Class Decomposition for GA-Based Classifier Agents—A Pitt Approach

Sheng-Uei Guan and Fangming Zhu

***Abstract*—This paper proposes a class decomposition approach to improve the performance of GA-based classifier agents. This approach partitions a classification problem into several class modules in the output domain, and each module is responsible for solving a fraction of the original problem. These modules are trained in parallel and independently, and results obtained from them are integrated to form the final solution by resolving conflicts. Benchmark classification data sets are used to evaluate the proposed approaches. The experiment results show that class decomposition can help achieve higher classification rate with training time reduced.**

***Index Terms*—Class decomposition, classifier agents, genetic algorithm, incremental genetic algorithm.**

I. Introduction

CLASSIFICATION problems play a major role in various fields of computer science and engineering, such as image processing and data mining. A number of soft computing approaches, such as neural networks [1], [11], [15], evolutionary algorithms [7], and fuzzy logic [13], [21], have been widely used to adaptively evolve solutions for classification problems. Among them, GA-based solutions have attracted much attention and become one of the popular techniques for classification [7], [18].

However, when GA is applied to larger-scale real-world classification problems, it still suffers from some drawbacks, such as the inefficiency in searching a large space, the difficulty in breaking the internal interference of training data, and the possibility of getting trapped in local optima. A natural approach to overcome these drawbacks is to decompose the original task into several subtasks based on certain techniques. Normally, a decomposition approach divides a task into smaller and simpler subtasks, supervises the learning of each subtask, and finally recombines individual solutions into a final solution.

Various task decomposition methods have been proposed. These methods can be roughly classified into the following categories: *functional modularity, domain modularity, class decomposition, and state decomposition, according to different partition strategies* [1], [12], [14], [15]. However, most of them are used in Artificial Neural Networks (ANN), and very few find their applications in GA, especially GA-based classification. In this paper, we aim to explore the use of class decomposition in GA and evaluate its performance on classification problems.

Our approach partitions a classification problem into several class modules in the output domain. Each module is responsible for solving a fraction of the original problem. These modules are trained in parallel and independently. Results obtained from them are integrated to obtain the final solution by resolving conflicts. The rule set obtained in each module not only delineates the class assigned, but also separates the assigned class from the other classes. Therefore, the integration module can combine the solution from each module into the final solution with some heuristics. The final solution is a global solution for all classes.

Our approach also incorporates the use of software agents [4]. Software agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy. Agents can be personalized to the end-users' preferences, and they are adaptive and can learn from past experiences. In this paper, we integrate the agent properties into traditional classifier systems, and call them classifier agents.

There are two general approaches for GA-based rule optimization and learning [8], [9], [22]. The *Michigan* approach uses GA to evolve individual rules, a collection of which comprises the solution for the classification system. Another approach is called the *Pitt* approach, where the solutions are represented by rule sets that compete against each other with respect to performance on the domain task. Those weaker ones will die, while those stronger ones survive. Little is known currently concerning the relative merits of these two approaches. In this paper, the *Pitt* approach is chosen, as we feel that the encoding mechanism for this approach is more straightforward.

For rule-based GA solution, most work in the literature concentrates on batch-mode, static domain, where the attributes, classes, and training data are all determined a priori and the task of GA is to find out the best rule sets which classify the available instances with the lowest error rate [7], [13]. However, the real-world situation is more complicated and continuously changing, a classifier agent is actually exposed to the changing environment, and it needs to evolve its solutions by adapting to various changes. Therefore, in this paper, classifier agents are designed to tackle two situations. One is that they will learn the solution from scratch, i.e., from nil initial knowledge. The other situation is that they have already possessed some available solutions, but they still need to evolve their solutions, as they need to incorporate new knowledge and adapt to the changing environment.

Manuscript received May 13, 2002; revised September 6, 2002 and December 18, 2002. This paper was recommended by Associate Editor T. Takagi.

The authors are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119260 (e-mail: eleguans@nus.edu.sg).

Digital Object Identifier 10.1109/TSMCB.2003.817030

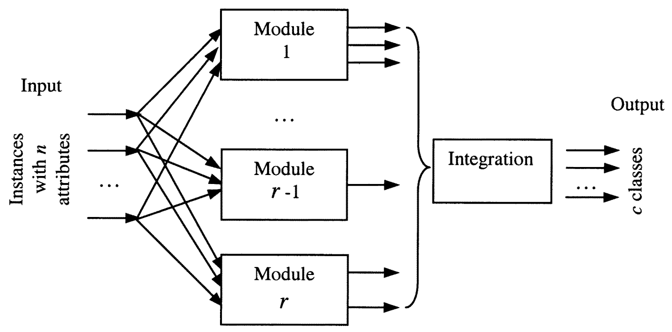

Fig. 1. Illustration of GA with class decomposition.

We design GA and incremental genetic algorithm (IGA) with class decomposition as the basic algorithms to cope with these two situations. Four benchmark data sets are used to evaluate the performance of class decomposition. The experiment results show that class decomposition can help achieve higher classification rate and save training time.

We first elaborate class decomposition in Section II. Then, our genetic approaches for classifier agents are presented in Section III. The experiment results on five benchmark data sets and their analysis are reported in Section IV. Section V places our work in the context by presenting a brief survey on related work. Section VI concludes the paper and presents future work.

II. Class Decomposition in GA-Based Classification

Let us assume a classification problem has c classes in the n-dimensional pattern space, and p vectors $X_i = (x_{i1}, x_{i2}, \ldots, x_{in})$, $i = 1, 2, \ldots, p$, $p \gg c$ are given as training patterns. The task of classification is to assign instances to one out of a set of pre-defined classes, by discovering certain relationship among attributes. Then, the discovered rules can be evaluated by classification accuracy or error rate either on the training data or test data.

Traditional GA maps attributes to classes directly in a batch manner, which means all the attributes, classes, and training data are used together to train a group of GA chromosomes. Our approach—GA with class decomposition is significantly different. As shown in Fig. 1, it generally consists of three steps. Firstly, an original problem is divided into r subproblems in terms of classes. Then,r GA modules are constructed for these subproblems, and GA in each module will be responsible for evolving a subsolution. Finally, these subsolutions are integrated to further obtain the final solution for the original problem. We present the details for each step in the following subsections.

A. Class Decomposition

First, a classification problem with a high-dimensional class space is decomposed into a set of subproblems with low-dimensional class spaces, in terms of class categories.

