INCREMENTAL EVOLUTION OF CLASSIFIER AGENTS USING INCREMENTAL GENETIC ALGORITHMS

ZHU FANGMING

(B.Eng. & M.Eng. Shanghai Jiaotong University)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgements

I am most grateful to my supervisor, Prof. Guan Sheng-Uei, Steven, for his continuous guidance during my PhD program.

I am truly indebted to the National University of Singapore for the award of the research scholarship, which supports me to finish this research.

I would also like to thank all my family members – my wife, son, parents, and parents-in-law. The warm encouragement from them really supports me to ride out the difficulties. I also present this thesis to my lovely son, who brought me much happiness during the whole process of my thesis writing.

Last but not least, I would like to thank all the fellow colleagues in Computer Communication Network Laboratory, and all the research students under Prof. Guan. My heartfelt thanks goes out to many friends who keep encouraging and helping me.

Contents

Summary

The embodiment of evolutionary computation techniques into software agents has been increasingly addressed in the literature within various application areas. Genetic algorithm (GA) has been used as a basic evolutionary algorithm for classifier agents, and a number of learning techniques have been employed by GA-based classifier agents. However, traditional learning techniques based on GAs have been focused on non-incremental learning tasks, while classifier agents in dynamic environment should incrementally evolve their solutions or capability by learning new knowledge incrementally. Therefore, the development of incremental algorithms is a key challenge to realize the incremental evolution of classifier agents. This thesis explores the incremental evolution of classifier agents with a focus on their incremental learning algorithms.

First, incremental genetic algorithms (IGAs) are proposed for incremental learning of classifier agents in a multi-agent environment. IGAs keep old solutions and use an “integration” operation to integrate them with new elements, while biased mutation and crossover operations are adopted to evolve further a reinforced solution with revised fitness evaluation. Four types of IGAs with different initialization schemes are proposed and compared. The simulation on benchmark classification data sets showed that the proposed IGAs can deal with the arrival of new input attributes/classes and integrate them with the original input/output space. It is also shown that the learning process can be speeded up as compared to normal GAs. This thesis explores the

performance of IGAs in two scenarios. The first scenario explores the condition when classifier agents incrementally learn new attributes, while the other one tackles the case when the classifier agents incrementally learn new classes.

Second, using the IGAs as our basic algorithms, continuous incremental genetic algorithms (CIGAs) are proposed as iterative algorithms for continuous incremental learning and training of input attributes for classifier agents. Rather than learning input attributes in batch as with normal GAs, CIGAs learn attributes one after another. The resulting classification rule sets are also evolved incrementally to accommodate new attributes. The simulation results showed that CIGAs can be used successfully for continuous incremental training of classifier agents and can achieve better performance than normal GAs using batch-mode training.

Finally, in order to improve the performance of classifier agents, a class decomposition approach is proposed. This 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, and results obtained from them are integrated to form the final solution by resolving conflicts. The simulation results showed that class decomposition can help achieve higher classification rate with training time reduced. This thesis further employs a new feature selection technique, Relative Importance Factor (RIF), to find irrelevant features in the input domain. By removing these features, classifier agents can improve classification accuracy and reduce the dimensionality of classification problems.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Software Agents

The term "agent" is used increasingly to describe a broad range of computational entities, although the academia has not reached a generally accepted definition for agents. Some agents may be physically embodied, such as robotic systems that cooperatively manipulate objects in a task environment, whereas others may be computationally coded, which are referred as software agents. In general, 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 (Bradshaw, 1997; Maes, 1994).

Despite some diversity in various applications, some common properties can be identified to make agents different from conventional programs. Each agent might possess to a greater or lesser degree attributes like those enumerated in (Etzioni and Weld, 1995) and (Franklin and Graesser, 1996):

- *Reactivity*: the ability to selectively sense and act;
- *Autonomy*: goal-directedness, proactive and self-starting behavior;
- *Collaborative behavior*: can work in concert with other agents to achieve a common goal;

- *"Knowledge-level" communication ability*: the ability to communicate with persons and other agents with language more resembling human-like "speech acts" than typical symbol-level program-to-program protocols;
- *Personality*: the capability of manifesting the attributes of a "believable" character such as emotion;
- *Adaptability*: being able to learn and improve with experience;
- *Mobility*: being able to migrate in a self-directed way from one host plat-form to another.

There are many classification approaches in the literature. Nwana's classification (Nwana, 1996) classifies the agent types according to the attributes of cooperation, learning, and autonomy. According to their mobility, agents can also be static or mobile. In terms of reasoning model, agents can be deliberative or reactive. Hybrid agents are also common in various applications.

