speed-up of Pittsburgh-style GBML algorithms. One popular technique is their parallel implementation, where multiple processors are used in parallel for rule set evaluation. Parallel implementation is a general technique for the speed-up of evolutionary algorithms [18]–[21]. Parallel implementation of population-based search algorithms is not difficult since the fitness evaluation of each solution can be performed in parallel. Motivated by recent rapid advancement of parallel hardware technologies, such as general-purpose graphics processing units, parallel evolutionary computation has become a very active research area [22]–[26].

Training data reduction has also been actively studied in machine learning, knowledge extraction, and data mining [27]–[30]. Its basic idea is to decrease the size of training data. A windowing method was studied as a training data reduction method for the speed-up of GBML algorithms in [31] which training data were stratified into multiple strata (i.e., divided into multiple disjoint subsets of the same size with the same class distribution). A different training data subset was used in every generation in [31]. Further speed-up of GBML algorithms was realized by parallel implementation of GBML algorithms with the windowing method in [32] where the fitness evaluation of each solution was performed in parallel.

In our former studies [33], [34], we used an island model for parallel distributed implementation of genetic fuzzy rule selection [35]. A different training data subset was assigned to each island. Genetic fuzzy rule selection was performed independently in each island for a prespecified number of generations. Assigned training data subsets were rotated periodically over the islands. We used the term "distributed" because the fitness evaluation of each solution was performed at each island using

Manuscript received November 26, 2011; revised March 24, 2012; accepted June 19, 2012. Date of publication August 24, 2012; date of current version March 27, 2013. This work was supported in part by the Japan Society for the Promotion of Science under Grant-in-Aid for Young Researchers (B) (22700239) and Takayanagi Foundation for Electronics Science and Technology.

The authors are with the Department of Computer Science and Intelligent Systems, Osaka Prefecture University, Osaka 599-8531, Japan (e-mail: hsaoi@cs.osakafu-u.ac.jp; mihara@ci.cs.osakafu-u.ac.jp; nojima@cs.osakafu-u.ac.jp).

Digital Object Identifier 10.1109/TFUZZ.2012.2215331

a different training data subset. A simple migration strategy was used in [34] where copies of the best rule sets in each island were moved to the next island. In [33], subpopulations were periodically merged and randomly redivided into multiple islands.

Although genetic fuzzy rule selection is an interesting postprocessing procedure of fuzzy data mining [36], accurate fuzzy rule-based classifiers are not always obtained. This is because it does not generate any new fuzzy rules. When important fuzzy rules are not included in a candidate rule set, it is difficult to design fuzzy rule-based classifiers with high accuracy through rule selection.

We examined parallel distributed implementation of a hybrid fuzzy GBML algorithm [37] in our former studies [38], [39] where no migration strategy was used. In this paper, we propose a parallel distributed model with training data rotation and rule set migration as an extended version of island models in our former studies [38], [39]. A different training data subset is assigned to each island and rotated periodically over multiple islands. As a migration strategy, a copy of the best rule set (i.e., the best individual) at each island is sent to another island. As we will show through computational experiments, the training data rotation and the rule set migration interfere with each other. That is, their synchronized use clearly deteriorates the search ability of our hybrid fuzzy GBML algorithm. In this paper, we propose an idea of using an inverse rotation for the rule set migration. The training data rotation and the rule set migration are performed in opposite directions. To the best of our knowledge, the handling of such a mutual interference between the training data rotation and the rule set migration has not been discussed in the literature.

The main contributions of this paper can be summarized as follows.

1) We discuss the mutual interference between the training data rotation and the rule set migration. In order to avoid its negative effects, we propose a new parallel distributed model with the training data rotation and the rule set migration in opposite directions.
2) We demonstrate that the proposed model drastically decreases the computation time of our hybrid fuzzy GBML algorithm without severely deteriorating the generalization ability of obtained fuzzy rule-based classifiers.

This paper is organized as follows. First, we briefly explain fuzzy rule-based classifiers in Section II. Next, we explain our hybrid fuzzy GBML algorithm in Section III. Then, we propose a parallel distributed model with the training data rotation and the rule set migration in opposite directions in Section IV. In Section V, our parallel distributed model and its variants are examined in detail through computational experiments. Finally, we conclude this paper in Section VI.

II. Fuzzy Rule-Based Classifiers

Let us explain a fuzzy rule-based classifier for an M-class pattern classification problem in an n-dimensional pattern space $[0, 1]^n$ with m training patterns $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$, $p = 1, 2, \ldots, m$. We assume that each attribute value x_{pi} has already been normalized into a real number in the unit interval [0, 1] for $i = 1, 2, \ldots, n$ and $p = 1, 2, \ldots, m$. For our n-dimensional pattern classification problem, we use fuzzy rules of the following type [40]:

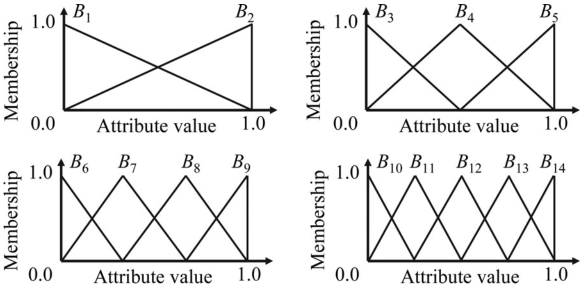

Fig. 1. Antecedent fuzzy sets "$B_1, B_2, \ldots, B_{14}$."

$$\text{Rule } R_q : \text{If } x_1 \text{ is } A_{q1} \text{ and } \ldots \text{ and } x_n \text{ is } A_{qn}$$
$$\text{then Class } C_q \text{ with } CF_q \qquad (1)$$

where R_q is the label of the qth fuzzy rule, A_{qi} is an antecedent fuzzy set $(i = 1, 2, \ldots, n)$, C_q is a class label, and CF_q is a rule weight (which is a real number in the unit interval [0, 1]). Since the early 1990s, fuzzy rules of this type have been frequently used in the literature [41]. For other types of fuzzy rules for pattern classification problems, see [41]–[43].

As antecedent fuzzy sets, we use 14 fuzzy sets "$B_1, B_2, \ldots, B_{14}$" in Fig. 1. We also use a special antecedent fuzzy set "B_0" defined by the unit interval [0, 1] to represent a "*don't care*" condition. The antecedent fuzzy set B_0 is always fully compatible with any normalized attribute values. The use of the "*don't care*" condition enables us to perform rule-level feature selection [44]. That is, each rule can have antecedent conditions on different attributes. The number of antecedent conditions (excluding "*don't care*") of a fuzzy rule is often referred to as its rule length. The total number of possible combinations of the antecedent fuzzy sets is 15^n for our n-dimensional pattern classification problem. The antecedent part of the fuzzy rule R_q in (1) is one of those combinations. The consequent class C_q and the rule weight CF_q are specified using compatible training patterns with its antecedent part in a heuristic manner [45].

The compatibility grade of each pattern $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$ with the antecedent part of the fuzzy rule R_q in (1) is calculated using the product operator as

$$\mu_{\mathbf{A}q}(\mathbf{x}_p) = \mu_{Aq1}(x_{p1}) \times \mu_{Aq2}(x_{p2}) \times \cdots \times \mu_{Aqn}(x_{pn}) \quad (2)$$

where $\mathbf{A}_q$ is the vector of the antecedent fuzzy sets of the fuzzy rule R_q (i.e., $\mathbf{A}_q = (A_{q1}, \ldots, A_{qn})$), and $\mu_{Aqi}(x_{pi})$ is the membership value of the antecedent fuzzy set A_{qi} at the input value x_{pi}.

To specify the consequent class C_q and the rule weight CF_q, we first calculate the confidence of the association from the antecedent fuzzy vector $\mathbf{A}_q$ to each class k $(k = 1, 2, \ldots, M)$ as

$$\text{Conf}(\mathbf{A}_q \Rightarrow \text{Class } k) = \frac{\sum_{\mathbf{x}_p \in \text{Class } k} \mu_{\mathbf{A}_q}(\mathbf{x}_p)}{\sum_{p=1}^{m} \mu_{\mathbf{A}_q}(\mathbf{x}_p)}. \qquad (3)$$

This is the confidence measure of the fuzzy association rule "$\mathbf{A}_q \Rightarrow$ Class k" [46], [47]. If the confidence value for a particular class C_q is larger than 0.5 in (3), we generate a fuzzy rule with the antecedent fuzzy vector $\mathbf{A}_q$ and the consequent class C_q. In this case, C_q is the majority class in the fuzzy pattern subspace $\mathbf{A}_q$. Then, the rule weight CF_q is specified as follows [45]:

$$CF_q = \mathrm{Conf}(\mathbf{A}_q \Rightarrow \mathrm{Class}\, C_q) - \sum_{\substack{k=1 \\ k \neq C_q}}^{M} \mathrm{Conf}(\mathbf{A}_q \Rightarrow \mathrm{Class}\, k). \tag{4}$$

Since the sum of the confidence value in (3) over all classes is 1, (4) can be rewritten as

$$CF_q = 2 \cdot \mathrm{Conf}(\mathbf{A}_q \Rightarrow \mathrm{Class}\, C_q) - 1. \tag{5}$$

Note that CF_q is always positive since we generate the fuzzy rule R_q only when the confidence value for the class C_q is larger than 0.5 in (3). If the confidence values are equal to or smaller than 0.5 for all classes, no fuzzy rules with the antecedent fuzzy vector $\mathbf{A}_q$ are generated.

