Cooperative Co-evolution of GA-based Classifiers Based on Input Increments

Fangming Zhu[1] and Sheng-Uei Guan[2*]

[1]Institute of Systems Science

[2]Department of Electrical and Computer Engineering

National University of Singapore

10 Kent Ridge Crescent, Singapore 119260

Abstract

Genetic algorithms (GAs) have been widely used as soft computing techniques in various applications, while cooperative co-evolution algorithms were proposed in the literature to improve the performance of basic GAs. In this paper, a new cooperative co-evolution algorithm, namely ECCGA, is proposed in the application domain of pattern classification. Concurrent local and global evolution and conclusive global evolution are proposed to improve further the classification performance. Different approaches of ECCGA are evaluated on benchmark classification data sets, and the results show that ECCGA can achieve better performance than the cooperative co-evolution genetic algorithm and normal GA. Some analysis and discussions on ECCGA and possible improvement are also presented.

Keywords: genetic algorithms, cooperative co-evolution, classifiers

[*] Corresponding author: eleguans@nus.edu.sg

1. Introduction

Genetic algorithms (GAs) have been successfully applied to a wide range of optimization problems including design, scheduling, routing, and control, etc. As typical algorithms in evolutionary computation, GAs have also attracted much attention and become one of the most popular techniques for pattern classification [1] [2]. Fidelis et al. [3] presented a classification algorithm based on GA that discovers comprehensible rules. Merelo et al. [4] presented a general procedure for optimizing classifiers based on a two-level GA operating on variable size chromosomes. Among these systems, rule-based solution is widely used for classification problems, either through supervised or unsupervised learning [5]. The advantage of GA becomes more compelling when the search space of a task is much larger.

In the literature, various models and approaches have been proposed to address difficulties in mapping the domain solutions into GA models, while avoiding the possibility of being trapped into local optima. For example, Holland [6] indicated that crossover induces a linkage phenomenon. It has been shown that GAs work well only if the building blocks are tightly linked on the chromosome [1]. In order to tackle the linkage-learning problem, some algorithms have been proposed to include linkage design into problem representation and recombination operator or use some probabilistic-based models. For instance, the linkage learning genetic algorithm (LLGA) was proposed in [7] for tackling the linkage and ordering problem, while several Probabilistic Model Building Genetic Algorithms (PMBGAs) have been proposed [8] to generate new child population based on probabilistic models. For multi-objective optimization problems, incremental multi-objective genetic algorithms have been employed to search for the Pareto-optimal set more accurately and efficiently [9][10].

In another aspect, complex systems can be decomposed and evolved in the form of interacting co-evolution systems. Classifier systems evolves interacting rule whose individual fitness are determined by their interaction with other rules, and concept of niches and species is employed [11] [12]. In the island model, a number of subpopulations compete each other, and individuals may migrate from one subpopulation to another [13].

Cooperative co-evolution has attracted more research interests as an effective approach to decompose complex structure and achieve better performance. The idea of cooperative co-evolution mainly derives from the species in the nature. As a normal practice, a complex problem may be decomposed into several sub-problems. The partial solutions in the subpopulations evolve separately, but with a common objective. Cooperative co-evolution, together with incremental learning, has been applied with many soft computing techniques such as neural networks [14-18]. The introductive work on cooperative co-evolution in the GA domain was conducted by Dejong and Potter [19] [20]. In their work, the cooperative co-evolution genetic algorithm (CCGA) was initially designed for function optimization, and later the general architecture was proposed for cooperative co-evolution with co-adapted subcomponents.

In this paper, the cooperative co-evolution scheme is revisited with a rule-based GA system for pattern classification [21-24]. In order to improve further the classification performance, an enhanced cooperative co-evolution genetic algorithm (ECCGA) approach is proposed. The concurrent global and local evolution and conclusive global evolution are integrated into the ECCGA. Different approaches with ECCGA are evaluated on benchmark data sets. The experimental results show that ECCGA performs better than the CCGA and normal GA. On the basis of the results, we have an extended discussion on the inner mechanisms and possible improvements.

The integration of the local fitness element is a major feature of ECCGA. We postulate that, apart from the global fitness as exercised normally in CCGA, the local fitness element is also a suitable indicator and facilitator for the whole evolution. Therefore, both local and global fitness are employed to guide the evolution. Our experimental results have supported such postulation. It is found that the additional evolution pressure from the local fitness element may produce more opportunities to escape from traps in local optima and advance the evolution further.

The rest of the paper is organized as follows. In section 2, the design of GA and CCGA is introduced. Then, the details of ECCGA are elaborated in section 3. The experimental results of CCGA/ECCGA on four benchmark data sets and their analysis are reported in section 4. Section 5 presents some analysis and discussions based on the experimental results, and section 6 concludes the paper.