Nowadays, agent-based solutions are explored and applied in many science and engineering applications, such as pattern recognition, scheduling, embedded systems, network management, simulation, virtual reality, etc. In the domain of commercial applications, agent-based e-commerce has emerged and become the focus of the next generation of e-commerce, where software agents act on behalf of customers to carry out delegated tasks automatically (Zhu et al., 2000). They have demonstrated tremendous potential in conducting various tasks in e-commerce, such as comparison shopping, negotiation, payment, etc. (Guan et al., 2000; Guan and Zhu, 2002a; Guan et al., 2002)

Pattern classification plays an important role in various applications such as image processing, information indexing, and information retrieval, and agent-based solutions

for pattern classification have attracted more and more research interests (Vuurpijl and Schomaker, 1998). This thesis explores incremental learning of evolutionary agents in the application domain of pattern classification. These agents are called as classifier agents.

1.2 Evolutionary Agents

It has attracted much attention in the literature to embody agents with some intelligence and adaptability (Smith et al., 2000). Soft computing has been viewed as a foundation component for this purpose. It differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation (Zadeh, 1997). The principal constituents of soft computing are fuzzy logic (FL), neural networks (NN), evolutionary computation (EC), and machine learning (ML) (Nwana and Azarmi, 1997).

Evolutionary computation (EC) is one of the main techniques of soft computing. As a naturally inspired computing theory, EC has already found applications in the development of autonomous agents and multi-agent systems (Smith et al. 1999). Imbuing agents with the ability to evolve their behavior and reasoning capabilities can give them the ability to exist within dynamic domains. EC techniques are good in any situation where agents must deal with many interacting variables that can result in many possible solutions to a problem. The agent's job, in some situations, is to find the optimal mix of values of those variables that produce an optimal solution (Namatame and Sasaki, 1998; Sheth and Maes, 1993; Haynes and Wainwright, 1995).

EC consists of many subcategories, such as evolutionary programming (Fogel et al., 1991), genetic algorithms (Holland, 1975; Michalewicz, 1996), evolution strategies

(Back et al. 1991; Schwefel and Rudolph, 1995), genetic programming (Koza, 1992), etc. Fogel (1995) and Back et al. (1997) provided a comprehensive treatment on the foundation and scope of EC. The most widely used form of evolutionary computation is genetic algorithms (GAs). Specifically, GAs work by maintaining a gene pool of possible solutions - chromosomes. Successive evaluations of the performance of chromosomes regarding some fitness function results in the unfit chromosomes being eliminated. Then mutation and crossover produce new offspring. After some generations, GAs ensure that the fittest chromosome is evolved as the final solution.

GAs have been widely used in the literature to learn rules for pattern classification problems, either through supervised or unsupervised learning, and they have been proved as effective approaches in globally searching solutions for classification problems (Corcoran and Sen 1994; Ishibuchi et al., 1999). In this thesis, genetic algorithms (GAs) are used as the basic evolution tools for classifier agents. On its basis, incremental genetic algorithms (IGAs) are proposed for incremental learning of classifier agents.

1.3 Incremental Learning for Classifier Agents

When agents are initially created, they have little knowledge and experience with relatively low capability. It is advantageous if they have the ability to evolve (Zhu and Guan, 2001a, 2001b; Guan and Zhu, 2002e). Learning is the basic approach for agents to advance the evolution process, hence the selection of learning techniques is important for agent evolution. There are a number of learning techniques being employed by agents in the literature. They can be categorized according to the following criteria: aim of learning, role of agents, and trigger of learning (Liu, 2001).

As the real-world situation is complicated and keeps changing, agents are actually exposed to a changing environment. Therefore, they need to evolve their solutions to adapt to various changes. That is, it should incrementally evolve their solutions or capability by incrementally learning some new knowledge. Another situation may be that the tasks or changes are too complicated, so that the agents may need to evolve incrementally, i.e., step by step. For example, an agent is using certain GA to resolve a new task *t*. As all the individual chromosomes may perform poorly and therefore the GA gets trapped in an unfruitful region in the solution space. If a population is first evolved on an easier task version *t'* and then on task *t*, it may be possible to evolve a better solution.

The term of incremental learning has been used rather loosely in the literature. However, there are some common criteria for an incremental learning algorithm, such as it should be able to learn additional information from new data; it should preserve previously acquired knowledge; it should be able to accommodate new classes that may be introduced with new data (Polikar et al., 2001).