Our fuzzy rule-based classifier is a set of fuzzy rules of the form in (1). Let us denote a set of fuzzy rules by S. When an input pattern $\mathbf{x}_p$ is presented to S, we use a single winner-based fuzzy reasoning method to classify $\mathbf{x}_p$ by S. A single winner rule R_W for $\mathbf{x}_p$ is chosen from S using the product of the compatibility grade in (2) and the rule weight of each fuzzy rule in S as follows:

$$\mu_{\mathbf{A}_W}(\mathbf{x}_p) \cdot CF_W = \max\{\mu_{\mathbf{A}_q}(\mathbf{x}_p) \cdot CF_q | R_q \in S\}. \tag{6}$$

The input pattern $\mathbf{x}_p$ is classified as the consequent class C_W of the winner rule R_W. When multiple fuzzy rules with different consequent classes have the same maximum value in (6), the classification of $\mathbf{x}_p$ is rejected. The classification of $\mathbf{x}_p$ is also rejected when no fuzzy rule in S is compatible with $\mathbf{x}_p$. For other fuzzy reasoning methods for pattern classification, see [41]–[43].

III. Hybrid Fuzzy Genetics-Based Machine Learning Algorithm

Our hybrid fuzzy GBML algorithm [37] has a Pittsburg-style framework where a rule set is handled as an individual. Its outline is shown in the left plot of Fig. 2. A Michigan-style algorithm is used as a kind of local search after genetic operations in the Pittsburgh-style framework.

As shown in the right plot of Fig. 2, the Michigan-style algorithm is probabilistically applied to each rule set. When it is applied, new rules are generated by genetic operations and a heuristic rule generation mechanism to partially modify the rule set. The Michigan-style algorithm is not iterated for each rule set. That is, new rule generation and population update are performed just once for each rule set when the Michigan-style algorithm is invoked (see the right plot of Fig. 2).

In this section, we explain a standard nonparallel nondistributed implementation of our hybrid fuzzy GBML algorithm. Its parallel distributed implementation is proposed in the next section.

Coding: A single fuzzy rule is represented by its n antecedent fuzzy sets as an integer string of length n. This integer coding is used in the Michigan part of our hybrid algorithm. In its Pittsburgh part, a set of N fuzzy rules is denoted by a concatenated integer string of length *nN*, where each substring of length n denotes a single fuzzy rule. The consequent class and the rule weight of each fuzzy rule are not coded (they are specified by the heuristic method in Section II).

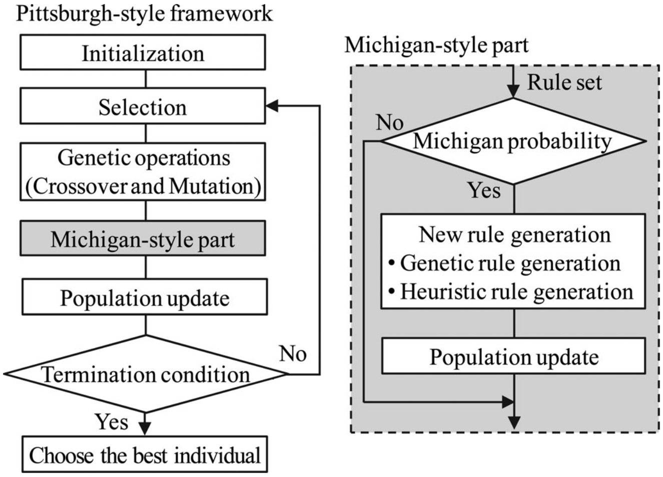

Fig. 2. Pittsburgh-style framework (left) of our hybrid algorithm and its Michigan-style part (right).

The consequent class and the rule weight are, however, stored together with the corresponding substring to avoid their unnecessary recalculation. The heuristic method is used to specify its consequent class and rule weight only when a new substring is generated. If there exists no majority class with a confidence value larger than 0.5, we tentatively assign an empty class and a zero rule weight to the substring. Such a substring, which can be viewed as a dummy rule, is removed when each string is evaluated in the Pittsburgh part. Our concatenated integer strings have variable string length because the number of fuzzy rules in fuzzy rule-based classifiers is not prespecified.

Initial Population: An initial string (i.e., initial rule set) is generated from a prespecified number of randomly selected training patterns. A substring of length n (i.e., an antecedent fuzzy vector) is specified from each training pattern. In our computational experiments, we randomly choose 30 patterns from available training patterns without replacement.

Let $\mathbf{x}_p = (x_{p1}, \ldots, x_{pn})$ be a selected training pattern. A substring of length n is specified from $\mathbf{x}_p$ in the following manner. First, we calculate the compatibility grade of each antecedent fuzzy set B_j in Fig. 1 with each attribute value x_{pi} of $\mathbf{x}_p$. Next, we choose one of the 14 antecedent fuzzy sets for each attribute x_i as A_{qi} with the following probability $(i = 1, 2, \ldots, n)$:

$$P(B_j) = \frac{\mu_{B_j}(x_{pi})}{\sum_{k=1}^{14} \mu_{B_k}(x_{pi})}, \quad j = 1, 2, \ldots, 14. \tag{7}$$

Then, each of the chosen antecedent fuzzy sets is replaced with B_0 with a prespecified "*don't care*" probability P_{DC}. In our computational experiment, P_{DC} is specified as $P_{\mathrm{DC}} = (n{-}5)/n$. This means that each substring has five antecedent conditions on average (except for "*don't care*"). We generate 30 substrings of length n from randomly selected 30 training patterns in this manner.

For each substring, the consequent class and the rule weight are specified by the heuristic method in Section II. If there exists no majority class with a confidence value larger than 0.5 for a substring, a dummy rule with an empty class and a zero rule weight is generated. In this manner, 30 fuzzy rules in an initial rule set S are generated. When all the generated 30 fuzzy rules are dummy rules, another set of 30 training patterns is randomly selected to generate 30 fuzzy rules. Let N_{pop} be the population size. By iterating the aforementioned procedure N_{pop} times, an initial population of N_{pop} rule sets is generated.

Fitness Evaluation in the Pittsburgh Part: Let S be a rule set (i.e., an individual) in the current population. After removing all dummy rules from S, its fitness value is calculated as

$$\text{fitness}(S) = w_1 f_1(S) + w_2 f_2(S) + w_3 f_3(S) \tag{8}$$

where w_1, w_2, and w_3 are nonnegative weights, and $f_1(S)$, $f_2(S)$, and $f_3(S)$ are as follows:

$f_1(S)$ error rate on training patterns in percentage;
$f_2(S)$ number of fuzzy rules in S;
$f_3(S)$ total rule length over fuzzy rules in S.

The weight values are specified as $w_1 = 100$, $w_2 = 1$, and $w_3 = 1$ in our computational experiments.

Parent Selection in the Pittsburgh Part: We use binary tournament selection with replacement to select N_{pop} pairs of parents from the current population.

Crossover in the Pittsburgh Part: Let S_1 and S_2 be a pair of selected parents. An offspring S is generated by randomly selecting fuzzy rules from each parent. The number of fuzzy rules to be selected from S_k is randomly and uniformly specified in the interval [1, $|S_k|$], where $|S_k|$ is the number of fuzzy rules in $S_k (k = 1, 2)$. The order of substrings in S is randomly updated. If the number of fuzzy rules in S becomes larger than a prespecified upper limit (60 in our computational experiments), it is decreased to the upper limit by randomly removing fuzzy rules from S. This crossover operation is applied to each pair of selected parents with a prespecified crossover probability (0.9 in our computational experiments). If the crossover operation is not applied, one parent is randomly chosen and handled as an offspring S, to which a mutation operation is applied.

Mutation in the Pittsburgh Part: A mutation operation is applied to each antecedent fuzzy set of the offspring S. Our mutation operation randomly replaces each antecedent fuzzy set (including "*don't care*") with a different one. This mutation operation is applied to each antecedent fuzzy set with a prespecified mutation probability (1/($n|S|$) in the Pittsburgh part in our computational experiments, where $n|S|$ is the string length of S). When the antecedent part of a fuzzy rule in S is mutated, its consequent class and rule weight are updated by the heuristic method using compatible training patterns.

Use of a Michigan-Style Algorithm: After the mutation operation in the Pittsburgh part, a Michigan-style rule generation procedure is applied to each rule set S with a prespecified probability (0.5 in our computational experiments). When the Michigan part is not applied to a rule set S, S is handled as a newly generated rule set (i.e., a new offspring) in the Pittsburgh part. Otherwise, the Michigan part is used to partially modify S, where all fuzzy rules in S are handled as the current population.

Fitness Evaluation in the Michigan Part: Training patterns are classified by the rule set S. The fitness value of each fuzzy rule R_q in S is defined by the number of correctly classified training patterns by R_q. Since we use the single winner-based fuzzy reasoning method in (6), each training pattern is classified by its single winner rule. Thus, we can count the number of correctly classified training patterns by each fuzzy rule in S. When multiple fuzzy rules with the same consequent class have the same maximum value in (6) for a training pattern, the first one among them in the rule set S is used as the winner rule. For example, if three fuzzy rules R_1, R_2, and R_3 in S are exactly the same, only the first rule R_1 among them has the possibility to be selected as a winner rule. The other two rules are never selected as a winner rule for any training patterns since R_1, R_2, and R_3 are the same. Thus, the fitness values of those duplicated rules R_2 and R_3 are always zero.

New Rule Generation in the Michigan Part: New fuzzy rules are generated from the existing ones in S. For generating a new fuzzy rule, a pair of parent fuzzy rules is first selected from S using binary tournament selection with replacement. Uniform crossover is applied to the selected pair with a prespecified crossover probability (0.9 in the Michigan part in our computational experiments). One of the generated offspring is randomly chosen as a new fuzzy rule. When the crossover is not applied, one parent is randomly chosen and handled as a new fuzzy rule. Then, the same mutation operation as in the Pittsburgh part is applied to each antecedent fuzzy set of the new fuzzy rule with a prespecified mutation probability (1/n in the Michigan part in our computational experiments, while it is 1/($n|S|$) in the Pittsburgh part).