2. Design of GA and CCGA

In normal GA, an initial population is created randomly. Based on fitness evaluation, some chromosomes are selected by a selection mechanism. Crossover and mutation will then be applied to these selected chromosomes and the child population is thus generated. Certain percentage of the parent population will be preserved and the rest will be replaced by the child population. The evolution process will continue until it satisfies the stopping criteria [1] [24].

Let us assume a classification problem has c classes in the n-dimensional pattern space, and p vectors $X_i = (x_{i1},\ x_{i2},\ \ldots,\ x_{in})$, $i = 1,2,\ldots,p,\quad p >> c$, 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.

```
Decompose the problem into n species;
gen=0;
for each species s
        {  randomly initialize population p(gen);
           evaluate fitness of each individual;
        }
while (not termination condition)
        {  gen++;
        for each species s
           {   select p(gen) from p(gen-1) based on fitness;
               apply genetic operators to p(gen);
               evaluate fitness of each individual in p(gen);
           }
        }
```

Figure 1. Pseudocodes of CCGA

Figure 1 shows the algorithm of CCGA. As the first step, the original problem should be decomposed into n sub-problems, and each of which is handled by one species. Then, each species evolves in a round robin fashion using the same procedure as the normal GA, with the exception of fitness evaluation. When an individual in one species is evaluated, it will be combined with other individuals from the other species and the fitness of the resulting chromosome is evaluated and returned.

In our rule-based GA system, we use the non-fuzzy IF-THEN rules with continuous attributes for classifiers. A rule set consisting of a certain number of rules is a solution candidate for a classification problem. The detailed designs are discussed in the following subsections and can be found further in [22].

2.1 Encoding Mechanism

An IF-THEN rule is represented as follows:

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

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 rule R_i according to the diagram shown in Figure 2.

Antecedent Element 1			……	Antecedent Element n			Consequence Element
Act_1	V_{1min}	V_{1max}	……	Act_n	V_{nmin}	V_{nmax}	C

Notes:

1. *Actj* denotes whether condition *j* is active or inactive, which is encoded as 1 or 0 respectively;
2. If *Vjmin* is larger than *Vjmax* at any time, this element will be regarded as an invalid element.

Figure 2. Encoding mechanism

Each antecedent element represents an attribute, and the consequence element stands for a class. Each chromosome CR_j consists of a set of classification rules R_i $(i=1,2,\ldots,m)$ by concatenation:

$$CR_j = \bigcup_{i=1,m} R_i \qquad j = 1,2,\ldots,s \qquad (1)$$

where *m* is the maximum number of rules allowed for each chromosome, *s* is the population size. 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.

2.2 Genetic Operators

One-point crossover is used in this paper. It can take place anywhere in a chromosome. Referring to the encoding mechanism, the crossover of two chromosomes will not cause inconsistency as all chromosomes have the same structure. On the contrary, the mutation operator has some constraints. Different mutation is available for different elements. 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 select randomly a substitute in the range of that attribute. The rates for mutation and crossover are selected as 0.01 and 1.0.

We set the survival rate or generation gap as 50% (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. Roulette wheel selection is used in this paper as the selection mechanism [25]. 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.

2.3 Fitness Function

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.

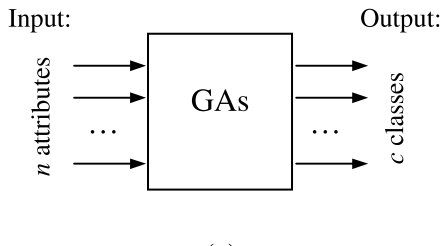
Since there is more than one rule in a chromosome, it is possible that multiple rules matching the conditions for all the attributes but predicting different classes. We use a voting mechanism to help resolve any conflict. That is, each rule casts a vote for the class predicted by itself, and finally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

2.4 Stopping Criteria

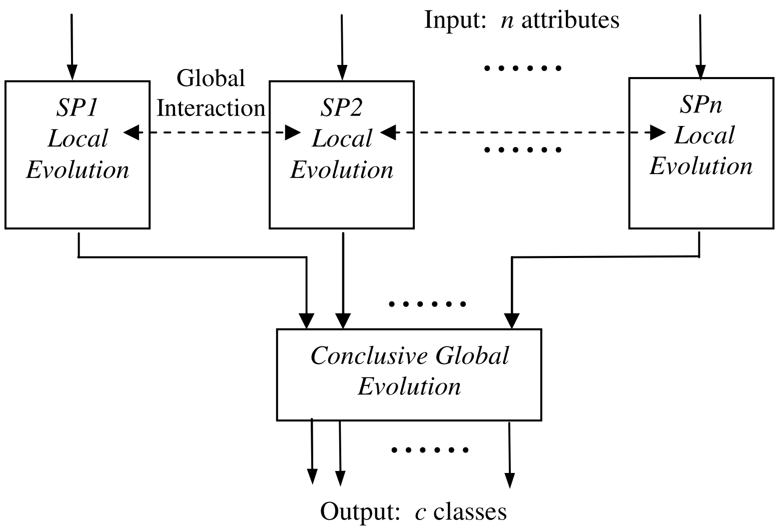
There are three factors in the stopping criteria. The evolution process stops after a preset generation limit, or when the best chromosome's fitness reaches a preset threshold (which is set at 1.0 through this paper), or when the best chromosome's fitness has shown no improvement over a specified number of generations -- *stagnationLimit*. The detailed settings are reported along with the corresponding results.