Specifically, incremental learning is also critical for classifier agents. There can be a number of changes occurring for classifier agents in a dynamic environment. For instance, new training patterns may become available; new attributes may emerge; and new classes may be found. In order to tackle these changes, classifier agents need to be equipped with special learning techniques. However, traditional learning techniques based on GAs have been focused on non-incremental learning. It is assumed that the problem to be solved is fixed and the training set is constructed *a priori*, so the learning algorithm stops when the training set is fully processed. On the contrary, incremental learning is an *ad hoc* learning technique whereby learning occurs with the

change of environmental settings, i.e., it is a continuing process rather than a one-shot experience (Giraud-Carrier, 2000). In order to satisfy these requirements, special approaches need to be designed for incremental learning of classifier agents under different circumstances. This motivates the research work of this thesis, where incremental genetic algorithms are proposed for this purpose. In addition, most literature work in classification uses neural networks as tools for incremental learning, while very few employ genetic algorithms. As GAs have been widely used as basic soft computing techniques, the exploration of incremental learning with genetic algorithms becomes more important. This thesis aims to establish an explorative research on incremental learning with proposed IGAs. Through this study, the application domains of GAs can be expanded, as IGAs can cater to more adaptive applications in a changing environment.

Agents are both self-interested and social. Communication between agents enables them to exchange information and to coordinate their activities. Multi-agent systems (MAS) have been established as an important subdiscipline of artificial intelligence. In general, MAS are computational systems in which several semi-autonomous agents interact or work together to perform some set of tasks or satisfy some set of goals (Lesser, 1995; Ferber, 1999; Wooldridge and Jennings, 1995; Jennings et al., 1995).

Learning in single-agent environment and multi-agent environment can be largely different. To date, most learning algorithms have been developed from a single-agent perspective. According to Stone and Veloso (1998), single-agent learning focuses on how one agent improves its individual skills, irrespective of the domain in which it is embedded. But in a multi-agent environment, the coordinated multi-agent learning is a more nature metaphor and may improve the effectiveness. There are two streams of

research about combining MAS and learning. One regards multi-agent systems in which agents learn from the environment where they operate. The second stream investigates the issues of multi-agent learning with a focus on the interactions among the learning agents (Lesser, 1995).

In this thesis, incremental learning is considered in both single-agent and multi-agent environment. However, incremental learning in this thesis has some difference from the above-mentioned multi-agent learning. In conventional approaches, multiple agents coexist in a competitive and collaborative environment. In order to achieve optimal solutions for multiple agents, these approaches concern more about coordination and collaboration among agents. Thus, their research is focused more on the game theory or constraint-based optimization. In this thesis, we make use of the communication and information exchange among agents and explore how they can facilitate incremental learning and boost performance. That is, we explore how agents can benefit from the knowledge provided by other agents, and how agents can adapt their learning algorithms to incorporate new knowledge acquired.

In addition to incremental learning, achieving higher performance for classifier agents is always an ultimate pursuit. In general, classification accuracy and training time are two main metrics for evaluating classifier performance. There are many techniques proposed for this purpose, among which decomposition methods and feature selection have attracted more interests.

The purpose of decomposition methodology is to break down a complex problem into several manageable subproblems. According to Michie (1995), finding a good decomposition is a major tactic both for ensuring the transparent solutions and for avoiding the combinatorial explosion. It is generally believed that problem

decomposition can benefit from: conceptual simplification of the problem, making the problem more feasible by reducing its dimensionality, achieving clearer results (more understandable), reducing run time by solving smaller problems and by using parallel or distributed computation and allowing different solution techniques for individual sub problems. The approach proposed in the thesis is based on the decomposition on the output classes of classification problems. It is shown that the proposed class decomposition approach can improve the classification accuracy with training time reduced. Very little research work has been done for class decomposition with genetic algorithms. In this thesis, the proposed class decomposition approach is applied to not only normal GAs, but also IGAs for incremental learning. This actually increases the adaptability of the decomposition approach, as it can be used in both static and adaptive applications.