New fuzzy rules are also generated from training patterns that are misclassified (including rejection). Let D_{MR} be the set of those training patterns (i.e., misclassified or rejected training patterns). Using the same heuristic method as in the initial population of the Pittsburgh part, a single fuzzy rule is generated from a pattern randomly selected without replacement from D_{MR}.

Population Update in the Michigan Part: The number of fuzzy rules to be generated for updating the current rule set S depends on the population size (i.e., $|S|$). In our computational experiments, only a single rule is generated when $|S| \leq 5$ (i.e., when S includes five or less fuzzy rules). We use one of the two rule generation mechanisms (i.e., genetic or heuristic rule generation) with the same probability. Of course, we always use the genetic rule generation when all training patterns are correctly classified by S (i.e., when D_{MR} is empty).

When $5 < |S| \leq 10$, two rules are generated: one by the genetic rule generation and the other by the heuristic rule generation. If D_{MR} is empty, two rules are generated by the genetic rule generation. In the case of $10 < |S| \leq 15$, three rules are generated. Two of them are generated in the same manner as in the case of $5 < |S| \leq 10$. One of the two mechanisms is randomly chosen to generate the other fuzzy rule. If D_{MR} is empty, the genetic rule generation is always used. When $15 < |S| \leq 20$, we generate four fuzzy rules: two by the genetic rule generation and the other two by the heuristic rule generation (when D_{MR}

includes two or more patterns). If D_{MR} includes only a single pattern, one fuzzy rule is generated from that pattern. The other three rules are generated by the genetic rule generation. In this manner, we generate k fuzzy rules when $5(k-1) < |S| \leq 5k$.

From the rule set S, the worst k fuzzy rules are removed when $5(k-1) < |S| \leq 5k$. Then, the newly generated k fuzzy rules are added to S. The updated rule set S is returned to the Pittsburgh part where S is handled as a newly generated rule set (i.e., a new offspring in the Pittsburgh part).

Population Update in the Pittsburgh Part: In each generation of the Pittsburgh part, we generate N_{pop} new rule sets by selection, crossover, mutation, and Michigan-style rule generation. The fitness of each new rule set is calculated. Then, we select the best N_{pop} rule sets as the next population from the N_{pop} rule sets in the current population and the newly generated N_{pop} rule sets. That is, we use the $(\mu+\lambda)$-ES population update mechanism with $\mu = \lambda = N_{\mathrm{pop}}$.

Our hybrid fuzzy GBML algorithm can be summarized as follows.

[Hybrid Fuzzy GBML Algorithm (Pittsburgh Part)]

Step 1: Generate an initial population of N_{pop} rule sets.

Step 2: Evaluate each rule set S in the current population.

Step 3: Generate N_{pop} rule sets by selection, crossover, and mutation.

Step 4: Apply the Michigan part to each new rule set with a prespecified probability.

Step 5: Construct the next population by choosing the best N_{pop} rule sets from the N_{pop} rule sets in the current population and the newly generated N_{pop} rule sets.

Step 6: If a prespecified termination condition is satisfied, terminate the execution of this algorithm. Otherwise, return to Step 3. In our computational experiments, we use the total number of generations as the termination condition.

[Michigan Part]

Step A: Let a new rule set in Step 4 of the Pittsburgh part be S, which is used as the current population in the Michigan part.

Step B: Classify training patterns by S. Then calculate the number of correctly classified training patterns by each fuzzy rule, which is used as the fitness value of each fuzzy rule.

Step C: Generate k fuzzy rules where k is an integer satisfying the inequality $5(k-1) < |S| \leq 5k$.

Step D: Remove the worst k fuzzy rules from S. Then add the newly generated k fuzzy rules to S.

Step E: Return the updated S to the Pittsburgh part where S is used as a newly generated rule set.

IV. Parallel Distributed Implementation

We propose a parallel distributed model in Fig. 3, where our model is explained for the case of seven parallel processors. The population is divided into multiple subpopulations of the same size. Each subpopulation is assigned to a different island with a single processor. The given training data are also divided into multiple disjoint subsets of the same size and the same class distribution (i.e., they are stratified into multiple strata as in a windowing method [31]). A different training data subset is assigned to each island. This is the main feature of our parallel distributed model, which is different from other parallel GBML models.

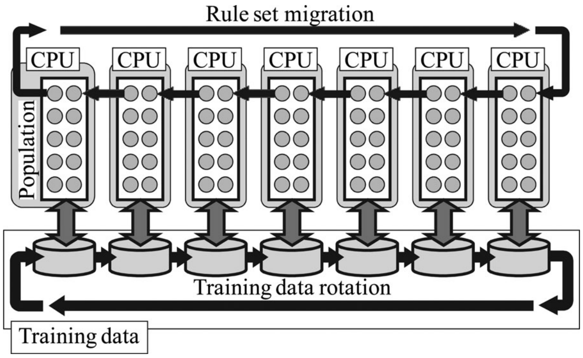

Fig. 3. Proposed parallel distributed model with the training data rotation and the rule set migration.

The subpopulation at each island is locally initialized using the assigned training data subset at the first generation. For a prespecified number of generations (e.g., 100 generations), our hybrid fuzzy GBML algorithm is executed at each island independently from the other islands.

The assigned training data subsets are periodically rotated over the islands (e.g., every 100 generations). The training data subset at the ith island is moved to the $(i+1)$th island. If the ith island is the last island, it is moved to the first island. After the training data rotation, our hybrid fuzzy GBML algorithm restarts from Step 2 where the fitness value of each string is evaluated using the newly assigned training data subset. In this step, we have the following two options with respect to the update of the consequent class and the rule weight of each fuzzy rule:

Option 1 (Local Update after Rotation): The consequent class and the rule weight of every fuzzy rule are updated at each island using the newly assigned training data subset.

Option 2 (No Update): They are not updated until the antecedent part is changed by mutation.

In this paper, we use the second option because better experimental results are obtained with less computation load. After the termination of our parallel distributed model, we also have similar two options as follows (we use the second one for the same reason):

Option 1 (Global Update after Termination): The consequent class and the rule weight of every fuzzy rule are updated using all training patterns after the termination of our model.

Option 2 (No Update): They are not updated.

We also use a migration operation where a copy of the best rule set at each island is periodically moved to another island. At the same time, the worst rule set at each island is removed. The best and worst rule sets are locally chosen using their error rates on the training data subset at each island. The worst rule set with the highest error rate is removed, and a copy of the best rule set with the lowest error rate is moved to another island. As we will show in the next section, when the training data rotation and the rule set migration are performed at the same generation in the same direction, they do not improve the search ability

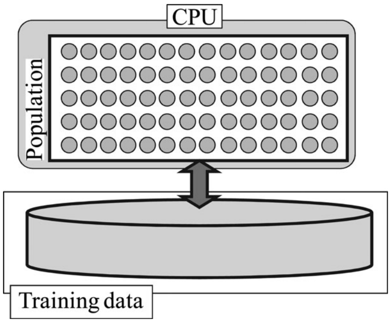

Fig. 4. Standard nonparallel nondistributed model.

of our parallel distributed model whereas their individual use improves its search ability. Thus, we propose an idea of the rule set migration in the opposite direction from the training data rotation. That is, a copy of the best rule set at the ith island is moved to the $(i-1)$th island while the training data subset at the ith island is moved to the $(i+1)$th island. In this manner, we can avoid their mutual interference that has negative effects on the search ability of our parallel distributed model.

After the termination of our parallel distributed model, all rule sets in the final population are examined using all training patterns. Before the fitness evaluation of each rule set, we remove from each rule set unnecessary fuzzy rules that are not used as the winner rule for any training patterns. The removal of those unnecessary fuzzy rules improves the second term (the number of fuzzy rules) and the third term (the total rule length) of our fitness function with no change of the first term (the error rate on training patterns). The best rule set with respect to the fitness function is selected from the final population. The selected rule set is used for performance evaluation.

Our parallel distributed model is compared with the standard nonparallel nondistributed implementation in Section III, which is illustrated in Fig. 4. The entire population is handled by a single CPU in Fig. 4, where each rule set is evaluated using all training patterns.

V. Computational Experiments

A. Algorithm Implementation

We performed computational experiments on a workstation with eight processors: Intel Xeon X5570 (4 core 2.93 GHz) × 2. In our parallel distributed model in Fig. 3, seven out of the eight processors were used for the execution of our hybrid fuzzy GBML algorithm. The other processor was used for other tasks such as the operation system. A population of size 210 was divided into seven subpopulations of size 30. Training patterns were also divided into seven subsets of the same size with the same class distribution. In the case of the standard nonparallel nondistributed model in Fig. 4, only a single processor was used. In both models, our hybrid fuzzy GBML algorithm was terminated after the 50 000th generation.

TABLE I
Nine Data Sets Used in This Paper

Name of Data Set	Number of Patterns	Number of Attributes	Number of Classes
Segment	2,310	19	7
Phoneme	5,404	5	2
Page-blocks	5,472	10	5
Texture	5,500	40	11
Satimage	6,435	36	6
Twonorm	7,400	20	2
Ring	7,400	20	2
PenBased	10,992	16	10
Magic	19,020	10	2

TABLE II
Results by the Standard Nonparallel Nondistributed Model in Fig. 4

Data Set	Training Error (%)	Test Error (%)	Number of Rules	Rule Length	Time (minutes)
Segment	3.73	5.99	21.93	4.27	203.66
Phoneme	13.53	15.43	19.97	4.32	439.18
Page-blocks	3.25	3.81	10.80	3.24	204.63
Texture	3.03	4.64	24.53	7.20	766.61
Satimage	13.69	15.54	17.77	5.21	658.89
Twonorm	4.51	7.36	22.90	4.67	856.58
Ring	4.38	6.73	31.27	2.78	1015.04
PenBased	1.95	3.07	32.67	5.64	1520.54
Magic	14.75	15.42	10.87	4.09	771.05

B. Test Problems

We used nine datasets in Table I, which were available from the UCI machine learning repository and the KEEL project webpage [8]. All attribute values were normalized into real numbers in the unit interval [0, 1] in our computational experiments.