3. Design of ECCGA

Figure 3 illustrates the concepts of normal GA and ECCGA. As shown in Figure 3(a), a normal 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. ECCGA is significantly different. As shown in Figure 3(b), there are *n* species (SP), each of which evolves the sub-solution for one attribute in classification. The normal GA is employed in each species to advance the local evolution.



(a)



(b)

Figure 3. Illustrations of (a) normal GA and (b) ECCGA

The global interaction in ECCGA can take place in predefined intervals. During such interaction, the global fitness of individuals in species will be assessed. Another enhancement is the introduction of the conclusive global evolution (CGE), which follows the completion of all sub-evolution in species. The objective of CGE is to escape from possible traps by the local optima in the local evolution of species and evolve further to the final solution.

Following the notations presented above, we denote the evolution in each species of ECCGA as:

$$f_i : \quad (X_i) \to C \qquad\qquad i = 1,2,...,n \tag{2}$$

where, f_i is a sub-solution for the sub-problem based on the i-th attribute. X_i is the vector of training patterns with the i-th attribute, and C is the set of output classes.

```
Construct s species each of which dealing with one attribute;
Initialize the fitness elements and prepare the training data for each species;
gen=0;
for each species s
        {  randomly initialize population p(gen);
           evaluate local and global fitness of each individual;
           fitness=(local fitness + global fitness)/2;
           compute the average global fitness;
        }
while (not termination condition)
        {  gen++;
           for each species s
           {   select p(gen) from p(gen-1) based on fitness;
               apply genetic operators to p(gen);
               evaluate local fitness of each individual in p(gen);
               if (gen % genInterval = = 0)
                   {  evaluate global fitness of each individual in p(gen);
                      update the average global fitness;
                   }
               else
                      assign the average global fitness in the previous generation
                      as the global fitness of each individual;
               fitness=(local fitness + global fitness)/2;
           }
Construct initial population by utilizing the chromosomes in all species;
Conduct a normal GA as a conclusive global evolution;
```

Figure 4. Pseudocode of ECCGA

Figure 4 shows the pseudocode of ECCGA. Compared to the CCGA shown in Figure 1, ECCGA has been innovated with new improvement. First, the fitness function is revised. The fitness in CCGA only involves the global fitness, while that of ECCGA consists of two elements, i.e. global fitness and local fitness. The global fitness is obtained with the same method as for CCGA. The local fitness is evaluated on the single attribute in each species. That is, the individual in each species classifies the partially masked training data, where only the training data portion matched with the targeted attribute function will be applied while the portion for the other attributes are treated as non-contributing. The resulting classification rate is recorded as its local fitness. The global and local fitness are then averaged as the representative fitness. In order to save training time, the global fitness is not computed in each generation. Instead, an option for a predefined interval (*genInterval*) is provided. When the evolution in each species advances with a certain number of generations and reaches the preset *genInterval*, the global fitness of each individual will be assessed. Otherwise, the average global fitness in the last generation will be inherited and used as the global fitness element. As normal evolution process advances steadily, the average global fitness will not change abruptly. Therefore, inheriting the average global fitness in the previous generation is an acceptable and reasonable choice. As the evaluation of global fitness involves the chromosome combination process which is time-consuming, it is aimed to save training time to conduct such evaluation in predefined intervals. After all the species reach the termination condition, the evolution will continue with a CGE. An initial population is constructed by combining the randomly selected individuals from the chromosomes in all species, with a definite inclusion of the best global solution recorded during the earlier evolution and the combination of the best chromosomes in all species. A normal GA is then continued until it reaches the stopping criteria. The best chromosome with the highest CR is recorded as the final solution.

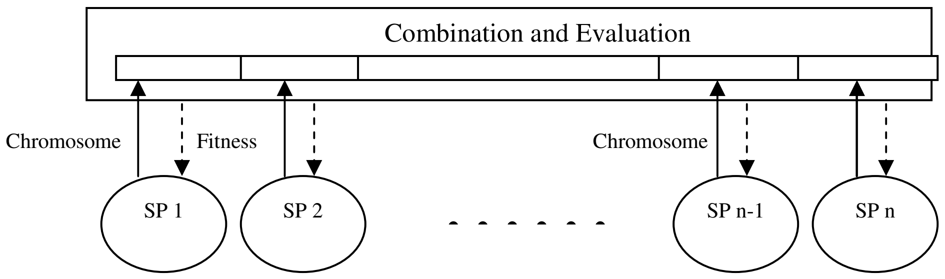


Figure 5. Chromsome combination and fitnesss evalaution

Figure 5 shows how the global fitness is obtained during the global interaction. When the preset *genInterval* is met, the chromosomes from all species are combined, and the resulting chromosome is evaluated and the fitness value will be returned as the global fitness component. In order to add more choices and increase robustness, we adopt the method used in [26]. That is, there are two ways to combine chromosomes coming from species. Either the evaluated individual is combined with the current best individual in each other species, or it is combined with individuals selected randomly from each other species. The two resulting chromosomes are then evaluated and the better of them is returned as the individual's global fitness. This method has been fully explored in [26], and it is found that it performs well in function optimization and it successfully escapes from the local optima resulting from interacting variables.