Following the notations presented above, the original classification problem can be denoted as

$$f : X \to T \tag{1}$$

where $X \in R^n$ is the set of instances with n attributes, and $T \in R^c$ is the set of classes. The objective of GA is to find a certain f with a satisfactory classification rate on the whole training set ξ, which can be represented as

$$\xi = \{(X_i, T_i)\}_{i=1}^{p}. \tag{2}$$

Assume the c-class problem is divided into r subproblems, each has $c_j (j = 1, 2, \ldots, r)$ classes. Denoting the class set for each subproblem as $T^{(j)}$, we have

$$T = T^{(1)} \bigcup T^{(2)} \bigcup \cdots \bigcup T^{(r)} \tag{3}$$

where $T \in R^c$, and $T^{(j)} \in R^{c_j}$, and each subproblem can be formulated as finding a certain f_j with a satisfactory classification rate on $T^{(j)}$

$$f_j : X \to T^{(j)}. \tag{4}$$

Note that it is not necessary to divide the whole class set into equal partitions. Agents can have various class partitions, which leaves them more freedom and flexibility in pursuit of suitable class decomposition.

B. Parallel Training

With the division of r subproblems, agents can construct r GA modules and run them in parallel, as shown in Fig. 1. Each module is provided with the whole training set with the complete attribute set and a fraction of the class categories to produce a corresponding fraction of the original problem.

We feed all the training data to each module, but the class categories for each module are different. We denote

$$\overline{T}^{(j)} = T - T^{(j)}, \quad j = 1, 2, \ldots, r \tag{5}$$

which means $\overline{T}^{(j)}$ is the complemented set of $T^{(j)}$. Then, the training set for each module can be represented as

$$\xi_j = \left\{ \left(X_q, T_q^{(j)} \right) \right\}_{q=1}^{M} \bigcup \left\{ \left(X_q, \overline{T}_q^{(j)} \right) \right\}_{q=M+1}^{p} \tag{6}$$

where we assume there are M instances in the training set whose classes belong to $T^{(j)}$, and the rest belong to $\overline{T}^{(j)}$.

Therefore, for each module, the class categories in interest are only those classes targeted by that module. When training each module, GA in module j has two objectives. It needs to not only classify the data with the classes in $T^{(j)}$ correctly, but also ensure that training data for the classes in $\overline{T}^{(j)}$ will not be wrongly classified into the classes in $T^{(j)}$. In other words, for those classes in $\overline{T}^{(j)}$, GA will just distinguish them from the classes in $T^{(j)}$, not necessary to differentiate them in between. As a result, GA in each module will converge more quickly.

The r GA modules are mutually independent, because the classes have been fully partitioned into several modules without overlapping. After each module gets a copy of the training patterns, they can run in parallel. Moreover, there is no communication among these modules. Therefore, the training process can be implemented with a couple of agents running on concurrent process elements. The training time for this stage is determined by the longest training time spent among the r modules.

```
t:=0;
initialize P(t);   //initialise a population of candidates randomly
evaluate P(t);   //evaluate each candidate using a fitness function
while (not terminate-condition) do        //stopping criteria
    {    select P'(t) from P(t);
         crossover P'(t);
         mutate P'(t);
         combine P'(t) and P(t) to generate P(t+1); //survivorsPercent applied
         evaluate P(t+1);
         t:=t+1;
    }
```

Fig. 2. Pseudocode of GA.

Antecedent Gene 1			......	Antecedent Gene n			Consequence Gene
Act_1	*V_{1min}*	*V_{1max}*	......	*Act_n*	*V_{nmin}*	*V_{nmax}*	*C*

where Act_j denotes whether the condition j is active or inactive, which is encoded as 1 or 0.

Note: If V_{jmin} is larger than V_{jmax} at any time, this gene will be regarded as an invalid gene. The invalid genes will make no contribution in the classification rule.

Fig. 3. Encoding mechanism for classification rules.

C. Integration

Although each GA module has evolved a portion of the solution, we cannot just simply aggregate their subsolutions as the final one. As discussed earlier, each GA module only classifies the classes in $T^{(j)}$, but not the classes in $\overline{T}^{(j)}$. Therefore, when the subsolutions are combined together, there may still exist some conflicts among the subsolutions. For example, rules from different modules may classify an instance into several classes. In order to resolve these conflicts and further improve the classification rate, the agent employs some intelligent decision rules. The detailed integration process is explained as follows.

- The agent constructs an overall rule set by aggregating all rules from r modules.
- Some decision rules are added to help resolve the above-mentioned conflicts. We believe that the ending classification rates obtained from all modules are helpful for this purpose. The following decision rules have been employed.
    - i) If an instance is classified to more than one class categories by the rule set, it will be classified to the class whose corresponding module achieves the higher classification rate in the parallel training phase, if available.
    - ii) If an instance is not classified to any class category by the rule set, it will be classified to the class whose corresponding module achieves the lowest classification rate in the parallel training phase, if available.

III. Genetic Approaches in Rule-Based Classification

In rule-based classification, there are various representation methods in terms of the rule properties (fuzzy or nonfuzzy) and the attribute properties (nominal or continuous). In this paper, we use the nonfuzzy IF-THEN rules with continuous attributes. A rule set consisting of a certain number of rules is a solution candidate for a classification problem.

As discussed earlier, we have GA and IGA for two different situations. The pseudocode for GA is shown in Fig. 2. The procedure of IGA is quite similar to that of GA. The main difference lies in the formation of initial population. Their common genetic settings are presented in the following subsections, together with their difference.

A. Encoding Mechanism

In our approach, an IF-THEN rule is represented as follows:

$$R_i : \text{IF}(V_{1\min} <= x_1 <=< V_{1\max}) \wedge (V_{2\min} <= x_2 <= V_{2\max}) \wedge \ldots \wedge (V_{n\min} <= x_n <= V_{n\max}) \text{ THEN } y = C \tag{7}$$

where R_i is a rule label, n is the number of attributes, $(x_1, x_2, \ldots x_n)$ is the input attribute set, and y is the output class category assigned with a value of C. V_{jmin} and V_{jmax} are the minimum and maximum bounds of the jth attribute x_j respectively. We encode the rule R_i according to the diagram shown in Fig. 3.

Each antecedent gene represents an attribute, and the consequence gene stands for a class. As we use the *Pitt* approach, each chromosome CR_j consists of a set of classification rules $R_i (i = 1, 2 \ldots, m)$ by concatenation

$$CR_j = \bigcup_{i=1,m} R_i \quad j = 1, 2, \ldots, s \tag{8}$$

where m is the maximum number of rules allowed for each chromosome (ruleNumber), s is the size of the population (popSize). Therefore, one chromosome will represent one rule set. Since we know the discrete value range for each attribute and class a priori, V_{jmin}, V_{jmax}, and C can be encoded each as a character by finding their positions in the ranges. Thus, the final chromosome can be encoded as a string.

B. Genetic Operators

We use one-point crossover in this paper. Referring to the encoding mechanism, we note that crossover will not cause inconsistency and thus can take place in any point of chromosome.