A number of features are usually associated with each classification problem. However, not all of the features are equally important for a specific task. Some of them may be redundant or even irrelevant. But they are often unknown *a priori*. Better performance may be achieved by discarding some features (Verikas and Bacauskiene, 2002). In many applications, the size of a data set is so large that learning might not work as well before removing these unwanted features. Reducing the number of irrelevant/redundant features drastically reduces the running time of a learning algorithm and yields a more general solution. This helps in getting a better insight into the underlying concept of a real-world classification problem (Koller and Sahami, 1996; Dash and Liu, 1997). In order to find these irrelevant/redundant features, many feature selection techniques have been proposed. However, these approaches are based on neural networks, and most of them are computation-intensive such as knock- out

techniques. This motivates us to use an approach to determine irrelevant features with small computation cost, and apply it to genetic algorithms. This thesis employs a feature selection technique - relative importance factor (RIF), which was originally proposed in (Guan and Li, 2002b). RIF has proved its effectiveness with NN-based classifiers. This thesis explores further the application of RIF in modular GA-based classifier agents where RIF is used together with the above-mentioned class decomposition approach. It is shown that RIF is effective with modular-GA based approach, and its performance is comparable to that of NN-based solutions.

1.4 Background and Related Work

1.4.1 Genetic Algorithms for Pattern Classification and Machine Learning

Pattern recognition/classification problems have been widely used as traditional formulation of machine learning problems and researched with different approaches including statistical methods (Fukunaga, 1990; Weiss and Kulikowski, 1991), neural networks (Yamauchi et al., 1999; Guan and Li, 2001; Su et al., 2001), fuzzy sets (Setnes and Roubos, 2000), cellular automata (Kang, 2000) and evolutionary algorithms (Ishibuchi et al., 1997; Merelo et al., 2001; Adeli and Hung, 1995). Among evolutionary algorithms, GA-based solutions have become one of the popular techniques for classification. De Jong and Spears (1991) considered the application of GAs to a symbolic learning task -- supervised concept learning from a set of examples. Corcoran and Sen (1994) used GAs to evolve a set of classification rules with real-valued attributes. Bala et al. (1995) introduced a hybrid learning methodology that integrates GAs and decision tree learning in order to evolve optimal subsets of discriminatory features for robust pattern classification. GAs are used to search the

space of all possible subsets of a large set of candidate discrimination features. Ishibuchi et al. (1999) examined the performance of a fuzzy genetic-based machine learning method for pattern classification problems with continuous attributes.

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. GAs have comparatively more explanatory power, as it explicitly shows the evolutionary process of solutions and the solution format is completely decodable.

GAs are widely used in rule-based machine learning (Goldberg, 1989; Grefenstette, 1993). Fidelis et al. (2000) presented a classification algorithm based on GA that discovers comprehensible rules. Merelo et al. (2001) presented a general procedure for optimizing classifiers based on a two-level GA operating on variable size chromosomes. There are two general approaches for GA-based rule optimization and learning (Cordon et al., 2001). The Michigan approach uses GAs to evolve individual rules, a collection of which comprises the solution for the classification system (Holland, 1986). Another approach is called the Pitt approach, where rule sets in a population compete against each other with respect to performance on the domain task (DeJong, 1988; Smith, 1980). Although little is known currently concerning the relative merits of these two approaches, the selection of encoding mechanism will not affect the final solution and performance. In this thesis, the Pitt approach is chosen, as it is more straightforward. Because each chromosome in the Pitt approach represents a candidate solution for a target problem, it facilitates implementation of encoding/ decoding mechanisms and genetic operators such as mutation and crossover.

Moreover, fitness evaluation is simpler than that in the Michigan approach, as fitness value is assigned to a single chromosome, not shared by a group of chromosomes.

One innovative form of the traditional GA is variable-length GA (VGA), where the length of chromosome is not fixed during evolution. VGA is suitable for specific problems where the representation of candidates is difficult to be determined in advance. Srikanth et al. (1995) proposed VGA-based methods for pattern clustering and classification. Bandyopadhyay et al. (2001) combined the concept of chromosome differentiation with VGA, and designed a classifier that is able to automatically evolve the appropriate number of hyperplanes to classify different land-cover regions from satellite images. Incremental genetic algorithm in this thesis is also a type of VGA. For instance, when new attributes or classes are acquired, chromosomes will be expanded in terms of structure and length as a result of the integration of the new attributes or classes. However, the length of chromosome in our approach is still fixed when the number of attributes remains unchanged, and varied when the new attributes or classes need to be integrated.