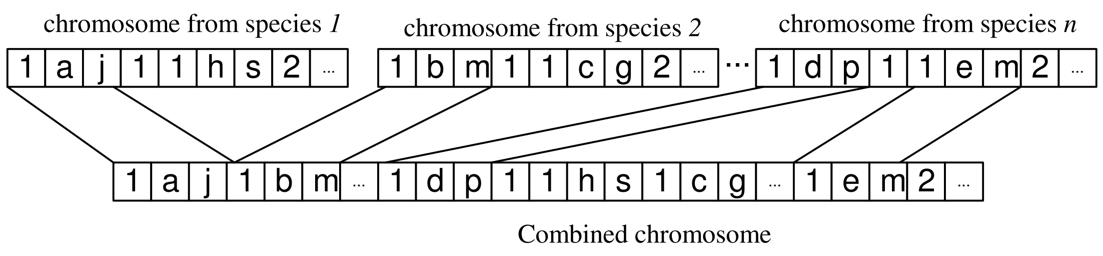

chromosome from species 1
chromosome from species 2
chromosome from species n
1 a j 1 1 h s 2 ...
1 b m 1 1 c g 2 ...
1 d p 1 1 e m 2 ...
1 a j 1 b m ... 1 d p 1 1 h s 1 c g ... 1 e m 2 ...
Combined chromosome

Figure 6. Combining chromsomes from species

Figure 6 shows the format of chromosomes in species and how they are combined to form a full candidate solution. One individual chromosome is selected from each species. Then the attribute elements coming from each species with the same class are combined to form a rule. In this way, rules can be built up in turn to form a combined chromosome – rule set. The resulting chromosome is then evaluated against the full training data, obtaining a global fitness value.

4. Experiments and Analysis

We have implemented several classifiers running on four benchmark data sets, which are the yeast data, glass data, housing data, and diabetes data. They all are real-world problems, and are available in the UCI machine learning repository [27]. The information of these data sets is provided in Table 7 of the Appendix. Each data set is equally partitioned into two parts. One half is for training, while the other half is for testing.

All experiments are completed on Pentium IV 1.4GHz PCs with 256MB memory. The results reported are all averaged over 10 independent runs. The parameters, such as mutation rate, crossover rate, generation limit, stagnation limit etc., are given under the results. We record the evolution process by noting down some indicative results, which include initial classification rate (CR), generation cost, training time, training CR, and test CR. (Their exact meanings can be found in the notes under Table 1.)

Table 1 shows the performance comparison on the yeast data among ECCGA, CCGA, and GA. The improvement percentage compared to the normal GA is also computed. For ECCGA, three different values for *genInterval* have been tried, i.e. *genInterval*=1, 5, and 10. As ECCGAs involve the additional CGE, both the generations and training time comprise two elements. The latter element indicates the additional generations and training time incurred by CGE. Here, the training time for CCGA and the first training time element for ECCGA are the summary of the time cost of all species in a serial implementation. If the evolution in species is implemented in parallel, or run in a multi-processor system with each computing element running for one species evolution, the training time can be tremendously reduced.

Table 1. Performance comparison on the yeast data – ECCGA, CCGA and GA

Summary	**GA**	**CCGA**	**ECCGA (*genInterval*=1)**	**ECCGA (*genInterval*=5)**	**ECCGA (*genInterval*=10)**
Initial CR	0.2356	0.2961	0.2996	0.2911	0.2947
Generations	139.6	91.3	152+104.6	131.5+121.5	106.5+110.5
CPU Time (ms)	592.5	7912.9	10082+454.1	3964.3+486.6	2024.4+469.0
Ending CR	0.3406	0.4147 (21.8%)	0.4771 (40.1%)	0.4493 (31.9%)	0.4156 (22.0%)
Test CR	0.3257	0.3908 (20.0%)	0.4348 (33.5%)	0.4147 (27.3%)	0.3936 (20.8%)

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200;
2. "Initial CR" means the best classification rate achieved by the initial population;
   "Generations" means the number of generations needed to reach the stopping criteria;
   "CPU time (ms)" means the CPU time cost, and its unit is millisecond;
   "Ending CR" means the best classification rate achieved by the resulting population on the training data;
   "Test CR" means the classification rate achieved on the test data;
3. The percentage shown for CCGA and ECCGA is the percent improvement over the normal GA.
4. The other tables follow the same notations as this table.