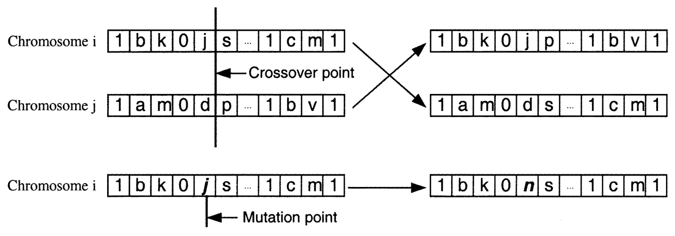

Fig. 4. Crossover and mutation.

On the contrary, the mutation operator has some constraints. The mutation point is randomly selected with a certain probability. According to the position of a selected point, we can determine whether it is an activeness, minimum or maximum element. Different mutation is available for each. For example, if an activeness element is selected for mutation, it will just be toggled. Otherwise, when a boundary-value element is selected, the algorithm will randomly select a substitute in the range of that attribute. Fig. 4 shows the operations of crossover and mutation. Each chromosome shown in the figure represents a rule set, and the characters inside represents different genes using the encoding mechanism presented in Fig. 3. The rates for mutation and crossover are selected as 0.01 and 1.0 ($\mathrm{mutationRate} = 0.01$ and $\mathrm{crossoverRate} = 1.0$). For reproduction, we set the survival rate as 50% ($\mathrm{SurvivorsPercent} = 50\%$), which means half of the parent chromosomes with higher fitness will survive into the new generation, while the other half will be replaced by the newly created children resulting from crossover and/or mutation.

Selection mechanism deals with the selection of a population that will undergo genetic operations. We use roulette wheel selection in this paper [19]. In this investigation, the probability that a chromosome will be selected for mating is given by the chromosome's fitness divided by the total fitness of all the chromosomes. By this means, chromosomes with higher fitness have a higher probability of producing offspring during selection for the next generation than those with lower fitness.

C. Fitness Function

The fitness of a chromosome reflects the success rate (i.e., classification accuracy) achieved while the corresponding rule set is used for classification. The genetic operators use this information to evolve better chromosomes over generations. As each chromosome in our approach comprises an entire rule set, the fitness function actually measures the collective behavior of the rule set. The fitness function simply measures the percentage of instances that can be correctly classified by the chromosome's rule set, which can be represented as

$$f = \frac{C}{N} = \frac{\text{number of instances correctly classified}}{\text{total number of instances}}. \tag{9}$$

Since there is more than one rule in a chromosome, it is possible that multiple rules matching the conditions for all attributes but predicting different classes. We use a voting mechanism to help resolve any conflict. That is, each rule casts a vote for the class predicted by itself, and finally the class with the highest votes is regarded as the conclusive result. If any classes tie on one instance, it is then concluded that this instance cannot be classified correctly by this rule set.

D. Stopping Criteria

There are three factors in the stopping criteria. The evolution process stops after a preset generation limit (generationLimit), or when the best chromosome's fitness has no improvement over a specified number of generations—stagnation limit (stagnationLimit), or when the best chromosome's fitness reaches a preset threshold. In this paper, this threshold is set as 1.0 for all experiments, which means 100% correct classification has been reached. Generally, the evolution process stops because either of the first two criteria satisfied, seldom the last one.

E. Initial Population for GA and IGA

Initially, a classifier agent has no prior knowledge at all. When it is presented with attributes, classes, and training data, it has to start from scratch. Therefore, the initial population of GA is randomly created. Later on, when agents have already possessed current solution and need to acquire new classes, IGA can be used to evolve the old solution into a new solution.

There are several ways to construct the new chromosomes for IGA, in terms of the selection of old chromosome. We can either use the best rule set (chromosome) as a seed for all the initial population of IGA, or the whole population in the last generation of the old GA can be used. We denote them as IGA1 and IGA2 respectively. When IGA1 is used, the genes in the best chromosome from the old population are preserved and the randomly created genes for the new class are appended. The difference of IGA2 from IGA1 is that the whole old population is used as "seeds" instead of the best one.

IV. Experimental Results and Analysis

A. Experiment Scheme

We have implemented several classifier agents running on five benchmark data sets to evaluate our approaches. The data sets chosen are the wine data, iris data, glass data, cancer data, and diabetes data. The first four are available in the UCI machine learning repository [3], and the last one is taken from the PROBEN1 collection [20]. They all are real-world problems.

All experiments are done on Pentium III 650 MHz PCs. The results reported are all averaged over ten independent runs. We record the evolution of each module and the integration process, but we are only interested in some indicative results, which include initial classification rate (CR), generation cost, training time, and ending CR. The CR in each generation is the best

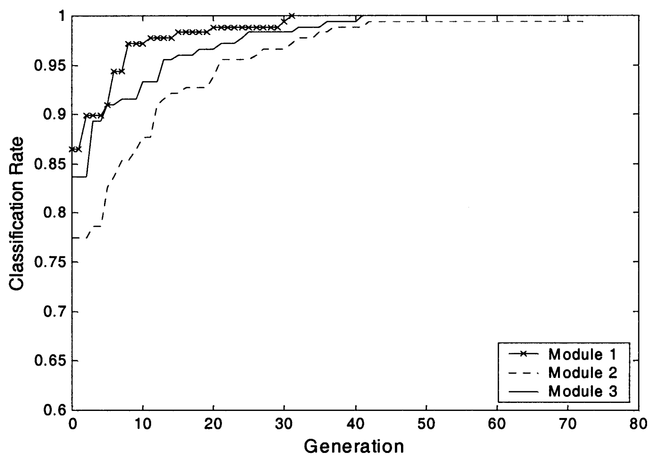

Fig. 5. Evolution process in three class modules on the wine data.

rate achieved by the whole population. The experiments in Sections IV-B and IV-C use the whole data set as the training data, while the experiments in Section IV-C use tenfold cross validation and the detailed partitioning on data sets are elaborated in that section.

B. Results and Analysis—GA Based Class Decomposition

1) Wine Data: The wine data contains the chemical analysis of 178 wines from three different cultivars in the same region in Italy. The analysis determines the quantities of 13 constituents found in each of the three types of wines. In other words, it has 13 continuous attributes, 3 classes, 178 instances, and no missing values.

Fig. 5 shows an example of the evolution process in three class modules on the wine data. Each curve shows that the best CR achieved in each generation rises steadily in each module. The evolution in module 1 and 3 stops when it reaches the maximum CR, while the evolution in module 2 stops when it reaches stagnationLimit.

Table I shows the results of GA with class decomposition on the wine data. The upper part of the table shows the approach of class decomposition and integration. We partition the wine problem into 3 modules, each for one class. The bottom part of the table provides a summary of the upper part and a comparison with normal GA approach which does not use class decomposition.