There is a stream of research called parallel genetic algorithms (PGAs) (Cantu-Paz, 2000b; Melab and Talbi, 2001), which are parallel implementation of GAs. PGAs can provide considerable gains in terms of performance and scalability and they can be implemented on networks of heterogeneous computers or on parallel mainframes. Cantu-Paz (2000a) 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. Melab and Talbi (2001) explored the application of PGAs in rule mining for large databases. There are two main models for PGA - Island model and Neighbourhood model (Cantu-Paz 2000a, 2000b). The first has a number of

subpopulations, each containing a number of individuals. Each subpopulation runs like a canonical GA with some communication (exchange of individuals) between subpopulations. The second model has each individual located on some topography with the restriction that it is only allowed to communicate with its immediate neighbours. The GA with class decomposition approach proposed in this thesis is similar to the method of PGAs, when it is implemented in a parallel model. The distinct feature of our class decomposition is that sub-populations in our approach are all independent, so that there is no migration among them. As a result, training time can be reduced. Moreover, no interaction required among populations for modules allows full-fledged parallel implementation. Our design of class decomposition also ensures that the final solutions are not trapped into local optima. The inner mechanism is that each module needs to not only classify the data with the target classes correctly, but also ensure that data for other classes will not be misclassified into these target classes. The use of intelligent decision rules in the integration step will resolve further the conflicts among sub-solutions.

1.4.2 Incremental Learning and Multi-Agent Learning

Many researchers have addressed incremental learning algorithms and methods in various application domains. Giraud-Carrier and Martinez (1994) created a self-organizing incremental learning model that attempts to combine inductive learning with prior knowledge and default reasoning. New rules may be created and existing rules modified, thus allowing the system to evolve over time. The model remains self-adaptive, while not having to unnecessarily suffer from poor learning environments. Tsumoto and Tanaka (1997) introduced an incremental learning approach to

knowledge acquisition, which induces probabilistic rules incrementally by using rough set technique, and their approach was evaluated on two clinical databases. Ratsaby (1998) presented experimental results for an incremental nearest-neighbor learning algorithm which actively selects samples from different pattern classes according to a querying rule as opposed to the a priori probabilities. It was found that the amount of improvement of this query-based approach over the passive batch approach depends on the complexity of the Bayes rule. Lange and Grieser (2002) provided a systematic study of incremental learning from noise-free and noisy data.

In pattern classification, a wealth of work on incremental learning uses neural networks as learning subjects, and few touch on the use of evolutionary algorithms. Fu et al. (1996) proposed an incremental backpropagation learning network which employs bounded weight modification and structural adaptation learning rules and applies initial knowledge to constrain the learning process. Yamauchi et al. (1999) proposed incremental learning methods for retrieving interfered patterns. In their methods, a neural network learns new patterns with a relearning of a few number of retrieved past patterns that interfere with the new patterns. Polikar et al. (2001) introduced Learn++, an algorithm for incremental training of neural network. Dalché-Buc and Ralaivola (2001) presented a new local strategy to solve incremental learning tasks. It avoids relearning of all the parameters by selecting a working subset where the incremental learning is performed. Other incremental learning algorithms include the growing and pruning of classifier architectures (Osorio and Amy, 1999) and the selection of most informative training samples (Engelbrecht and Brits, 2001). As discussed earlier, some incremental learning algorithms are employed for a complicated problem, learning from an easier task to a more difficult task. Liu et al.

(2001) presented a constructive learning algorithm for feedforward neural networks, employing an incremental training procedure where training patterns are learned one by one. Guan and Liu (2002) presented an incremental training method with an increasing input dimension (ITID). ITID divides the whole input dimension into several sub dimensions each of which corresponds to an input attribute. Neural Networks learn input attributes one after another through their corresponding sub-networks. In this thesis, continuous incremental genetic algorithms (CIGAs) are proposed for incremental training of GA-based classifiers. The incremental training with genetic algorithms has not been addressed in the literature so far. Different from using input attributes in a batch as is done in normal GAs, CIGAs learn attributes one after another. The resulting rule sets are also evolved incrementally to reinforce the final solution. As CIGAs are developed based on IGAs, various approaches are also explored in terms of the corresponding IGA approaches. It is shown that this type of incremental training/learning method can improve classification accuracy.

As mentioned earlier, some multi-agent learning systems are explored with the use of MAS. Enee and Escazut (1999) explored the evolution of multi-agent systems with distributed elitism. It uses classifier systems as the evolution subjects. Caragea et al., (2000) proposed a theoretical framework for the design of learning algorithms for knowledge acquisition from multiple distributed, dynamic data sources. Abul et al. (2000) proposed two new multi-agent based domain independent coordination mechanisms for reinforcement learning.