It is found from Table 1 that all ECCGAs and CCGA outperform the normal GA in terms of training CR and test CR, with a significant improvement around 20% - 40%. ECCGAs also outperform CCGA, which means the enhancement really helps achieve better performance. As for the comparison among the three ECCGAs, we find that ECCGA with *genInterval*=1 achieves the best performance and the performance degrades with the increase of the *genInterval*, which means the more frequent global interaction, the better the final results.

It is also found that training time becomes longer in CCGA and ECCGAs, compared to the normal GA. It is because that the chromosome combination and global fitness evaluation takes more time. Considering the larger improvement on the CRs, this is affordable. Furthermore, we can also find that with a larger value of *genInterval*, the training time can be largely reduced with a small degradation in performance. Therefore, we can choose to set a larger value for *genInterval* when the time cost is an issue.

Table 2. Performance comparison on the glass data – ECCGA, CCGA and GA

Summary	**GA**	**CCGA**	**ECCGA (*genInterval*=1)**	**ECCGA (*genInterval*=5)**	**ECCGA (*genInterval*=10)**
Initial CR	0.3271	0.3710	0.3673	0.3869	0.3673
Generations	98.6	92.1	104.4+88.2	115.1+85.7	96.5+112.6
CPU Time (ms)	77.5	3162.1	1534.6+41.4	1309.9+66.5	622.6 +88.0
Ending CR	0.5262	0.6327 (20.2%)	0.7374 (40.1%)	0.7159 (36.1%)	0.6972 (32.5%)
Test CR	0.3841	0.4112 (7.1%)	0.4701 (22.4%)	0.4654 (21.2%)	0.4374 (13.9%)

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200.

Table 3. Performance comparison on the housing data – ECCGA, CCGA and GA

Summary	**GA**	**CCGA**	**ECCGA (*genInterval*=1)**	**ECCGA (*genInterval*=5)**	**ECCGA (*genInterval*=10)**
Initial CR	0.4553	0.5372	0.5301	0.5364	0.5344
Generations	106.8	80.8	137.8+151.6	88.1+105.1	69.2+93.9
CPU Time (ms)	98.3	7291.6	6168.4+155.4	1950.8+110.9	972.4+93.3
Ending CR	0.6526	0.6941 (6.4%)	0.8506 (30.3%)	0.7700 (18.0%)	0.7364 (12.8%)
Test CR	0.4368	0.4593 (5.2%)	0.5747 (31.6%)	0.4791 (9.7%)	0.4672 (7.0%)

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200.

Table 4. Performance comparison on the diabetes data – ECCGA, CCGA and GA

Summary	**GA**	**CCGA**	**ECCGA (*genInterval*=1)**	**ECCGA (*genInterval*=5)**	**ECCGA (*genInterval*=10)**
Initial CR	0.6297	0.6526	0.6508	0.6521	0.6516
Generations	180.4	70.7	102.5+129.5	79.3+126.2	90.5+158.5
CPU Time (ms)	406.1	4012.3	2809.9+157.9	2288.8+302.8	1674.7+373.1
Ending CR	0.7403	0.7440 (0.5%)	0.7776 (5.0%)	0.7672 (3.6%)	0.7706 (4.1%)
Test CR	0.6935	0.7250 (4.5%)	0.7407 (6.8%)	0.7401 (6.7%)	0.7242 (4.4%)

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200.

The experimental results on the glass, housing, and diabetes data sets are reported in Tables 2, 3, and 4 respectively. Similar results as the yeast data are obtained. That is, ECCGAs outperform CCGA, and CCGA outperforms the normal GA. The improvement percentage over the normal GA is different for each data set. The improvement on the glass and house data is significant, but relatively smaller on the diabetes data. The ECCGA with *genInterval*=1 still achieves the best performance among the three ECCGAs on all three data sets.

5. Analysis and Discussion

In this section, we will have some analysis and discussion on the inner mechanisms of the ECCGA, the issues of parameter selection, and possible further improvement.

First, we investigate the contribution of local fitness in the fitness function. Figure 7 shows a typical run of the ECCGA (*genInterval*=1) and CCGA in one species on the yeast data, excluding the CGE stage for the ECCGA. Both ECCGA and CCGA perform similarly in the earlier stage of evolution. However, it is found that CCGA is trapped later and stagnates around the 110-th generation, while ECCGA advances steadily and reaches a higher CR finally. If we recall the design of ECCGA and CCGA, the only difference here is that CCGA only employs global fitness, while ECCGA involves the local fitness in addition. That means the local fitness of ECCGA will give another opportunity to escape from traps in local optima and advance the evolution further.