We can find from the table that each module uses GA to evolve a partial rule set, achieving a comparatively higher ending CR of about 1.0. After these rule sets are integrated and the decision rules presented in Section II-C are employed, an ending CR of 0.9978 is achieved. We record the whole process including the parallel module training and the integration process with a summary on generation, training time, and ending CR, as shown in the column of GA with class decomposition in the bottom of the table. Note that when computing the whole generation and training time, we provide two values as explained in Note 5 under Table I, considering that two different implementation methods are possible. The first value is for parallel implementation, and the other one is for serial implementation. For the initial CR, because each decomposed module has its own initial CR, we don't list any value in the summary.

Comparing the performance of GA with class decomposition to the normal GA, we find the former performs better in terms of training time and ending CR. For example, we find the approach of class decomposition improves the ending CR from 0.9534 to 0.9978 (improved by 4.7%), using much less training time (decreased from 635.3 s to 169.5 s for parallel implementation, to 368.0 s for serial implementation).

2) Iris Data: The iris data is a common benchmark in classification and pattern recognition studies. It contains 150 instances for 3 classes of iris species, i.e., iris setosa, iris versicolor, and iris virginica. Four numeric attributes are used for classification, and they are sepal length, sepal width, petal length, and petal width.

Table II shows the results on the iris data, with the same table structure as Table I. An evolved rule set is shown in Appendix A. We still divide the iris problem into 3 modules, each with one class. From the summary part of Table II, we can see that GA with class decomposition spends less training time than normal GA. The former obtains higher ending CR as well (improved by 2%). It is noted that the number of generations for the former is larger than that of latter in the case of serial implementation. However, the total training time in both parallel and serial implementation is still reduced, which is more important.

3) Diabetes Data: The diabetes data diagnose diabetes of Pima Indians. It has 8 attributes, 2 classes, and 768 instances. All attributes are continuous, and they are number of times pregnant, plasma glucose concentration, diastolic blood pressure, triceps skin fold thickness, 2-h serum insulin, body mass index, diabetes pedigree function, and age.

The diabetes problem is divided into two modules each of which deals with one class. From Table III, we can find that CR is improved from 0.7633 to 0.7897 (improved by 3.5%) as a result of using class decomposition in GA.

4) Glass Data: The glass data set contains data of different glass types. The results of a chemical analysis of glass splinters plus the refractive index are used to classify a sample to be either float processed or nonfloat processed building windows, vehicle windows, containers, tableware, or head lamps. This data set consists of 214 instances with 9 continuous attributes from 6 classes.

As the glass data have more classes, we have tried different approaches in terms of different class partitions. Table IV shows the result of an experiment where we partition the whole problem into 3 modules, each with 2 classes. We also tried another two approaches, i.e., 2-module and 6-module partitioning, which decompose the original problem into 2 and 6 modules respectively. Table V shows their comparison. An evolved rule set for 6-module partitioning is shown in Appendix B.

We find that the class decomposition approaches in all experiments improve the CR as compared to the normal GA approach. If we further compare the three class decomposition approaches with each other, we find that with the increase in the number of modules used, CR scores higher from 0.6276 to 0.7033 (improved by 12.1%), then to 0.8037 (improved by 28.1%). This tells us that a more fine-grained class decomposition approach

TABLE I
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE WINE DATA

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)
Initial CR	0.8657	0.7758	0.8427
Generations	35.6	73.3	48.5
T. time (s)	83.2	169.5	115.3
Ending CR	1.0	0.9966	0.9994
	Integration (Class=1, 2, 3)		
Ending CR	0.9978		

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.3831
Generations	73.3 / 157.4	142.8
T. time (s)	169.5 / 368.0	635.3
Ending CR	0.9978	0.9534

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=10, popSize=100, generationLimit=150, stagnationLimit=30;
3. For normal GA, ruleNumber=30, popSize=200, generationLimit=150, stagnationLimit=30.
4. "Initial CR" means the best classification rate achieved by the initial population;
"Generations" means the generation needed to reach stopping criteria;
"T. time (s)" means the training time cost in seconds;
"ending CR" means the best classification rate achieved by the resulting population.
5. Both Generations and T. time for "GA with class decomposition" have two values, the first value is the longest training time consumed by all modules, and the other value is the total training time consumed by all modules.
6. The computation time of the integration process is only an evaluation time on the training patterns. As it is less than 0.01s, it is ignored here.
7. Other tables follow the same notation as this table.

TABLE II
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE IRIS DATA

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)
Initial CR	0.9020	0.9167	0.8853
Generations	7.2	55.3	50.5
T. time (s)	2.1	14.7	13.2
Ending CR	1.0	0.9820	0.9747
	Integration (Class=1, 2, 3)		
Ending CR	0.9820		

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.546
Generations	55.3 / 113	78.0
T. time (s)	14.7 / 30	38.5
Ending CR	0.9820	0.9627

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=2, popSize=50, generationLimit=100, stagnationLimit=30;
3. For normal GA, ruleNumber=6, popSize=100, generationLimit=100, stagnationLimit=30.

will achieve a higher CR, which may be explained as a finer partition can reduce the internal interference among the training data and resolve better the conflicts in each module.

In [12], a modular neural network approach is used to classify the glass data, and the final error rate achieved is 34.9%, which is equivalent of 0.651 for the classification rate. We can find that the result of our 2-module approach is comparable to their approach, while our 3-module and 6-module approaches perform even better.

C. Results and Analysis—IGA Based Class Decomposition

In this section, we step further to explore the application of class decomposition in IGA. The main feature of IGA is that it assumes an agent already has a solution and needs to evolve a new solution to accommodate new classes.

Assuming a target problem has 3 classes, we explain the detailed experiment steps as follows. (It is easy to derive similar experiment settings for other problems with different number of classes.)

- We assume that an agent only knows classes 1 and 2 at first. It uses normal GA to evolve an optimal rule set on the currently known attributes and classes.
- When the agent knows another class—class 3, it will use IGA to evolve a new rule set. We experiment on two different approaches for IGA, namely, IGA1 and IGA2, and compare the effect of class decomposition on these two

TABLE III
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE DIABETES DATA

	Module 1 (Class=1)	Module 2 (Class=2)
Initial CR	0.7122	0.6845
Generations	159.8	184.2
T. time (s)	556.0	679.6
Ending CR	0.7879	0.8044
	Integration (Class=1, 2)	
Ending CR	0.7897	

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.6434
Generations	184.2 / 344	167.5
T. time (s)	679.6 / 1235.6	947.1
Ending CR	0.7897	0.7633

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=15, popSize=100, generationLimit=200, stagnationLimit=30;
3. For normal GA, ruleNumber=30, popSize=100, generationLimit=200, stagnationLimit=30.