Learning Classifier System (LCS) (Lanzi, 2000; Takadama et al., 2001) is a machine learning technique which uses reinforcement learning, evolutionary computing, and heuristics to develop adaptive systems. They have been used in various

applications such as knowledge discovery and adaptive expert systems. LCS is designed as a stimulus-response system, which means LCS passively matches messages from the environment and generates actions to modify the environment. In contrast, in addition to passive response, classifier agents in our work are capable of autonomously interacting and collaborating with each other. They are working in a multi-agent environment, which motivates and facilitates collaborative learning. As a result, agents can benefit from such collaboration, and achieve higher performance than in a stand-alone situation.

1.4.3 Decomposition and Feature Selection

Decomposition methods have been used in various fields, such as classification, data mining, clustering, etc. Rokach and Maimon (2002) presented a feature decomposition approach for improving supervised learning tasks. The original set of features is decomposed into several subsets. A classification model is built for each subset, and then all generated models are combined. A greedy procedure is developed to decompose the input features set into subsets and to build a classification model for each subset separately. Weile and Michielssen (2000) explored the application of domain decomposition genetic algorithms to the design of frequency selective surfaces. Masulli and Valentini (2000) presented a new machine learning model for classification problems. It decomposes multi-class classification problems into sets of two-class subproblems which are assigned to non-linear dichotomizers. Apté et al. (1997) presented a new measure to determine the degree of dissimilarity between two given problems, and suggested a way to search for a strategic splitting of the feature space that identifies different characteristics. Watson and Pollack (2000) used

techniques from multi-objective optimization to devise an automatic problem decomposition algorithm that solves test problems effectively.

In artificial neural networks, some class decomposition methods have been proposed for pattern classification. The method proposed in (Anand et al., 1995) is to split a *c*-class problem into *c* two-class sub-problems and each module is trained to learn a two-class sub-problem. Therefore, each module discriminates one class of patterns from patterns belonging to the remaining classes. The method in (Lu and Ito, 1999) divides a *c*-class problem into $\binom{c}{2}$ two-class sub-problems. Each of the two-class sub-problems is learned independently while the existence of the training data belonging to the other *c*-2 classes is ignored. The final overall solution is obtained by integrating all of the trained modules into a min-max modular network. (Guan and Li, 2002a) proposed a simple neural-network task decomposition method based on output parallelism. Incorporated with a constructive learning algorithm, the approach does not require excessive computation and any prior knowledge concerning decomposition. While the above research work has been focused on decomposition methods with neural networks, our class decomposition approach aims to explore its new application based on genetic algorithms which is untouched in the literature. Furthermore, it is not only a direct application to traditional GAs, we also have come up with a new class of IGAs for incremental learning. Our class decomposition is also different from traditional approaches, as an intelligent decision method is used to integrate sub-solutions achieved by those sub-modules. Conflicts are then removed based on some heuristics using the difference of accuracy among sub-modules. Moreover, the time

cost for integration is low and negligible, as the intelligent decision does not require any evolution process.

There are many feature selection techniques developed from various perspectives such as performance (Setiono and Liu, 1997), mutual information (entropy) (Battiti, 1994; Kwak and Choi, 2002), and statistic information (Lerner et al., 1994). Setiono and Liu (1997) proposed a technique based on the performance evaluation of a neural network. In their technique, the original features are excluded one by one and the neural network is retrained and evaluated repeatedly. Pal et al. (2000) demonstrated a way of formulating neuro-fuzzy approaches for feature selection under unsupervised learning. A fuzzy feature evaluation index for a set of features is defined in terms of degree of similarity between two patterns. Yang and Honavar (1998) applied a genetic algorithm to feature subset selection, aiming to improve the effectiveness in the automated design of neural networks for pattern classification and knowledge discovery.

Guan and Li (2002b) proposed two feature selection techniques – relative importance factor (RIF) and relative FLD weight analysis (RFWA) for modular neural network classifiers. They involved the use of Fisher's linear discriminant (FLD) function to obtain the importance of each feature and find out correlation among features. As a new application of RIF (Guan and Li, 2002b), this thesis applies RIF in modular GA-based classifier agents where RIF is used together with the class decomposition approach. It is shown that RIF will be more effective with a modular-GA based classification approach, as it is easier to find irrelevant features in each class module. By removing the irrelevant features detected by RIF in each module, it is illustrated that RIF is effective in finding irrelevant features and can improve

classification accuracy and reduce the complexity of solutions.

1.5 Approaches and Results

The hypotheses of this thesis cover mainly two aspects. It is postulated that incremental learning of classifier agents with GAs is feasible with specially-designed algorithms. Different types of incremental learning algorithms should be designed for various circumstances. It is also postulated that decomposition methods and feature selection techniques coupled with GAs are potential solutions to improve classification performance of GA-based classifier agents. The proposed approaches together with the results obtained confirming with these hypotheses are summarized as follows.