C. Results of Nonparallel Nondistributed Model

First, we report experimental results by the standard nonparallel nondistributed model in Fig. 4. A population of 210 rule sets was evolved on a single processor for 50 000 generations. The tenfold cross-validation (10CV) procedure was iterated three times using different data partitions into ten subsets (i.e., 3 × 10CV). Average results over 30 runs in the 3 × 10CV are summarized in Table II. As shown in the last column of Table II, the average computation times were very long for some datasets (e.g., more than 25 h for the PenBased data). In the next section, experimental results of our parallel distributed model are compared with those in Table II.

D. Results by Our Parallel Distributed Model

In the same manner as Table II, experimental results by our parallel distributed model are summarized in Table III. The

TABLE III
RESULTS BY OUR PARALLEL DISTRIBUTED MODEL IN FIG. 3

Data Set	Training Error (%)	Test Error (%)	Number of Rules	Rule Length	Time (minutes)
Segment	4.24	5.90	14.83	5.19	4.69
Phoneme	14.70	15.96	14.27	3.98	13.19
Page-blocks	3.25	3.62	6.93	4.95	4.74
Texture	3.24	4.77	19.33	7.54	15.72
Satimage	11.43	12.96	11.80	8.86	15.38
Twonorm	2.83	3.39	3.17	9.34	7.84
Ring	4.11	5.25	16.70	4.43	22.52
PenBased	2.40	3.30	28.70	5.29	35.56
Magic	14.19	14.89	7.67	4.45	22.58

TABLE IV
RESULTS BY OUR PARALLEL DISTRIBUTED MODEL
(DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**0.51**	- 0.09	- 7.10	0.92	2.30
Phoneme	**1.17**	**0.53**	- 5.70	- 0.34	3.00
Page-blocks	0.00	**- 0.19**	- 3.87	1.71	2.32
Texture	0.21	0.13	- 5.20	0.34	2.05
Satimage	**- 2.26**	**- 2.58**	- 5.97	3.65	2.33
Twonorm	**- 1.68**	**- 3.97**	- 19.73	4.67	0.92
Ring	**- 0.27**	**- 1.48**	- 14.57	1.65	2.22
PenBased	**0.45**	0.23	- 3.97	- 0.35	2.34
Magic	**- 0.56**	**- 0.53**	- 3.20	0.36	2.93

rotation and migration intervals were specified as 100 generations. The average computation times were decreased from Table II by an order or two of magnitude.

To compare Table III with Table II in detail, we calculated their differences by subtracting the average results in Table II from the corresponding results in Table III. Only for the average computation times, we calculated the relative computation times of our parallel distributed model in percentage in comparison with the corresponding results in Table II. The calculated differences and the relative computation times are summarized in Table IV. A negative value in Table IV shows the decrease in the corresponding average result in Table III from Table II.

The fourth column of Table IV shows that the average number of fuzzy rules was decreased by our parallel distributed model for all nine datasets. In the fifth column, the average rule length was decreased only for two datasets (Phoneme and PenBased). These observations show that fuzzy rule-based classifiers with a smaller number of longer fuzzy rules were often obtained by our parallel distributed model. From the third column of Table IV, we can see that the average error rates on test data were decreased (i.e., improved) by our parallel distributed model for six datasets. The largest deterioration in the test data error rate was 0.53% for the Phoneme dataset, while the largest improvement was 3.97% for the Twonorm dataset. These observations show that our parallel distributed model did not severely deteriorate the generalization ability of obtained fuzzy rule-based classifiers for the nine datasets used in our computational experiments. An interesting observation in Table IV is that the improvement in the test data error rate was larger than that in the training data error rate for almost all datasets (except for the Magic dataset). A similar effect of training data rotation on the generalization ability was also reported for a windowing method [31].

Using the Wilcoxon signed-rank test (two-tailed, $\alpha = 0.05$), we examined the existence of statistically significant differences between the two models in error rates on training data in our computational experiments. Except for the Page-blocks and Texture data, error rates on training data had statistically significant differences between the two models. Those seven datasets are shown by using boldface for the training data error rates in Table IV. In the same manner, the Wilcoxon signed-rank test was also applied to error rates on test data. Statistically significant differences are found between the two models for six datasets. Those six datasets are shown by using boldface for the test data error rates in Table IV. These statistical tests show that different results were obtained by the two models for almost all datasets except for the Texture dataset.

From the last column of Table IV, we can see that the average computation times of our parallel distributed model in Table III were in the range between 0.9% and 3.0% of those of the standard nonparallel nondistributed model in Table II. In our parallel distributed model, both the population and the training data were divided into seven subsets. Thus, the computation load at a single CPU was decreased to 1/49 (i.e., 2.04%) if compared with the case of the standard nonparallel nondistributed model. In general, the speed-up by parallel computation is smaller than the decrease in computation load at a single CPU due to computational overhead. As a result, the relative computation times in the last column of Table IV were slightly larger than 2.04% for almost all datasets. However, the relative computation time for the Twonorm dataset (i.e., 0.92%) was much smaller than 2.04%. This speed-up can be explained by a large decrease in the average number of fuzzy rules (i.e., -19.73 in Table IV from 22.90 in Table II to 3.17 in Table III). For comparison, we also examined a simple master–slave model where the fitness evaluation of each rule set in the Pittsburgh part and all procedures in the Michigan part of the nonparallel nondistributed model in Fig. 4 were performed in parallel on seven processors. Since the training data were not divided, the speed-up by our master–slave model was much smaller than that by our parallel distributed model (e.g., the relative computation time of our master–slave model for the Ring dataset was 24.64% while that of our parallel distributed model was 2.22% in Table IV).

E. Recalculation of Consequent Class and Rule Weight

As explained in Section IV, we do not recalculate the consequent class and the rule weight of each fuzzy rule after the training data rotation in our parallel distributed model. Only when the antecedent part of a fuzzy rule is changed by mutation,

TABLE V
RESULTS WITH RECALCULATION AFTER TRAINING DATA ROTATION (DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**1.29**	0.17	- 6.66	1.32	2.42
Phoneme	**2.11**	**1.25**	- 3.70	- 0.41	3.25
Page-blocks	**0.44**	0.09	- 3.70	2.15	2.34
Texture	**0.70**	**0.43**	- 5.80	2.02	2.06
Satimage	**- 2.04**	**- 2.44**	- 7.00	8.82	2.14
Twonorm	**- 1.97**	**- 4.36**	- 19.90	5.59	0.82
Ring	- 0.02	**- 1.09**	- 13.90	1.03	2.41
PenBased	**0.68**	0.35	- 4.90	- 0.23	2.26
Magic	**- 0.35**	**- 0.58**	- 3.70	0.58	2.82

TABLE VI
RESULTS WITH GLOBAL RECALCULATION AFTER THE TERMINATION (DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**1.29**	0.09	- 6.50	1.27	2.46
Phoneme	**2.30**	**1.18**	- 4.70	- 0.46	3.26
Page-blocks	**0.49**	0.14	- 4.13	2.26	2.38
Texture	**0.66**	**0.42**	- 5.63	1.95	2.03
Satimage	**- 1.36**	**- 2.09**	- 6.00	3.67	2.35
Twonorm	**- 1.95**	**- 4.32**	- 20.00	5.52	0.82
Ring	- 0.01	**- 1.41**	-14.87	1.72	2.24
PenBased	**0.89**	**0.46**	- 4.10	- 0.35	2.33
Magic	- 0.13	**- 0.27**	- 3.14	0.41	2.87

are its consequent class and rule weight recalculated. For comparison, we implemented a variant of our parallel distributed model, where the consequent class and the rule weight of each fuzzy rule were recalculated at each island using the newly assigned training data subset after the training data rotation. Experimental results are summarized in Table V in the same manner as in Table IV (i.e., differences from Table II and the relative computation times). The recalculation after the training data rotation increased (i.e., deteriorated) the test data error rates for seven datasets (except for Twonorm and Magic).

In our parallel distributed model, the consequent class and the rule weight of each fuzzy rule are not recalculated after the termination, either. For comparison, we implemented a variant where the consequent class and the rule weight of each fuzzy rule were recalculated using all training patterns after the termination of our parallel distributed model. The best rule set was chosen from the final population after the recalculation. We summarize experimental results by this variant in Table VI. The recalculation after the termination increased (i.e., deteriorated) the average error rates on test data for eight datasets (except for Twonorm).

TABLE VII
RESULTS WITH NO TRAINING DATA ROTATION (DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**4.19**	**3.32**	- 3.60	- 0.75	2.19
Phoneme	**3.13**	**2.64**	1.53	- 0.34	2.68
Page-blocks	**0.82**	**0.69**	- 1.00	- 0.59	2.84
Texture	**4.73**	**4.39**	1.60	- 1.49	2.55
Satimage	**0.71**	0.08	3.23	- 0.09	2.68
Twonorm	**3.43**	**1.84**	1.87	- 0.73	2.99
Ring	**3.80**	**2.60**	- 2.84	0.14	2.53
PenBased	**4.43**	**4.11**	4.43	- 1.10	2.82
Magic	**0.94**	**0.93**	1.90	0.40	3.74

TABLE VIII
RESULTS WITH NO RULE SET MIGRATION (DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**3.06**	**1.98**	- 4.73	1.46	2.80
Phoneme	**2.28**	**1.27**	- 3.97	0.00	3.23
Page-blocks	0.13	- 0.09	- 3.63	2.49	2.34
Texture	**2.51**	**2.20**	- 8.36	8.98	1.99
Satimage	**- 0.18**	**- 0.78**	- 4.50	4.39	2.52
Twonorm	**- 1.72**	**- 4.28**	- 19.60	5.69	0.97
Ring	**0.29**	**- 1.46**	- 25.44	7.24	1.68
PenBased	**1.69**	**1.35**	- 8.04	0.90	2.44
Magic	0.10	- 0.09	- 3.80	1.21	3.06

F. Training Data Rotation and Rule Set Migration

To examine the effects of the training data rotation and the rule set migration on the search ability of our parallel distributed model, we implemented the following three variants.