TABLE IV
PERFORMANCE OF GA WITH 3-MODULE CLASS DECOMPOSITION ON THE GLASS DATA

	Module 1 (Class=1, 2)	Module 2 (Class=3, 4)	Module 3 (Class=5, 6)
Initial CR	0.4953	0.8645	0.8911
Generations	145.3	104.4	104.1
T. time (s)	189.9	170.3	167.0
Ending CR	0.7411	0.9126	0.9612
	Integration (Class=1, 2, 3, 4, 5, 6)		
Ending CR	0.7033		

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.3332
Generations	145.3 / 353.8	146.6
T. time (s)	189.9 / 527.2	577.2
Ending CR	0.7033	0.5495

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=10, popSize=100, generationLimit=150, stagnationLimit=50;
3. For normal GA, ruleNumber=30, popSize=200, generationLimit=150, stagnationLimit=50.

TABLE V
COMPARISON OF DIFFERENT APPROACHES OF GA WITH CLASS DECOMPOSITION ON THE GLASS DATA

Summary	Normal GA	GA with Class Decomposition (2-module)	GA with Class Decomposition (3-module)	GA with Class Decomposition (6-module)
Initial CR	0.3332	-	-	-
Generations	146.6	142.1 / 276.8	145.3 / 353.8	147.9 / 605.9
T. time (s)	577.2	244.7 / 442.2	189.9 / 527.2	185.5 / 693.5
Ending CR	0.5495	0.6276	0.7033	0.8037

approaches. (The difference between IGA1 and IGA2 has been discussed in Section III-E.)

Table s VI–VIII summarize the results of experiments on the wine, iris, and glass data respectively. For each data set, normal GA runs with an incomplete set of classes first. Upon getting the results from GA, IGA will then work with different approaches on a complete set of classes. Therefore, these approaches have the same starting point that is fair for comparison.

It is found from these tables that the approaches with class decomposition perform better than those without class decomposition, for both IGA1 and IGA2. The former always achieves higher CR and spends less training time in both serial and parallel implementation. For IGA1 in Table VII, the class decomposition approach improves the ending CR from 0.9610 to 0.9807 with an increase of 2% in accuracy, and a decrease of training time from 35.9 s to 31.3 s in the case of serial implementation, with a saving of about 12.8%. For the glass data, the improvement on the ending CR becomes more significant. It is noted from Table VIII that the ending CR of IGA1 improves by 0.04 (i.e., 6.9%), and CR of IGA2 improves by 0.05 (i.e., 7.9%). As described earlier in Section III-E, IGA1 uses the best chromosome from the old, evolved population, while IGA2 uses the whole evolved population. Results show that IGA2 outperforms IGA1 on the iris and wine data, but is inferior on the glass data. This means group population may be better for incremental learning on some data sets than the best chromosome alone, as it

TABLE VI
COMPARISON OF PERFORMANCE OF IGA WITH/WITHOUT CLASS DECOMPOSITION ON THE WINE DATA

		Agent 1 (Class=1& 2)			
GA	Initial CR	0.5846			
	Generations	51.9			
	T. time (s)	432.5			
	Ending CR	0.9992			
		Agent 1 (Class=1, 2 & 3) (Without Class Decomposition)		Agent 1 (Class=1, 2 & 3) (With Class Decomposition)	
		IGA1	IGA2	IGA1	IGA2
IGAs	Initial CR	0.7573	0.7534	-	-
	Generations	126.8	110.9	84.9 / 185.6	76.9 / 167.8
	T. time (s)	1544.7	1324.0	197.7 / 429.5	172.8 / 378
	Ending CR	0.9961	0.9966	0.9966	0.9983

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For GA and IGA without class decomposition, ruleNumber=30, popSize=500, generationLimit=150, stagnationLimit=30;
3. For IGA with class decomposition, the parameters are the same as those listed in table 1.

TABLE VII
COMPARISON OF PERFORMANCE OF IGA WITH/WITHOUT CLASS DECOMPOSITION ON THE IRIS DATA

		Agent 1 (Class=1& 2)			
GA	Initial CR	0.8110			
	Generations	6.6			
	T. time (s)	2.3			
	Ending CR	1.0			
		Agent 1 (Class=1, 2 & 3) (Without Class Decomposition)		Agent 1 (Class=1, 2 & 3) (With Class Decomposition)	
		IGA1	IGA2	IGA1	IGA2
IGAs	Initial CR	0.7487	0.6953	-	-
	Generations	72.6	74.9	57.1 / 114.1	55.5 / 104.6
	T. time (s)	35.9	36.7	16.2 / 31.3	14.9 / 27.9
	Ending CR	0.9610	0.9627	0.9807	0.9833

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For GA and IGA without class decomposition, ruleNumber=6, popSize=100, generationLimit=100, stagnationLimit=30;
3. For IGA with class decomposition, the parameters are the same as those listed in table 2.

TABLE VIII
COMPARISON OF PERFORMANCE OF IGA WITH/WITHOUT CLASS DECOMPOSITION ON THE GLASS DATA

		Agent 1 (Class=1, 2, 3, 4)			
GA	Initial CR	0.4273			
	Generations	144.5			
	T. time (s)	974.2			
	Ending CR	0.6648			
		Agent 1 (Class=1, 2, 3, 4, 5 & 6) (Without Class Decomposition)		Agent 1 (Class=1, 2, 3, 4, 5 & 6) (With Class Decomposition)	
		IGA1	IGA2	IGA1	IGA2
IGAs	Initial CR	0.5388	0.5332	-	-
	Generations	138.8	146.7	146.0 / 365.3	147.5 / 377.2
	T. time (s)	1147.0	1156.5	117.1 / 306.2	107.7 / 303.3
	Ending CR	0.6808	0.6664	0.7280	0.7192

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For GA and IGA without class decomposition, ruleNumber=30, popSize=500, generationLimit=150, stagnationLimit=50;
3. For IGA with class decomposition, the parameters are the same as those listed in table 4.

carries more information. However, group population demands extra resource, for example storage, which may not be available in some circumstances.

D. Generalization Performance and Comparison to Related Work

The generalization performance of the evolved rule set is evaluated with tenfold cross validation. In this scheme, the complete data set is divided into ten subsets in the same size. Then, nine subsets are used as training data and the other subset is used as test data. Ten iterations are performed so that each of the ten subsets is used as test data just once. The results are averaged over ten iterations. Experiments on three data sets—wine, iris, and cancer data—are conducted and their results are shown in Table s IX–XI, respectively. The cancer problem diagnoses whether a breast cancer is benign or malignant. It has 9 attributes, 2 classes, and 699 instances.

TABLE IX
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE WINE DATA

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.4072
Generations	74.2 / 150.8	146.4
T. time (s)	141.2 / 294.7	571.7
Ending CR	0.9978	0.9817
Test CR	0.9167	0.8450

Notes:
1. The experiment settings are the same as that for Table 1, except that tenfold cross validation is used.
2. "Test CR" means the classification rate achieved by the resulting population on the test data.