First, this thesis employs GAs as basic learning algorithms and proposes incremental genetic algorithms (IGAs) for incremental learning within one or more classifier agents in a multi-agent environment. IGAs eliminate the need to re-evolve the rule set from scratch in order to adapt to the ever-changing environment. Using IGAs, a classifier agent can fully utilize current knowledge and quickly respond to the changes in environment. IGAs keep old solutions and use an “integration” operation to integrate them with new elements, while biased mutation and crossover operations are adopted to further evolve a reinforced solution with revised fitness evaluation. Four types of IGAs with different initialization schemes are proposed and compared. As IGAs inherit old solutions and use the specially-designed algorithms based on incremental evolution, they can outperform traditional GAs in terms of accuracy and training time. The simulation results on various benchmark classification data sets show that the proposed IGAs can deal with the arrival of new input attributes/classes and integrate them with the original input/output space. It is also shown that the

proposed IGAs can be successfully used for incremental learning and speed up the learning process as compared to normal GAs (Guan and Zhu, 2002b, 2002c).

This thesis explores the performance of IGAs in two scenarios. The first scenario explores the condition when classifier agents incrementally learn new attributes, while the other one tackles the case when the classifier agents incrementally learn new classes (Guan and Zhu, 2003). They are elaborated separately in two chapters.

Second, using IGAs as the basic algorithms, continuous incremental genetic algorithms (CIGAs) are proposed as iterative algorithms for continuous incremental learning and training of input attributes for classifier agents. Rather than using input attributes in a batch as with normal GAs, CIGAs learn attributes one after another. The resulting classification rule sets are also evolved incrementally to accommodate the new attributes. Different approaches of CIGAs are evaluated with four benchmark classification data sets. Their performance is also compared with normal GAs. As CIGAs learn attributes sequentially and the candidate solutions are improved gradually with the introduction of each new attribute, candidate solutions are less likely to be trapped in local optima. As a result, the final classification accuracy will be higher. The simulation results show that CIGAs can be used successfully for continuous incremental learning of classifier agents and can achieve better performance than normal GAs using batch-mode training (Guan and Zhu, 2002d).

Third, to improve the classification performance of classifier agents, a class decomposition approach is proposed. 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.

Two conditions are considered for the use of class decomposition in classifier agents. One is that agents should learn solutions from scratch. The other is that they already have some solutions available, yet still need to evolve their solutions to accommodate new classes. GAs and IGAs are used for these two conditions respectively, and the performance of class decomposition is evaluated based on these two algorithms. As the class decomposition approach breaks up a target problem into several modules, the inter-class interference is reduced. Furthermore, with a specially-designed integration mechanism, the conflicts among sub-solutions obtained from sub-modules are removed without much computation effort. The experiments with four benchmark data sets show that class decomposition can help achieve a higher classification rate with training time reduced (Guan and Zhu, 2004a).

Finally, this thesis further explores the use of feature selection in modular GA-based classifier agents. A new feature selection technique based on relative importance factor (RIF) is employed to find irrelevant features in the feature space. As RIF is employed with class decomposition approach, it is easier to find the irrelevant features (IRFs) in individual class, eliminating the interference from other classes. By removing these irrelevant features from each module, the feature space is reduced and the classifiers can converge to the final solution easily. The experiment results show that RIF can be used to determine the irrelevant features and help achieve higher classification accuracy with feature space reduced. The complexity of the resulting rule sets is also reduced which means the modular classifiers with irrelevant features removed will be able to classify data with a higher throughput (Guan et al., 2004b).

1.6 Structure of this Thesis

This thesis is divided into seven chapters. In this chapter, the background and motivation of this thesis has been addressed, and approaches and results are briefly presented. The remainder of this thesis is organized as follows.

In Chapter 2, the design of rule-based classification and GAs is elaborated. Incremental genetic algorithms (IGAs) are proposed to incrementally learn new attributes in a multi-agent environment. The performance of IGAs is evaluated through experiments with some real-world classification data sets.

Chapter 3 continues the exploration of incremental leaning of classifier agents from another viewpoint of acquiring new classes. GAs and IGAs are still employed as the main techniques to evolve the rule set for classification. IGAs are adapted to incorporate two types of new class acquisition, i.e., class expansion (CE) and rule integration (RI). The performance of IGAs is still investigated through simulation on some real-world classification data sets.