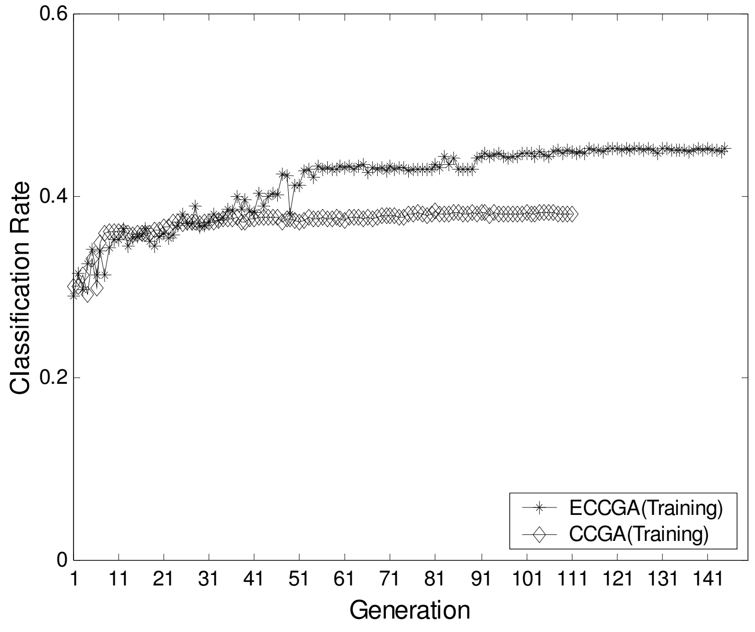


Figure 7. A typical run of ECCGA and CCGA in species on the yeast data

We also have some investigation on the weightage of local and global fitness. As shown in earlier experiments, local fitness and global fitness are averaged as representative fitness. We like to explore further the effect of changing their percentage weights.

Table 5. Performance of different combinations of fitness elements on the yeast data

Summary	**ECCGA (*genInterval*=1) (0.0 vs 1.0)**	**ECCGA (*genInterval*=1) (1.0 vs 0.0)**	**ECCGA (*genInterval*=1) (1/2 vs 1/2)**	**ECCGA (*genInterval*=1) (0.8 vs 0.2)**	**ECCGA (*genInterval*=1) (1/3 vs 2/3)**
Initial CR	0.2928	0.2882	0.2996	0.2950	0.2916
Generations	114.4+75.1	52+153.4	152+104.6	134.5+146.5	165.8+109.3
CPU Time (ms	5177.7+293.7	652.0+620.9	10082+454.1	8720.5+577.4	10830+462.6
Ending CR	0.4349	0.4399	0.4771	0.4755	0.4770
Test CR	0.4048	0.4046	0.4348	0.4332	0.4322

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200;
2. The pair of numbers shown in the column head indicates the weights for the two fitness elements. e.g. (0.8 vs 0.2) means representative fitness=0.8*local fitness + 0.2*global fitness.

Table 5 shows the performance comparison of different combinations of fitness elements. Each column shows one weight percentage combination. We have purposely tried several combinations, including two extremities, i.e. exclusively relying on either pure local fitness (1.0 vs 0.0) or pure global fitness (0.0 vs 1.0) (see note 2 under Table 5 for the meaning of weights). The results show that both selections with extremities receive worse results, while using a more balanced combination of local and global fitness helps achieve better results. The other three weight combinations for these two elements do not show significant difference. It may be due to the randomness of the GA process that smoothens the difference during evolution. As the best combination may vary for different data sets, it is reasonable to use the average weight combination.

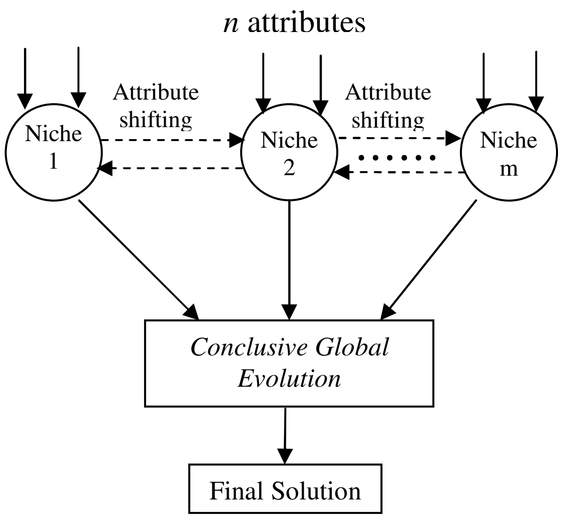


Figure 8. Niche-based ECCGA

In order to improve further the performance, we have proposed a niche-based ECCGA approach. In the current CCGA/ECCGA method, the attribute domain is fully decomposed, i.e. each species dealing with only one attribute. As shown in Figure 8, in the niche-based ECCGA, the original *n* attributes are assigned into *m* niches, and the CCGA is employed in each niche. The conclusive global evolution process combines partial solutions from these niches and evolves further as the final solution.