TABLE X
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE IRIS DATA

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.5150
Generations	54.4 / 109.4	82.1
T. time (s)	13.9 / 27.8	37.0
Ending CR	0.9820	0.9624
Test CR	0.9560	0.9360

Notes:
1. The experiment settings are the same as that for Table 2, except that tenfold cross validation is used.

TABLE XI
PERFORMANCE OF GA WITH CLASS DECOMPOSITION ON THE CANCER DATA

Summary	GA with Class Decomposition	Normal GA
Initial CR	-	0.7538
Generations	129.7 / 229.2	122.2
T. time (s)	393.9 / 716.9	615.2
Ending CR	0.9815	0.9769
Test CR	0.9530	0.9451

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=15, popSize=100, generationLimit=200, stagnationLimit=30;
3. For normal GA, ruleNumber=30, popSize=100, generationLimit=200, stagnationLimit=30;
4. Tenfold cross validation is used.

We can find from these tables that GA with class decomposition always performs better than normal GA, in terms of both ending CR and test CR. Using the wine data as an example, the classification rate is improved from 0.9817 to 0.9978 on the training data (i.e., 1.6%), and improved from 0.8450 to 0.9167 on the test data (i.e., 8.5%). The results show that the generalization power of the rule set is also enhanced by the use of class decomposition.

We compare the performance of our approach with other conventional methods. Table XII shows the classification error rates for different methods on the iris data $(\text{error rate} = 1 - \text{classification rate})$. The error rates for the first six methods were reported in [25], and the last two items are the results from our approach. We find that the error rate with our normal GA approach is comparable to the other methods. The training error rate of our GA with class decomposition approach, which is 0.018, is better than most of the other methods listed. In terms of the test error rate, our GA with class decomposition approach achieves a comparable error rate.

For the wine data, Corcoran and Sen [7] used a real-coded genetic-based machine learning approach to evolve nonfuzzy IF-THEN rules, and achieved an average training rate of 99.5%. Ishibuchi *et al.* [13] designed a fuzzy classifier by means of an integer-coded GA and grid partitioning, and they got an average training rate of 98.5%. Setnes and Roubos [21] used GA-fuzzy classifiers which achieve three misclassifications out of 178 instances (i.e., a classification rate of 98.3%). Comparing to these results in the literature, our approach achieves the highest rate as 99.78%.

Regarding the cancer data, [26] reported a test rate of 95.9% with the use of hyperplanes, and [27] reported 93.7% with the use of 1-nearest neighbor. Our approach achieves 95.3%, which is better than the latter approach, while very close to the former.

V. RELATED WORK

Pattern classification problems have been widely researched with different approaches including statistical methods [25], neural networks [15], [23], fuzzy sets [21], and evolutionary algorithms [18]. GA-based solutions have become one of the popular techniques for classification. De Jong *et al.* considered the application of GA to a symbolic learning task—supervised concept learning from a set of examples [10]. Corcoran *et al.* used GA to evolve a set of classification rules with real-valued attributes [7]. Ishibuchi *et al.* examined the performance of a fuzzy genetic-based machine learning methods for pattern classification problems with continuous attributes [13]. However, their work only covers batch-mode algorithms, while our work here presents a new approach based on class decomposition.

Compared to the other methods, GA-based approaches have many advantages. For example, neural networks have no explanatory power by default to describe why results are as they are. This means that the knowledge (models) extracted by neural networks is still hidden and distributed over the network. GA has comparatively more explanatory power, as it explicitly shows the evolutionary process of solutions and the solution format is completely decodable.

There is a stream of research called parallel genetic algorithms (PGAs) [5], [17], which are parallel implementation of GAs. [5] proposed a Markov Chain model to predict the effect of parameters, such as number of population, size, topology, migration rate, on the performance of PGAs. [17] explored the application of PGAs in rule mining for large databases. There are two main models for PGA—Island model and Neighborhood model [5], [6]. Our class decomposition is similar to the method of PGAs, since they all run several populations in parallel. The distinct feature of class decomposition is that subpopulations in our approach are all independent, so that there is no need for migration among them.

Decomposition methods have been used in various fields, such as classification, data mining, clustering, etc. [24] explored the application of domain decomposition genetic algorithms to the design of frequency selective surfaces. [16] presented a new machine learning model for classification problems. It decomposes multi-class classification problems into sets of two-class

TABLE XII
COMPARISON OF ERROR RATES OF VARIOUS CLASSIFICATION METHODS ON THE IRIS DATA

Methods	Error Rate (Training)	Error Rate (Test)
Quadratic	0.020	0.027
Bayers indep.	0.047	0.067
Bayers 2nd order	0.040	0.160
Neural net (BP)	0.017	0.033
PVM rule	0.027	0.040
CART tree	0.040	0.047
Normal GA	**0.038**	**0.064**
GA with decomposition	**0.018**	**0.044**

Notes: Error rate=1-clssifcation rate (CR).

subproblems which are assigned to nonlinear dichotomizers. [2] presented a new measure to determine the degree of dissimilarity between two given problems, and suggests a way to search for a strategic splitting of the feature space that identifies different characteristics.

VI. CONCLUSIONS AND DISCUSSIONS

This paper proposes a new approach named class decomposition for GA-based classifier agents. A classification problem is decomposed into several modules in terms of class decomposition, and each module is responsible for solving a fraction of the original problem. These modules are trained in parallel, and the subsolutions obtained from them are integrated to obtain the final solution by resolving conflicts. Five benchmark data sets are used for evaluation. The results show that class decomposition can help achieve a higher classification rate with training time reduced.

Classifier agents may require some intelligence on determining a suitable decomposition which includes the size of each module, the selection of classes into modules. Agents can try on different combinations to find the most suitable partition. For example, in this paper, agents partition the output classes in a nonoverlapping manner, which means rules for each class are only trained in one single module. Alternatively, agents can have some overlapping in class decomposition, which will be more robust in the presence of faults and may lead to further improvement on classification accuracy. In a more challenging environment where the number of classes is unknown in advance, the classifier agents may incrementally introduce new modules, starting from a small number of modules. The new modules can be introduced via class decomposition to attain higher accuracy. The agents should have the ability to observe the trend of achieved classification rate, and stop at a desirable stage.

Classifier agents can also be implemented in a multi-agent environment, where agents can exchange information on new attributes and classes. If available, they can also exchange evolved rule sets. They can help provide new training/test data, or even challenge each other with unsolved instances.