1) No training data rotation variant where each island continues to use the same training data subset. The rule set migration is used. Experimental results are summarized in Table VII.
2) No rule set migration variant where we do not perform any migration operation. The training data rotation is used. Experimental results are summarized in Table VIII.
3) Synchronized rotation–migration variant where the rotation and the migration are performed in the same direction. Experimental results are summarized in Table IX.

Experimental results in Tables VII–IX are shown in the same manner as in Table IV (i.e., differences from Table II and the relative computation times). The existence of statistically significant differences in error rates from the standard nonparallel nondistributed model in Table II is also shown by boldface in the same manner as in Table IV.

The no training data rotation variant in Table VII increased the average error rates on training and test data for all datasets from Table II (a positive value in Table VII shows the increase

TABLE IX
RESULTS WITH ROTATION AND MIGRATION IN THE SAME DIRECTION
(DIFFERENCES FROM TABLE II)

Data Set	Difference in Training Error (%)	Difference in Test Error (%)	Difference in Number of Rules	Difference in Rule Length	Relative Computation Time (%)
Segment	**3.65**	**2.93**	- 2.70	- 0.20	2.73
Phoneme	**2.75**	**2.14**	5.80	- 0.28	3.78
Page-blocks	**0.64**	**0.44**	0.17	- 0.56	3.49
Texture	**3.54**	**2.73**	4.00	- 1.25	3.04
Satimage	0.26	- 0.49	7.10	- 0.04	3.45
Twonorm	**3.31**	**1.95**	8.03	- 0.68	3.84
Ring	**3.46**	**2.06**	2.46	0.28	3.29
PenBased	**3.69**	**3.26**	11.86	- 0.65	3.32
Magic	**0.57**	**0.52**	6.60	0.43	5.43

from Table II). This observation supports the importance of the training data rotation.

Experimental results in Table VIII by the no rule set migration variant are much better than those in Table VII by the no training data rotation variant. This observation shows that the training data rotation is more important than the rule set migration. The comparison between Tables IV and VIII shows that lower average error rates on test data were obtained for eight datasets (except for Twonorm) by our parallel distributed model than the no rule set migration variant. This observation shows the importance of using both the training data rotation and the rule set migration.

In Table IX, good results were not obtained by the synchronized rotation–migration variant. This variant increased the average error rates on test data from Table II of the nonparallel nondistributed model for eight datasets (except for Satimage). The comparison between Table VIII (no rule set migration) and Table IX (synchronized rotation and migration) shows that lower average error rates on test data were obtained for all nine datasets by the no rule set migration variant. This observation shows negative effects of the mutual interference between the training data rotation and the rule set migration on the search ability of our parallel distributed model.

To further examine the effect of the training data rotation and the rule set migration, we performed additional computational experiments using various specifications of the rotation interval T_{R} and the migration interval T_{M}. More specifically, we examined 6 × 6 combinations of T_{R} and T_{M} with $T_{\mathrm{R}} = 20, 50, 100, 200, 500, \infty$ and $T_{\mathrm{M}} = 20, 50, 100, 200, 500, \infty$, where $T_{\mathrm{R}} = \infty$ and $T_{\mathrm{M}} = \infty$ mean no rotation and no migration, respectively.

Average error rates on test data of the Ring data are summarized in Fig. 5 for the case of the training data rotation and the rule set migration in the same direction. Fig. 6 shows the corresponding results for the case of the opposite directions. In the same manner, we show experimental results on the Satimage data in Figs. 7 and 8. In Figs. 5–8, good results were not obtained with no training data rotation (i.e., $T_{\mathrm{R}} = \infty$). This observation is consistent with Table VII with no training data rotation. The necessity of the rule set migration is not clear in Figs. 5 and 6, where good results were obtained even when $T_{\mathrm{M}} = \infty$. However, its necessity is clear in Fig. 8, where experimental results with $T_{\mathrm{M}} = \infty$ were improved by the rule set migration (e.g., $T_{\mathrm{M}} = 500$).

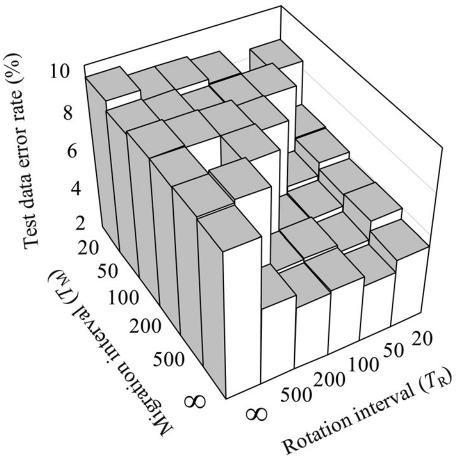

Fig. 5. Test data error rates with rotation and migration in the same direction (Ring data).

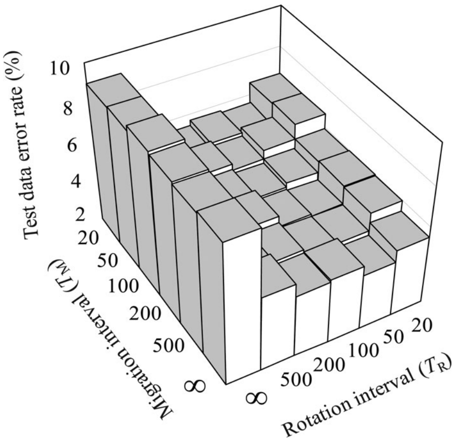

Fig. 6. Test data error rates with rotation and migration in opposite directions (Ring data).

In Figs. 5 and 7, good results were not obtained when $T_{\mathrm{R}} = aT_{\mathrm{M}}$ holds for an integer a [i.e., when $(T_{\mathrm{R}}, T_{\mathrm{M}})$ = (500, 20), (500, 50), (500, 100), (500, 500), . . . , (50, 50), (20, 20)]. Under these parameter settings, no training data rotation was performed without the rule set migration. That is, the training data rotation was always performed together with the rule set migration in Figs. 5 and 7. Such a synchronization of the training data rotation with the rule set migration in the same direction clearly degraded the performance of our parallel distributed model in Figs. 5 and 7.

In Figs. 5 and 7, good results were obtained from the two combinations (500, 200) and (50, 20) of $(T_{\mathrm{R}}, T_{\mathrm{M}})$ when $T_{\mathrm{R}} > T_{\mathrm{M}}$. Under these combinations, the training data rotation was performed with and without the rule set migration. When $(T_{\mathrm{R}}, T_{\mathrm{M}})$ = (500, 200), the first training data rotation after the 500th generation was performed without the rule set migration, while the next rotation after the 1000th generation was performed with

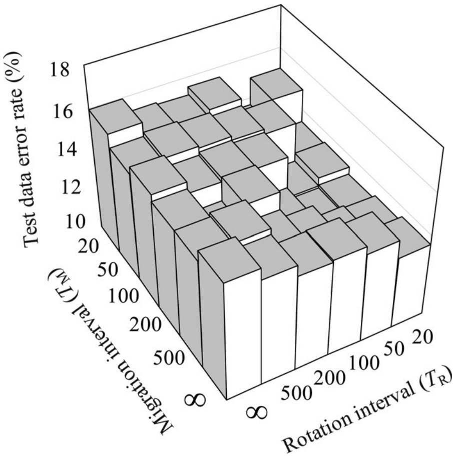

Fig. 7. Test data error rates with rotation and migration in the same direction (Satimage data).

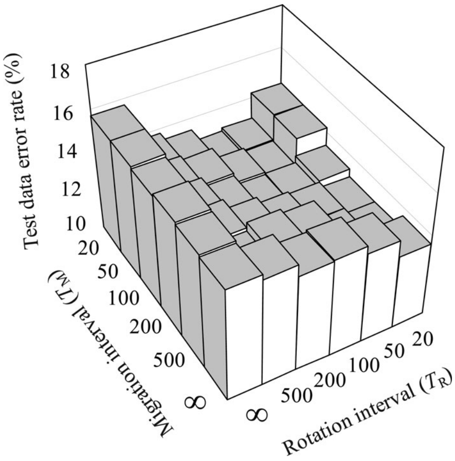

Fig. 8. Test data error rates with rotation and migration in opposite directions (Satimage data).

the migration. Good results from these combinations suggest that the training data rotation without the rule set migration has a positive effect on the search ability of our parallel distributed model. This is also suggested by good results in Figs. 5 and 7 when the rotation interval T_{R} was shorter than the migration interval T_{M} [i.e., when $T_{\mathrm{R}} < T_{\mathrm{M}}$ (the training data rotation was more frequent than the rule set migration)].

Even in Figs. 5 and 7 with the training data rotation and the rule set migration in the same direction, good results were obtained when the rotation and migration intervals were appropriately specified. The point is to avoid the following situation: The training data rotation is always performed together with the rule set migration in the same direction. In Figs. 6 and 8, we obtained good results from a wide range of the rotation interval T_{R} and the migration interval T_{M}. This is because the rotation and the migration were performed in opposite directions.