CCGA/ECCGA is computation-intensive algorithms, as each time when a chromosome needs fitness evaluation, the chromosomes from all subpopulations should be combined together. With the niche-based approach, training time is expected to be shorter, as more evolution and fitness evaluation are conducted locally inside niches. Furthermore, the niche-based attribute decomposition may have some advantage to improve the classification rates. As mentioned earlier in section 1, the linkage phenomenon exists in GA-based learning, which means some attribute may be tightly linked and breaking their linkage may harm the final performance [7] [24]. Therefore, the niche-based ECCGA may avoid linkage-breaking and obtain better performance.

Table 6. Performance of niche-based ECCGA on the yeast data

Summary	**ECCGA (*genInterval*=5)**	**ECCGA-niche (*genInterval*=5) (2-niche)**	**ECCGA-niche (*genInterva*=5) (4-niche)**	**ECCGA (*genInterval*=10)**	**ECCGA-niche (*genInterval*=10) (2-niche)**	**ECCGA-niche (*genInterval*=10) (4-niche)**
Initial CR	0.2911	0.2952	0.3066	0.2947	0.3041	0.2967
Generations	131.5+121.5	137.5+144.8	118+130.4	106.5+110.5	111.2+150.6	119.4+133.1
CPU Time(ms	3964.3+486.6	2365.7+601.8	1565.6+579.8	2024.4+469.0	1314+612.6	963.0+549.1
Ending CR	0.4493	0.4530 (0.8%)	0.4512 (0.4%)	0.4156	0.4496 (8.2%)	0.4282 (3.0%)
Test CR	0.4147	0.4205 (1.4%)	0.4155 (1.9%)	0.3936	0.4170 (5.9%)	0.3984 (1.2%)

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=200.
2. In the 2-niche and 4-niche approaches, attributes are assigned into niches according to their original sequence.

Table 6 shows the results of niche-based ECCGA on the yeast data. Four different scenarios have been tried with selections on different niche number and *genIntervals*. It is found that there is some improvement over their counterparts of normal ECCGA shown in Table 1, although the improvement is minor. Also, with some preliminary experiments, we do not find significant difference on performance with different niche numbers.

It is not easy to determine how to assign attributes into different niches for better performance. We consider it as future work to implement a scheme to explore the best performance using an attribute-shifting method. Under this scheme, the attributes in different niches may migrate in niches after a round of ECCGA, as shown in Figure 8. A new round of ECCGA will restart after attribute shifting, until the stopping criteria are met. The best overall solution is recorded during the whole process. The whole procedure is shown in Figure 9.

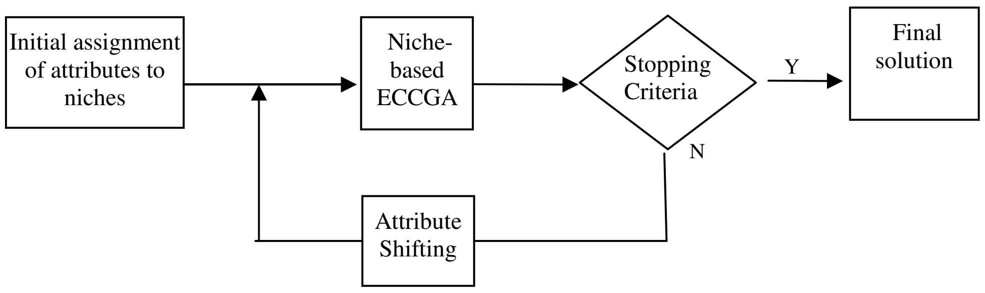


Figure 9. Attribute shifting in Niche-based ECCGA

The niche-based ECCGA may also facilitate incremental learning on new arriving attributes. When new attributes are being introduced into some current niches, we only need to restart the evolution in the corresponding niches, keeping the other niches untouched. The CGE continues to obtain the final solution for the new full set of attributes. It may be achieved in a relatively shorter time [22] [28].

6. Conclusions

In this paper, the cooperative co-evolution scheme is revisited with a rule-based GA system in the application domain of pattern classification. An innovative approach of ECCGA is proposed to improve further the classification performance. In this approach, the original problem is decomposed into several species, each dealing with one attribute only. The evolution in each species advances together based on the evaluation on both local fitness and global fitness. After all species reach convergence, CGE is used for further global evolution.

The simulation results on four benchmark data sets showed that ECCGA outperforms CCGA and the normal GA. The introduction of local fitness and CGE is helpful to the classification performance. Finally, the inner mechanism of ECCGA is analyzed and possible improvement such as the niche-based ECCGA is discussed. The future work involves further exploration on the niche-based ECCGA and its performance in tackling newly arriving attributes.

Appendix

Table 7 lists the number of instance, attributes, and classes in each experimental data set [27]. The yeast problem predicts the protein localization sites in cells. The glass data set contains data of different glass types. The results of chemical analysis of glass splinters (the percentage of eight different constituent elements) plus the refractive index are used to classify a sample to be either float processed or non-float processed building windows, vehicle windows, containers, tableware, or head lamps. The housing data concern housing values in suburbs of Boston. The diabetes data contain the diagnostic data to investigate whether the patient shows signs of diabetes according to World Health Organization criteria such as the 2-hour post-load plasma glucose.