APPENDIX A
EXAMPLE OF RULE SET FOR THE IRIS DATA

no. 1 $\mathrm{IF}(0.00 <= X4 <= 0.50)$ THEN Class = 1.
no. 2 $\mathrm{IF}(3.10 <= X2 <= 4.00)$ AND $(1.50 <= X3 <= 3.20)$ THEN Class = 1.
no. 3 $\mathrm{IF}(1.90 <= X3 <= 5.30)$ AND $(0.60 <= X4 <= 1.70)$ THEN Class = 2.
no. 4 $\mathrm{IF}(2.20 <= X2 <= 4.50)$ AND $(3.80 <= X3 <= 4.70)$ AND$(0.40 <= X4 <= 2.50)$ THEN Class = 2.
no. 5 $\mathrm{IF}(5.00 <= X3 <= 6.20)$ THEN Class = 3.
no. 6 $\mathrm{IF}(1.70 <= X4 <= 2.60)$ THEN Class = 3.

APPENDIX B
EXAMPLE OF RULE SET FOR THE GLASS DATA

no. 1 $\mathrm{IF}(1.27 <= X4 <= 1.77)$ AND $(0.29 <= X6 <= 3.47)$ AND $(0.24 <= X9 <= 0.27)$ THEN Class = 1.
no. 2 $\mathrm{IF}(1.52 <= X1 <= 1.53)$ AND $(70.59 <= X5 <= 72.31)$ AND $(6.45 <= X7 <= 9.43)$ AND THEN Class = 1.
no. 3 $\mathrm{IF}(0.53 <= X3 <= 1.77)$ AND $(2.26 <= X8 <= 2.58)$ THEN Class = 1.
no. 4 $\mathrm{IF}(12.65 <= X2 <= 13.61)$ AND $(0.46 <= X4 <= 2.12)$ AND $(70.75 <= X5 <= 71.72)$ AND $(2.13 <= X6 <= 4.27)$ THEN Class = 1.
no. 5 $\mathrm{IF}(1.52 <= X1 <= 1.53)$ AND $(0.38 <= X6 <= 0.63)$ THEN Class = 1.
no. 6 $\mathrm{IF}(12.91 <= X2 <= 13.47)$ AND $(1.44 <= X4 <= 1.63)$ THEN Class = 2.
no. 7 $\mathrm{IF}(3.55 <= X3 <= 3.82)$ AND $(70.68 <= X5 <= 73.05)$ AND $(3.70 <= X6 <= 5.63)$ AND $(2.29 <= X8 <= 3.20)$ AND $(0.00 <= X9 <= 0.51)$ THEN Class = 2.
no. 8 $\mathrm{IF}(0.40 <= X3 <= 1.73)$ AND $(70.27 <= X5 <= 71.93)$ AND $(0.27 <= X6 <= 4.62)$ AND $(0.04 <= X8 <= 1.43)$ THEN Class = 2.
no. 9 $\mathrm{IF}(12.91 <= X2 <= 13.47)$ AND $(1.44 <= X4 <= 1.63)$ THEN Class = 2.
no. 10 $\mathrm{IF}(12.89 <= X2 <= 15.31)$ AND $(1.20 <= X3 <= 3.87)$ AND $(0.12 <= X9 <= 0.46)$ THEN Class = 2.
no. 11 $\mathrm{IF}(1.52 <= X1 <= 1.53)$ AND $(0.74 <= X4 <= 2.48)$ AND $(8.56 <= X7 <= 14.42)$ AND $(0.18 <= X8 <= 0.64)$ AND $(0.24 <= X9 <= 0.47)$ THEN Class = 3.

no. 12 IF(1.52 <= X1 <= 1.53) AND (10.79 <= X2 <= 13.59) AND (0.16 <= X3 <= 3.54) AND (0.47 <= X4 <= 1.13) AND (0.22 <= X9 <= 0.52) THEN Class = 3.

no. 13 IF(0.96 <= X3 <= 3.07) AND (70.40 <= X5 <= 75.15) AND (3.00 <= X6 <= 3.50) THEN Class = 3.

no. 14 IF(0.78 <= X3 <= 0.96) AND (69.99 <= X5 <= 71.89) AND (0.97 <= X8 <= 2.71) THEN Class = 3.

no. 15 IF(0.52 <= X4 <= 0.85) AND (70.79 <= X5 <= 73.65) AND (7.93 <= X7 <= 10.87) THEN Class = 3.

no. 16 IF(70.66 <= X5 <= 73.94) AND (9.85 <= X7 <= 11.70) THEN Class = 4.

no. 17 IF(1.52 <= X1 <= 1.53) AND (0.23 <= X3 <= 3.34) AND (0.36 <= X4 <= 1.32) AND (70.40 <= X5 <= 74.03) AND (13.77 <= X7 <= 15.19) AND (1.22 <= X8 <= 2.53) THEN Class = 4.

no. 18 IF(1.85 <= X3 <= 1.88) AND (0.91 <= X4 <= 1.72) AND (4.64 <= X6 <= 6.01) AND (13.95 <= X7 <= 14.39) AND (0.05 <= X9 <= 0.52) THEN Class = 4.

no. 19 IF(1.82 <= X8 <= 2.33) THEN Class = 4.

no. 20 IF(1.52 <= X1 <= 1.53) AND (14.81 <= X2 <= 15.64) AND (1.31 <= X3 <= 3.23) AND (0.57 <= X4 <= 0.68) AND (71.45 <= X5 <= 73.05) AND (3.01 <= X6 <= 4.34) AND (0.27 <= X8 <= 1.46) THEN Class = 4.

no. 21 IF(1.52 <= X1 <= 1.53) AND (14.01 <= X2 <= 16.74) AND (2.15 <= X3 <= 2.31) AND (72.61 <= X5 <= 74.32) AND (8.01 <= X7 <= 12.59) AND (0.06 <= X9 <= 0.32) THEN Class = 5.

no. 22 IF(3.69 <= X3 <= 3.84) AND (2.94 <= X4 <= 3.13) AND (4.65 <= X6 <= 5.15) AND (8.00 <= X7 <= 8.73) AND (0.78 <= X8 <= 0.90) THEN Class = 5.

no. 23 IF(1.52 <= X1 <= 1.53) AND (0.93 <= X3 <= 2.87) AND (69.78 <= X5 <= 73.17) AND (5.00 <= X6 <= 6.03) THEN Class = 5.

no. 24 IF(1.52 <= X1 <= 1.53) AND (10.80 <= X2 <= 16.86) AND (0.82 <= X3 <= 1.81) AND (71.21 <= X5 <= 71.23) THEN Class = 5.

no. 25 IF(1.51 <= X1 <= 1.52) AND (0.51 <= X3 <= 1.61) AND (1.07 <= X4 <= 1.88) AND (70.00 <= X5 <= 72.32) AND (10.11 <= X7 <= 1.24) AND (0.01 <= X8 <= 1.73) THEN Class = 5.