G. Comparison With Other Reported Results

In this section, we compare average error rates on test data by our parallel distributed model with other reported results by fuzzy and nonfuzzy GBML algorithms in the literature. Since computational experiments were performed in different settings in each study, our performance comparison in this subsection is not rigorous. Our intention is not to compare different GBML algorithms with each other, but to show that our experimental results are at least comparable with other reported results in the literature.

TABLE X
REPORTED TEST DATA ERROR RATES IN THE LITERATURE

Data Set	Our Parallel model	GP-COACH [48]	IVFS-Amp [49]	IVFS-Coop [50]	FARC-HD [51]	ILAS [31]	Bio HEL [52]
Segment	5.90	24.04	-	-	-	-	**2.90**
Phoneme	**15.96**	-	-	-	17.86	-	-
Page-blocks	**3.62**	8.77	5.84	6.57	4.99	-	-
Texture	**4.77**	-	-	-	7.11	-	-
Satimage	12.96	27.50	-	-	12.68	20.10	**11.60**
Twonorm	**3.39**	15.17	-	-	4.72	-	-
Ring	**5.25**	-	16.89	12.57	5.92	-	-
PenBased	**3.30**	17.80	21.73	17.00	3.96	20.10	6.00
Magic	**14.89**	20.18	20.82	19.82	15.49	-	-

In Table X, we compare our parallel distributed model with recently proposed four fuzzy GBML algorithms (GP-COACH [48], IVFS-Amp [49], IVFS-Coop [50], and FARC-HD [51]) and two nonfuzzy GBML algorithms (ILAS [31] and BioHEL [52]). GP-COACH [48] is a Michigan-style algorithm. All three of the other fuzzy GBML algorithms are Pittsburgh-style algorithms. IVFS-Amp [49] and IVFS-Coop [50] use interval-valued membership functions. The width of each membership function is tuned in IVFS-Amp [49], while both the location and the width are tuned in IVFS-Coop [50]. Standard membership functions are used in FARC-HD [51] where rule extraction, rule selection, and lateral tuning of membership functions are performed. ILAS [31] and BioHEL [52] are nonfuzzy GBML algorithms. ILAS [31] is based on Pittsburgh approach with a training data rotation method called "windowing" while BioHEL [52] is based on IRL approach.

The lowest error rate for each dataset is highlighted by boldface in Table X. Good results were obtained by our parallel distributed model in comparison with the other algorithms in Table X.

H. Search Behavior of Our Parallel Distributed Model

As shown in Table IV, our parallel distributed model improved both the training data and test data error rates on some datasets in comparison with the standard nonparallel nondistributed model. In this section, we examine why our parallel distributed model can improve the performance of our hybrid fuzzy GBML algorithm using experimental results on the Satimage dataset with the largest improvement in the training data error rate in Table IV (i.e., -2.26%).

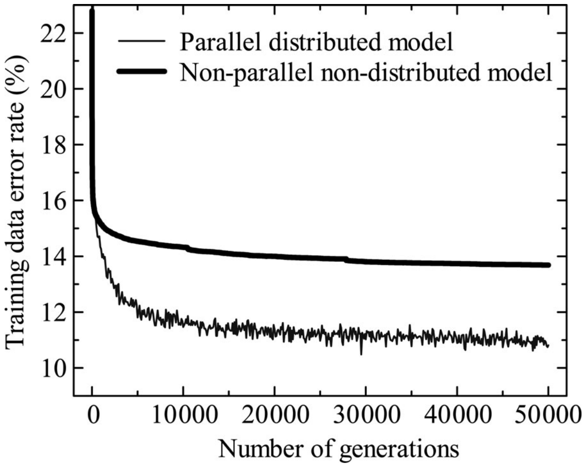

Fig. 9. Average error rates on all training patterns (Satimage data).

In Fig. 9, we compare training data error rates between our parallel distributed model and the standard nonparallel nondistributed model. Fig. 9 shows the average error rate of the best rule set at each generation over 30 runs in the 3 × 10CV. The best rule set was selected at each generation in each run using all training patterns, even in our parallel distributed model (i.e., the best rule set was selected at each generation in the same manner as in the final generation). Since the learning of rule sets was performed using periodically rotated training data subsets in our parallel distributed model, its experimental results in Fig. 9 have a number of ups and downs.

Our parallel distributed model in Fig. 9 continued to decrease the training data error rate after the standard nonparallel nondistributed model was slowed down. This may be because our parallel distributed model can help the hybrid fuzzy GBML algorithm to escape from local minima through diversity maintenance by the training data rotation. The use of an island model for parallel implementation also has a positive effect on the diversity of solutions.

To further examine the diversity maintenance ability of our parallel distributed model, we monitored 30 rule sets in the first island in a single run of our parallel distributed model. In Fig. 10, we show the maximum and minimum error rates on the assigned training data subset among the 30 rule sets in the first island for 500 generations after the 30 000th generation. The error rate of each rule set was calculated for the assigned training data subset, which was rotated every 100 generations. In Fig. 10, we can observe clear periodical ups and downs of the error rates caused by the training data rotation every 100 generations. The difference between the maximum and minimum error rates in Fig. 10 shows that the 30 rule sets in the first island had some diversity.

For comparison, Fig. 10 also shows the corresponding results by a single run of the standard nonparallel nondistributed model. The maximum and minimum error rates were calculated among 210 solutions in the whole population using all training patterns. The maximum and minimum error rates were the same over 500 generations in Fig. 10. This means that the 30 rule sets in the first island of our parallel distributed model had a larger diversity

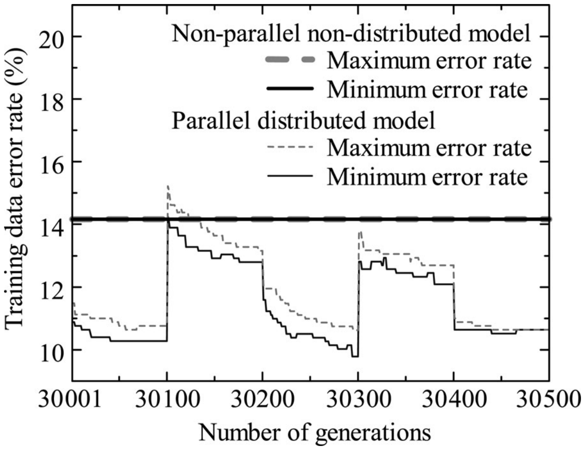

Fig. 10. The maximum and minimum error rates among 30 rule sets at the first island of our parallel distributed model, and those among 210 rule sets in the population of the standard implementation (Satimage data).

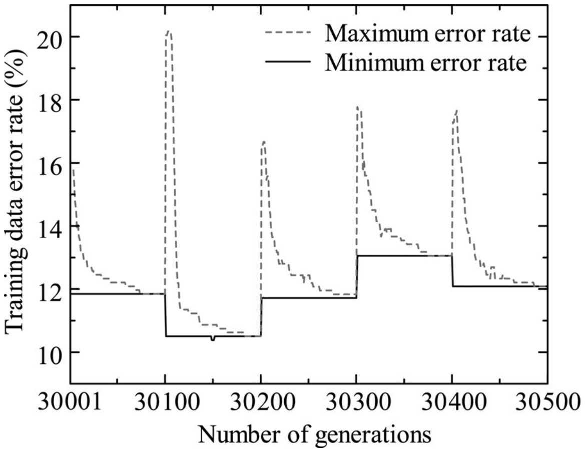

Fig. 11. Maximum and minimum error rates among 30 solutions at the first island of our parallel distributed model with the synchronized rotation and migration in the same direction (Satimage data).

than the 210 rule sets in the whole population of the standard nonparallel nondistributed model in Fig. 10.

In Fig. 11, we show experimental results by our parallel distributed model with the training data rotation and the rule set migration in the same direction (i.e., the synchronized rotation-migration variant). The maximum and minimum error rates on the training data subset were calculated at each generation among 30 rule sets in the first island in the same manner as in Fig. 10. The training data subset was rotated from the seventh island to the first one together with a copy of the best rule set in the seventh island every 100 generations in Fig. 11. The search behavior of our parallel distributed model is totally different between Figs. 10 and 11. Computational experiments in these two figures were performed under the same setting except for the migration direction. In Fig. 11, the training data error rates of the 30 rule sets in the first island converged to the best one during 100 generations before each training data rotation (except for the 30 300th generation). Large increases of the maximum error rate in Fig. 11 suggest that this convergence

TABLE XI
DIFFERENCE BETWEEN THE AVERAGE TRAINING AND TEST ERROR RATES

Data Set	Standard Non-Parallel	Our Parallel Distributed	No Rotation	No Migration	Same Direction
Segment	2.26*	1.66	1.39	1.18	1.54
Phoneme	1.90*	1.26	1.41	0.89	1.29
Page-blocks	0.56*	0.37	0.43	0.34	0.36
Texture	1.61*	1.53	1.27	1.30	0.80
Satimage	1.85*	1.53	1.22	1.25	1.10
Twonorm	2.85*	0.56	1.26	0.29	1.49
Ring	2.35*	1.14	1.15	0.60	0.95
PenBased	1.12*	0.90	0.80	0.78	0.69
Magic	0.67	0.70*	0.66	0.48	0.62

TABLE XII
AVERAGE TRAINING DATA ERROR RATE OVER THE BEST RULE SET AT EACH ISLAND. THE ERROR RATE OF EACH RULE SET WAS CALCULATED ON THE ASSIGNED TRAINING DATA SUBSET

Data Set	Standard Non-Parallel	Our Parallel Distributed	No Rotation	No Migration	Same Direction
Segment	3.73	3.17	**1.86**	4.83	**1.52**
Phoneme	13.53	13.13	**10.61**	13.77	**9.55**
Page-blocks	3.25	2.82	**2.61**	3.20	**2.38**
Texture	**3.03**	3.16	3.04	6.32	**2.05**
Satimage	13.69	**10.91**	11.11	13.36	**9.72**
Twonorm	4.51	2.24	**1.77**	2.42	**1.32**
Ring	4.38	3.67	**2.55**	4.23	**1.96**
PenBased	**1.95**	2.36	2.38	4.17	**1.58**
Magic	14.75	14.00	**13.86**	14.69	**12.93**

caused the overfitting of the 30 rule sets to the assigned training data subset.