Table 7. Datasets used for the experiments

Data Set	**No. of Instances**	**No. of Attributes**	**No. of Classes**
Yeast	1484	8	10
Glass	214	9	6
Housing	506	13	3
Diabetes	768	8	2

Acknowledgements

The first author is grateful to the Singapore Millennium Foundation for the fellowship awarded.

References

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

[2] J. Grefenstette eds. Genetic Algorithms for Machine Learning, Kluwer Academic Publishers, 1993.

[3] M.V. Fidelis, H.S. Lopes, and A.A. Freitas, Discovering comprehensible classification rules with a genetic algorithm. In Proc. of the 2000 Congress on Evolutionary Computation, vol. 1, 805-810, 2000.

[4] 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., Mass.: MIT press, 91-108, 2001.

[5] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Learning Classifier Systems: from Foundations to Applications, Berlin: Springer, 2000.

[6] J. H. Holland, Adaptation in Nature and Artificial Systems, Ann Arbor: Univ. of Michigan Press, 1975.

[7] G. R. Harik and D. E. Goldberg, Learning linkage through probabilistic expression, Computer Methods in Applied Mechanics and Engineering, 186, 295–310, 2000.

[8] M. Pelikan, D. E. Goldberg, and E. Cantú-paz, Linkage problem, distribution estimation and bayesian networks, Evolutionary Computation, 8 (3), 311-340, 2000.

[9] Q. Chen and S.-U. Guan, Incremental multiple objective genetic algorithms, IEEE Trans. on Systems, Man, and Cybernetics, Part B, vol. 34, no. 3, 1325-1334, 2004.

[10] S.-U. Guan, Q. Chen, and W. Mo, Evolving dynamic multi-objective optimization problems with objective replacement, Artificial Intelligence Review, vol. 23, no. 3, 267 – 293, 2005.

[11] S. Forrest, B. Javornik, R. Smith, and A. S. Perelson. Using genetic algorithms to explore pattern recognition in the immune system, Evolutionary Computation, 1(3), 191--211, 1993.

[12] K. A. DeJong and M. A. Potter, Evolving complex structure via cooperative coevolution, Proceedings of the Fourth Annual Conference on Evolutionary Programming, CA, 1995.

[13] D. Whitley and T. Starkweather, GENITOR II: a distributed genetic algorithm, Journal of Experimental and Theoretical Artificial Intelligence 2, 189-214, 1990.

[14] N. García-Pedrajas, C. Hervás-Martínez, J. Muñoz-Pérez, Multi-objective cooperative coevolution of artificial neural networks, Neural Networks, 15(10), 1259-1278, December 2002.

[15] D. E. Moriarty and R. Miikkulainen, Forming neural networks through efficient and adaptive coevolution, Evolutionary Computation, 5 (4), 373-399, 1997.

[16] S.-U. Guan and S. Li, Incremental learning with respect to new incoming input attributes, Neural Processing Letters, 14 (3), 241-260, 2001.

[17] S.-U. Guan and J. Liu, Incremental ordered neural network training, Journal of Intelligent Systems, 12 (3), 137-172, 2002.

[18] S.-U. Guan and J. Liu, Incremental neural network training with an increasing input dimension, Journal of Intelligent Systems, 13 (1), 43-69, 2004.

[19] A. Potter and K. A. DeJong, The coevolution of antibodies for concept learning, Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, 530-539, Springer-Verlag, 1998.

[20] M. A. Potter and K. A. DeJong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation, 8(1), 1-29, 2000.

[21] S.-U. Guan and F. Zhu,, Incremental learning of collaborative classifier agents with new class acquisition – an incremental genetic algorithm approach, International Journal of Intelligent Systems, 18 (1), 1173-1193, 2003.

[22] S.-U. Guan and F. Zhu, An incremental approach to genetic-algorithms-based classification, IEEE Trans. on Systems, Man and Cybernetics, Part B, 35 (2), 227-239, 2005.

[23] S.-U. Guan and F. Zhu, Class decomposition for GA-based classifier agents – a pitt approach, IEEE Trans. on Systems, Man and Cybernetics, Part B, 34 (1), 381-392, 2004.

[24] F. Zhu and S.-U. Guan, Ordered incremental training with genetic algorithms, International Journal of Intelligent Systems, 19 (12), 1239-1256, 2004.

[25] Z. Michalewicz, Genetic Algorithms + Data Structures =Evolution Programs, 3rd ed. NewYork: Springer, 1996.

[26] M. A. Potter and K.A. DeJong, A cooperative coevolutionary approach to function optimization. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), Parallel Problem Solving from Nature (PPSN III), Berlin: Springer-Verlag, 249-257, 1994.

[27] C. L. Blake and C. J. Merz, UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[28] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, Learn++: an incremental learning algorithm for supervised neural networks, IEEE Trans. on Systems, Man and Cybernetics. Part C, vol. 31, no. 4, 497-508, 2001.