no. 26 IF(1.52 <= X1 <= 1.53) AND (14.44 <= X2 <= 16.09) AND (11.59 <= X7 <= 14.12) AND (0.65 <= X8 <= 1.60) AND (0.30 <= X9 <= 0.39) THEN Class = 6.

no. 27 IF(1.52 <= X1 <= 1.53) AND (11.58 <= X2 <= 16.31) AND (3.29 <= X3 <= 3.82) AND (0.37 <= X6 <= 3.72) THEN Class = 6.

no. 28 IF(2.15 <= X3 <= 2.38) AND (0.76 <= X6 <= 3.13) THEN Class = 6.

no. 29 IF(13.64 <= X2 <= 13.67) AND (1.84 <= X4 <= 2.44) AND (6.36 <= X7 <= 13.65) AND (0.06 <= X8 <= 2.10) AND (0.17 <= X9 <= 0.45) THEN Class = 6.

no. 30 IF(16.19 <= X2 <= 16.44) AND (1.35 <= X3 <= 3.13) AND (74.51 <= X5 <= 74.85) AND (8.01 <= X7 <= 10.48) AND (0.03 <= X8 <= 2.78) AND (0.19 <= X9 <= 0.52) THEN Class = 6.

References

[1] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka, "Efficient classification for multiclass problems using modular neural networks," *IEEE Trans. Neural Networks*, vol. 6, no. 1, pp. 117–124, 1995.

[2] C. Apté, S. J. Hong, J. Hosking, J. Lepre, E. Pednault, and B. Rosen, "Decomposition of heterogeneous classification problems," in *Proc. 2nd Int. Symp. Advances in Intelligent Data Analysis, Reasoning About Data*: LNCS, 1997, vol. 1280, pp. 17–28.

[3] UCI Repository of Machine Learning Databases, C. L. Blake and C. J. Merz. (1998). *[http://www.ics.uci.edu/~mlearn/MLRepository.html]* [Online]

[4] J. M. Bradshaw, *Software Agents*, Mass.: MIT Press, 1997.

[5] E. Cantu-Paz, "Markov chain models of parallel genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 216–226, 2000.

[6] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Boston, MA: Kluwer, 2000.

[7] A. L. Corcoran and S. Sen, "Using real-valued genetic algorithm to evolve rule sets for classification," in *Proc. 1st IEEE Conf. on Evolutionary Computation*, Orlando, FL, 1994, pp. 120–124.

[8] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*, Singapore: World Scientific, 2001.

[9] K. A. DeJong, "Learning with genetic algorithms: an overview," *Mach. Learn.*, vol. 3, pp. 121–138, 1988.

[10] K. A. DeJong and W. M. Spears, "Learning concept classification rules using genetic algorithms," in *Proc. Int. Joint Conf. Artif. Intell.*, 1991, pp. 651–656.

[11] S. U. Guan and S. C. Li, "Incremental learning with respect to new incoming input attributes," *Neural Process. Lett.*, vol. 14, no. 3, pp. 241–260, 2001.

[12] ——, "Parallel growing and training of neural networks using output parallelism," *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 1–9, 2002.

[13] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems," *IEEE Trans. Syst., Man, Cybern.*, pt. Part B, vol. 29, no. 5, pp. 601–618, 1999.

[14] R. E. Jenkins and B. P. Yuhas, "A simplified neural network solution through problem decomposition: the case of the truck backer-upper," *IEEE Trans. Neural Networks*, vol. 4, no. 4, pp. 718–720, 1993.

[15] B. L. Lu and M. Ito, "Task decomposition and module combination based on class relations: A modular neural network for pattern classification," *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1244–1256, 1999.

[16] F. Masulli and G. Valentini, "Parallel nonlinear dichotomizers," in *Proc. IEEE-INNS-ENNS Int. Joint Conf. on Neural Networks*, vol. 2, 2000, pp. 29–33.

[17] N. Melab and E. Talbi, "A parallel genetic algorithm for rule mining," in *Proc. 15th Int. Parallel Distributed Processing Symp.*, 2001, pp. 1347–1352.

[18] J. J. Merelo, A. Prieto, and F. Moran, "Optimization of classifiers using genetic algorithms," in *Advances in the Evolutionary Synthesis of Intelligent Agents*, M. Patel, V. Honavar, and K. Balakrishnan, Eds. Cambridge, MA: MIT Press, 2001, pp. 91–108.

[19] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. New York: Springer, 1996.

[20] L. Prechelt, "PROBEN1: A Set of Neural Network Benchmark Problems and Benchmarking Rules," Dept. Informatics, Univ. Karlsruhe, Karlsruhe, Germany, Technical Report 21/94, 1994.

[21] M. Setnes and H. Roubos, "GA-fuzzy modeling and classification: complexity and performance," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 5, pp. 509–522, 2000.

[22] S. F. Smith, "A Learning System Based on Genetic Adaptive Algorithms," Ph.D. dissertation, Univ.Pittsburgh, Pittsburgh, PA, 1980.

[23] L. Su, S. U. Guan, and Y. C. Yeo, "Incremental self-growing neural networks with the changing environment," *J. Intell. Syst.*, vol. 11, no. 1, pp. 43–74, 2001.

[24] D. S. Weile and E. Michielssen, "The use of domain decomposition genetic algorithms exploiting model reduction for the design of frequency selective surfaces," *Comput. Methods Appl. Mech. Eng.*, vol. 18, no. 6, pp. 439–458, 2000.

[25] S. M. Weiss and C. A. Kulikowski, *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. San Mateo, CA: Morgan Kaufmann, 1991.

[26] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," in *Proc. Nat. Acad. Sci.*, 1990, pp. 9193–9196.

[27] J. Zhang, "Selecting typical instances in instance-based learning," in *Proc. 9th Int. Mach. Learn. Conf.*, 1992, pp. 470–479.

Sheng-Uei Guan received the M.Sc. and Ph.D. degrees from the University of North Carolina at Chapel Hill.

He is currently with the Electrical and Computer Engineering Department at National University of Singapore. He has worked in a prestigious R&D organization for several years, serving as a Design Engineer, Project Leader, and Manager. He has served as a member on the R.O.C. Information and Communication National Standard Draft Committee. After leaving the industry, he joined Yuan-Ze University, Taiwan, R.O.C., for three and half years. He served as Deputy Director for the Computing Center, and also as the Chairman for the Department of Information and Communication Technology. He joined La Trobe University in Australia with the Department of Computer Science and Computer Engineering where he helped to create a new multimedia systems stream.

Fangming Zhu received the B.S. and M.S. degrees from Shanghai Jiaotong University, China, in 1994 and 1997, respectively. He is now pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering at National University of Singapore.

His current research interests include evolutionary computation, pattern classification, and intelligent agents.