In our parallel distributed model, seven training data subsets are rotated over seven islands every 100 generations. Thus, the cycle of the rotation is 700 generations. That is, the environment of each island is periodically changed every 100 generations with the cycle of 700 generations. Thus, each island can be viewed as having a dynamic optimization problem. In other words, our parallel distributed model can be viewed as solving a static pattern classification task by generating a dynamic optimization problem at each island. However, we do not use any dynamic optimization techniques in our parallel distributed model (for evolutionary dynamic optimization, see [53]–[55]). This is because our goal is not the fast adaptation to the changing environment at each island but the design of fuzzy rule-based classifiers with high generalization ability.

In order to examine positive effects of our parallel distributed model on the generalization ability of obtained fuzzy rule-based classifiers, we calculated the difference between the training data and test data error rates on each dataset for the following five implementations of our hybrid fuzzy GBML algorithm: the standard nonparallel nondistributed model, our parallel distributed model, and its three variants (i.e., no training data rotation, no rule set migration, and synchronized rotation–migration). The calculated difference is summarized in Table XI. The largest difference for each dataset is shown by "∗."

From Table XI, we can see that the difference between the average training and test data error rates was small in our parallel distributed model and its variants. This is because only 1/7 of training patterns were used for fitness evaluation. The other 6/7 of training patterns played a role of validation data when a final rule set is chosen.

We also calculated the average training data error rate of the best rule set at each island in the final generation over 30 runs in the 3×10CV. The error rate and the fitness value of each rule set were locally calculated at each island using the assigned training data subset in the final generation. The calculated average error rate is summarized in Table XII. For comparison, we also show the corresponding experimental results by the standard nonparallel nondistributed model where all training patterns were used for the error rate calculation and the fitness evaluation.

In Table XII, the lowest and second lowest error rates for each dataset are shown in boldface. For all nine datasets in Table XII, the lowest error rates on the assigned training data subsets were obtained by the synchronized rotation–migration variant. That is, the best fitting of rule sets to the assigned training data subset at each island was achieved by this variant. However, high generalization ability was not obtained from this variant as we have already shown in this paper. For some datasets in Table XII, the second lowest error rates on the assigned training data subsets were obtained by the no rotation variant. However, good results were not obtained for test data from this variant, either. These observations suggest that good fitting of rule sets in each island to the assigned training data subset does not mean high generalization ability.

VI. CONCLUSION

We have proposed a parallel distributed model of our hybrid fuzzy GBML algorithm. The following observations were obtained from computational experiments using the proposed model.

1) The proposed model drastically decreased the computation time of our hybrid fuzzy GBML algorithm. When we used seven processors for parallel computation, the computation time of the proposed model was in the range from 0.92% to 3.00% of that of the standard nonparallel nondistributed model. The proposed model with such a drastic speed-up did not severely degrade the generalization ability of obtained fuzzy rule-based classifiers. The largest increase (i.e., deterioration) in the average error rate on test data was 0.53% in our computational experiments on nine datasets. Actually, the average error rate on test data was decreased (i.e., improved) by the proposed model for six out of the nine datasets.
2) The search ability of the proposed model was severely degraded by removing the training data rotation. The rule set migration also had a positive effect on the search ability,

while its effect is much smaller than that of the training data rotation. The best results were obtained by the proposed model with both the training data rotation and the rule set migration. The search ability of the proposed model, however, was severely degraded when the training data rotation and the rule set migration were performed in the same direction. Such a negative effect was removed by performing them in opposite directions.

The basic idea of the proposed parallel distributed model is applicable to other fuzzy and nonfuzzy Pittsburgh-style GBML algorithms (e.g., IVFS-Amp [49], IVFS-Coop [50], and FARC-HD [51]). This is because their implementation as an island model is usually easy. Parallel distributed implementation of other Pittsburgh-style GBML algorithms is a promising future research topic. Parallel distributed implementation of multiobjective Pittsburgh-style GBML algorithms is also a future research topic, which will be much more difficult than single-objective implementation. Recently various population-based approaches have been proposed for the design of Type-1 and Type-2 fuzzy rule-based systems [56]–[60]. Those approaches can be viewed as special types of fuzzy GBML algorithms. Their parallel distributed implementation is also a future research issue.

REFERENCES

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

[2] V. Nissen and J. Propach, "On the robustness of population-based versus point-based optimization in the presence of noise," *IEEE Trans. Evol. Comput.*, vol. 2, no. 3, pp. 107–119, Sep. 1998.

[3] M. M. Ali and A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Comput. Oper. Res.*, vol. 31, no. 10, pp. 1703–1725, Sep. 2004.

[4] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments: A survey," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, Jun. 2005.

[5] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 10, no. 4, pp. 459–472, 2006.

[6] L. B. Booker, D. E. Goldberg, and J. H. Holland, "Classifier systems and genetic algorithms," *Artif. Intell.*, vol. 40, no. 1–3, pp. 235–282, Sep. 1989.

[7] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Berlin, Germany: Springer-Verlag, 2002.

[8] J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera, "KEEL: A software tool to assess evolutionary algorithms for data mining problems," *Soft Comput.*, vol. 13, no. 3, pp. 307–318, Feb. 2009.

[9] S. García, A. Fernández, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability," *Soft Comput.*, vol. 13, no. 10, pp. 959–977, Aug. 2009.

[10] J. Bacardit and N. Krasnogor, "Performance and efficiency of memetic Pittsburgh learning classifier systems," *Evol. Comput.*, vol. 17, no. 3, pp. 307–342, Fall 2009.

[11] A. Fernández, S. García, J. Luengo, E. Bernadó-Mansilla, and F. Herrera, "Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 913–941, Dec. 2010.

[12] Y. Jin, Ed., *Multi-Objective Machine Learning*. Berlin, Germany: Springer-Verlag, 2006.

[13] H. Ishibuchi and Y. Nojima, "Analysis of interpretability-accuracy tradeoff of fuzzy systems by multiobjective fuzzy genetics-based machine learning," *Int. J. Approx. Reason.*, vol. 44, no. 1, pp. 4–31, Jan. 2007.

[14] H. Ishibuchi, "Multiobjective genetic fuzzy systems: Review and future research directions," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, London, U.K., Jul. 23–26, 2007, pp. 913–918.

[15] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Trans. Syst., Man Cybern C Appl. Rev.*, vol. 38, no. 3, pp. 397–415, May 2008.

[16] P. Ducange, B. Lazzerini, and F. Marcelloni, "Multi-objective genetic fuzzy classifiers for imbalanced and cost-sensitive datasets," *Soft Comput.*, vol. 14, no. 7, pp. 713–728, May 2010.

[17] C. J. Carmona, P. Gonzalez, M. J. del Jesus, and F. Herrera, "NMEEF-SD: Non-dominated multiobjective evolutionary algorithm for extracting fuzzy rules in subgroup discovery," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 5, pp. 958–970, Oct. 2010.

[18] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, Oct. 2002.

[19] S. Cahon, N. Melab, and E. G. Talbi, "ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics," *J. Heuristics*, vol. 10, no. 3, pp. 357–380, May 2004.

[20] N. Nedjah, E. Alba, and L. de Macedo Mourelle, *Parallel Evolutionary Computations*. Berlin, Germany: Springer-Verlag, 2006.

[21] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications*. Berlin, Germany: Springer-Verlag, 2011.

[22] O. Maitre, L. A. Baumes, N. Lachiche, A. Corma, and P. Collet, "Coarse grain parallelization of evolutionary algorithms on GPGPU cards with EASEA," in *Proc. Genetic Evol. Comput. Conf.*, Montreal, QC, Canada, Jul. 8–12, 2009, pp. 1403–1410.

[23] P. Vidal and E. Alba, "A multi-GPU implementation of a cellular genetic algorithm," in *Proc. IEEE Congr. Evol. Comput.*, Barcelona, Spain, Jul. 18–23, 2010, pp. 396–402.

[24] S. Tsutsui and N. Fujimoto, "An analytical study of GPU computation for solving QAPs by parallel evolutionary computation with independent run," in *Proc. IEEE Congr. Evol. Comput.*, Barcelona, Spain, Jul. 18–23, 2010, pp. 889–896.

[25] M. Rouhipour, P. J. Bentley, and H. Shayani, "Fast bio-inspired computation using a GPU-based systemic computer," *Parallel Comput.*, vol. 36, no. 10–11, pp. 591–617, Oct./Nov. 2010.

[26] W. B. Langdon, "Graphics processing units and genetic programming: An overview," *Soft Comput.*, vol. 15, no. 8, pp. 1657–1669, Aug. 2011.

[27] H. Liu and H. Motoda, "On issues of instance selection," *Data Min. Knowl. Discovery*, vol. 6, no. 2, pp. 115–130, Apr. 2002.

[28] J. R. Cano, F. Herrera, and M. Lozano, "On the combination of evolutionary algorithms and stratified strategies for training set selection in data mining," *Appl. Soft Comput.*, vol. 6, no. 3, pp. 323–332, Mar. 2006.

[29] J. R. Cano, F. Herrera, and M. Lozano, "Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability," *Data Knowl. Eng.*, vol. 60, no. 1, pp. 90–108, Jan. 2007.

[30] J. R. Cano, F. Herrera, and M. Lozano, "Stratification for scaling up evolutionary prototype selection," *Pattern Recognit. Lett.*, vol. 26, no. 7, pp. 953–963, May 2005.

[31] J. Bacardit, D. E. Goldberg, M. V. Butz, X. Llora, and J. M. Garrell, "Speeding-up Pittsburgh learning classifier systems: Modeling time and accuracy," in *Proc. 8th Int. Conf. Parallel Probl. Solving Nat.*, Birmingham, U.K., Sep. 18–22, 2004, pp. 1021–1031.

[32] M. A. Franco, N. Krasnogor, and J. Bacardit, "Speeding up the evaluation of evolutionary learning systems using GPGPUs," in *Proc. Genetic Evol. Computat. Conf.*, Portland, OR, Jul. 7–11, 2010, pp. 1039–1046.

[33] Y. Nojima, H. Ishibuchi, and I. Kuwajima, "Parallel distributed genetic fuzzy rule selection," *Soft Comput.*, vol. 13, no. 5, pp. 511–519, Mar. 2009.

[34] Y. Nojima, H. Ishibuchi, and S. Mihara, "Use of very small training data subsets in parallel distributed genetic fuzzy rule selection," in *Proc. 4th Int. Workshop Genetic Evol. Fuzzy Syst.*, Mieres, Spain, Mar. 17–19, 2010, pp. 27–32.

[35] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 3, pp. 260–270, Aug. 1995.

[36] H. Ishibuchi and T. Yamamoto, "Fuzzy rule selection by multi-objective genetic local search algorithms and rule evaluation measures in data mining," *Fuzzy Sets Syst.*, vol. 141, no. 1, pp. 59–88, Jan. 2004.

[37] H. Ishibuchi, T. Yamamoto, and T. Nakashima, "Hybridization of fuzzy GBML approaches for pattern classification problems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 2, pp. 359–365, Apr. 2005.

[38] Y. Nojima, S. Mihara, and H. Ishibuchi, "Parallel distributed implementation of genetics-based machine learning for fuzzy classifier design," in *Proc. 8th Int. Conf. Simulated Evol. Learn.*, Kanpur, India, Dec. 1–4, 2010, pp. 309–318.

[39] H. Ishibuchi, S. Mihara, and Y. Nojima, “Training data subdivision and periodical rotation in hybrid fuzzy genetics-based machine learning,” in *Proc. Int. Conf. Mach. Learn. Appl.*, Honolulu, HI, Dec. 18–21, 2011, pp. 229–234.

[40] H. Ishibuchi, K. Nozaki, and H. Tanaka, “Distributed representation of fuzzy rules and its application to pattern classification,” *Fuzzy Sets Syst.*, vol. 52, no. 1, pp. 21–32, Nov. 1992.

[41] H. Ishibuchi, T. Nakashima, and M. Nii, *Classification and Modeling With Linguistic Information Granules: Advanced Approaches to Linguistic Data Mining*. Berlin, Germany: Springer-Verlag, 2004.

[42] O. Cordón, M. J. del Jesus, and F. Herrera, “A proposal on reasoning methods in fuzzy rule-based classification systems,” *Int. J. Approx. Reason.*, vol. 20, no. 1, pp. 21–45, Jan. 1999.

[43] H. Ishibuchi, T. Nakashima, and T. Morisawa, “Voting in fuzzy rule-based systems for pattern classification problems,” *Fuzzy Sets Syst.*, vol. 103, no. 2, pp. 223–238, Apr. 1999.

[44] H. Ishibuchi, T. Nakashima, and T. Murata, “Performance evaluation of fuzzy classifier systems for multi-dimensional pattern classification problems,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 5, pp. 601–618, Oct. 1999.

[45] H. Ishibuchi and T. Yamamoto, “Rule weight specification in fuzzy rule-based classification systems,” *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 4, pp. 428–435, Aug. 2005.

[46] T. P. Hong, C. S. Kuo, and S. C. Chi, “Trade-off between computation time and number of rules for fuzzy mining from quantitative data,” *Int. J. Uncertain., Fuzziness Knowl.-Based Syst.*, vol. 9, no. 5, pp. 587–604, Oct. 2001.

[47] H. Ishibuchi, T. Yamamoto, and T. Nakashima, “Fuzzy data mining: Effect of fuzzy discretization,” in *Proc. IEEE Int. Conf. Data Min.*, San Jose, CA, Nov. 29/Dec. 2, 2001, pp. 241–248.

[48] F. J. Berlanga, A. J. Rivera, M. J. del Jesus, and F. Herrera, “GP-COACH: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems,” *Inf. Sci.*, vol. 180, no. 8, pp. 1183–1200, Apr. 2010.

[49] J. A. Sanz, A. Fernández, H. Bustince, and F. Herrera, “Improving the performance of fuzzy rule-based classification systems with interval-valued fuzzy sets and genetic amplitude tuning,” *Inf. Sci.*, vol. 180, no. 19, pp. 3674–3685, Oct. 2010.

[50] J. Sanz, A. Fernández, H. Bustince, and F. Herrera, “A genetic tuning to improve the performance of fuzzy rule-based classification systems with interval-valued fuzzy sets: Degree of ignorance and lateral position,” *Int. J. Approx. Reason.*, vol. 52, no. 6, pp. 751–766, Sep. 2011.

[51] J. Alcalá-Fdez, R. Alcalá, and F. Herrera, “A fuzzy association rule-based classification model for high-dimensional problems with genetic rule selection and lateral tuning,” *IEEE Trans. Fuzzy Syst.*, vol. 19, no. 5, pp. 857–872, Oct. 2011.

[52] J. Bacardit, E. K. Burke, and N. Krasnogor, “Improving the scalability of rule-based evolutionary learning,” *Memetic Comput. J.*, vol. 1, no. 1, pp. 55–67, Mar. 2009.

[53] S. Yang and X. Yao, “Experimental study on population-based incremental learning algorithms for dynamic optimization problems,” *Soft Comput.*, vol. 9, no. 11, pp. 815–834, Nov. 2005.

[54] C. Cruz, J. R. González, and D. A. Pelta, “Optimization in dynamic environments: A survey on problems, methods and measures,” *Soft Comput.*, vol. 15, no. 7, pp. 1427–1448, Jul. 2011.

[55] C. M. Fernandes, J. J. Merelo, and A. C. Rosa, “A comparative study on the performance of dissortative mating and immigrants-based strategies for evolutionary dynamic optimization,” *Inf. Sci.*, vol. 181, no. 20, pp. 4428–4459, Oct. 2011.

[56] O. Castillo, P. Melin, A. A. Garza, O. Montiel, and R. Sepúlveda, “Optimization of interval type-2 fuzzy logic controllers using evolutionary algorithms,” *Soft Comput.*, vol. 15, no. 6, pp. 1145–1160, Jun. 2011.

[57] G. M. Fathi and A. M. Saniee, “A fuzzy classification system based on ant colony optimization for diabetes disease diagnosis,” *Expert Syst. Appl.*, vol. 38, no. 12, pp. 14650–14659, Nov./Dec. 2011.

[58] E. K. Aydogan, I. Karaoglan, and P. M. Pardalos, “hGA: Hybrid genetic algorithm in fuzzy rule-based classification systems for high-dimensional problems,” *Appl. Soft Comput.*, vol. 12, no. 2, pp. 800–806, Feb. 2012.

[59] O. Castillo and P. Melin, “A review on the design and optimization of interval Type-2 fuzzy controllers,” *Appl. Soft Comput.*, vol. 12, no. 4, pp. 1267–1278, Apr. 2012.

[60] O. Castillo, R. Martínez-Marroquín, P. Melin, F. Valdez, and J. Soria, “Comparative study of bio-inspired algorithms applied to the optimization of Type-1 and Type-2 fuzzy controllers for an autonomous mobile robot,” *Inf. Sci.*, vol. 192, no. 1, pp. 19–38, Jun. 2012.

Hisao Ishibuchi (M’93–SM’10) received the B.S. and M.S. degrees in precision mechanics from Kyoto University, Kyoto, Japan, in 1985 and 1987, respectively, and the Ph.D. degree in computer science from Osaka Prefecture University, Osaka, Japan, in 1992.

Since 1987, he has been with Osaka Prefecture University, where he is currently a Professor with the Department of Computer Science and Intelligent Systems. His research interests include fuzzy rule-based classifiers, evolutionary multiobjective optimization, and evolutionary game.

Dr. Ishibuchi is currently the IEEE Computational Intelligence Society Vice-President for technical activities for 2010–2013. He is also an Associate Editor for the IEEE TRANSACTIONS ON FUZZY SYSTEMS, the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, and IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE.

Shingo Mihara received the B.S. and M.S. degrees in computer science and intelligent systems from Osaka Prefecture University, Osaka, Japan, in 2010 and 2012, respectively.

His research interests include genetic fuzzy systems and their parallel distributed implementation.

Mr. Mihara received the Best Presentation Award from the 12th International Symposium on Advanced Intelligent Systems in 2011.

Yusuke Nojima (M’00) received the B.S. and M.S. degrees in mechanical engineering from Osaka Institute of Technology, Osaka, Japan, in 1999 and 2001, respectively, and the Ph.D. degree in system function science from Kobe University, Hyogo, Japan, in 2004.

Since 2004, he has been with Osaka Prefecture University, Osaka, Japan, where he was a Research Associate and is currently an Assistant Professor with the Department of Computer Science and Intelligent Systems. His research interests include multiobjective genetic fuzzy systems, evolutionary multiobjective optimization, parallel distributed data mining, and ensemble classifier design.

Dr. Nojima has been a Vice-Chair of Task Force on Evolutionary Fuzzy Systems in the Fuzzy Systems Technical Committee at the IEEE Computational Intelligence Society since 2009. He is a Program Co-Chair at International Workshop on Genetic and Evolutionary Fuzzy Systems in 2010, 2011, and 2013.