Chapter 4 proposes continuous incremental genetic algorithms (CIGAs) on the basis of IGAs. CIGAs learn input attributes one after another, and the resulting classification rule sets are also evolved incrementally to accommodate the new attributes. Different approaches of CIGAs are evaluated with benchmark classification data sets, and their performance is compared with normal GAs.

A class decomposition approach for GA-based classifier agents is proposed in Chapter 5. The simulation result shows that class decomposition can help achieve higher classification rate with training time reduced.

Chapter 6 proposes a simple feature selection technique - relative importance factor (RIF). RIF is used to find irrelevant features in the input domain for modular GA-based classification. By removing these features, classifier agents aim to improve the classification accuracy and reduce the dimensionality of the classification problems.

Chapter 7 summarizes the work presented in this thesis and indicates some possible future work.

Chapter 2

Incremental Learning of Classifier Agents Using Incremental Genetic Algorithms

2.1 Introduction

Traditional pattern classification work in the literature focuses on batch-mode, static domain, where the attributes, classes, and training data are all determined in advance and the task of learning algorithms is to find out the best rule set which classify the available instances with the lowest error rate (Corcoran and Sen, 1994). However, some learning tasks do not fit into this static model. As the real-world situation is more dynamic and keeps changing, a classifier agent is actually exposed to a changing environment. Therefore, it needs to evolve its solution to adapt to various changes. In general, there are three types of changes in classification problems. First, new training data may be available for the solution to be refined. Second, new input attributes may be found to be possible contributors for a classification problem. Third, new classes may become possible categories for classification. To deal with these types of changes, classifier agents have to learn incrementally and adapt to the new environment gradually. This chapter chooses the arrival of new attributes as the target for incremental learning.

Incremental learning has attracted much research effort in the literature. However, as discussed in Chapter 1, research on incremental learning based on genetic algorithms (GAs) is still open with full challenges. As GAs have been widely used as basic soft computing techniques, the exploration of incremental learning with GAs becomes more meaningful. It will broaden the application domains of GAs, as more and more applications using GAs demand certain incremental algorithms to survive in a changing environment. To achieve incremental learning, GAs will need to be revised accordingly. With a scenario of new attributes being acquired, a classifier agent needs some algorithms to revise its rule set to accommodate the new attributes. That means it should find out how new attributes can be integrated into the old rule set to generate new solutions. Of course, the agent can run GAs from scratch again as some conventional approaches do. However, this approach requires a lot of time and wastes the previous training effort. In some applications with some hard constraints on time and resource, a classifier agent may need to respond quickly in an online manner.

In this chapter, GAs are employed as basic learning algorithms and new approaches called incremental genetic algorithms (IGAs) are proposed for incremental learning. Classifier agents are implemented in a multi-agent environment where the agents can exchange information and benefit each other. IGAs inherit old solutions and integrate them with new elements to accommodate new attributes, while biased mutation and crossover operations are used to further evolve a reinforced solution. Four types of IGAs with different initialization schemes are proposed and compared. The simulation results on benchmark classification problems show that IGAs can be successfully used for incremental learning. IGAs also speed up the learning process as compared to normal GAs.

2.2 Incremental Learning in a Multi-Agent Environment

As discussed earlier, in some classification problems, new training data, attributes and classes may become available or some existing elements may get changed. Thus, classifier agents should have some capability to cope with these changes. Either they may sense the environment and evolve their solutions by themselves, or they may collaborate to adapt to the new environment, as shown in Figure 2.1. There are many possible types of cooperation among a group of agents to boost their capability. Classifier agents can exchange information on new attributes and classes. If available, they can also exchange evolved rule sets (chromosomes). They can even provide each other with new training/testing data, or challenge each with unsolved problems. Various combinations of these operational modes are also possible.

Figure 2.1 also shows the integration of GA and IGA as the main approach for incremental learning, either with self-learning or collaborative learning. Each classifier agent may first use GA from scratch to obtain certain solution (current solution in the figure) based on the attributes, classes, and training data currently known. When new attributes, classes, or data are sensed or acquired from the environment or other agents, IGA is then used to learn the new changes and evolve into a reinforced solution. As long as the learning process continues, the IGA procedure can be repeated for incremental learning.

When designing IGAs for incremental learning, we aim to achieve the following objectives. Firstly, previous knowledge should be preserved and reused if possible, which means IGA works on currently available solutions, instead of working from scratch again. Secondly, the overall performance of a classifier agent should not be

degraded by using IGA. Thirdly, the complexity of IGA should be moderate so that the speed of IGAs can outperform GAs in incremental leaning. The complexity is measured in terms of training time and number of generations in the reported simulations.

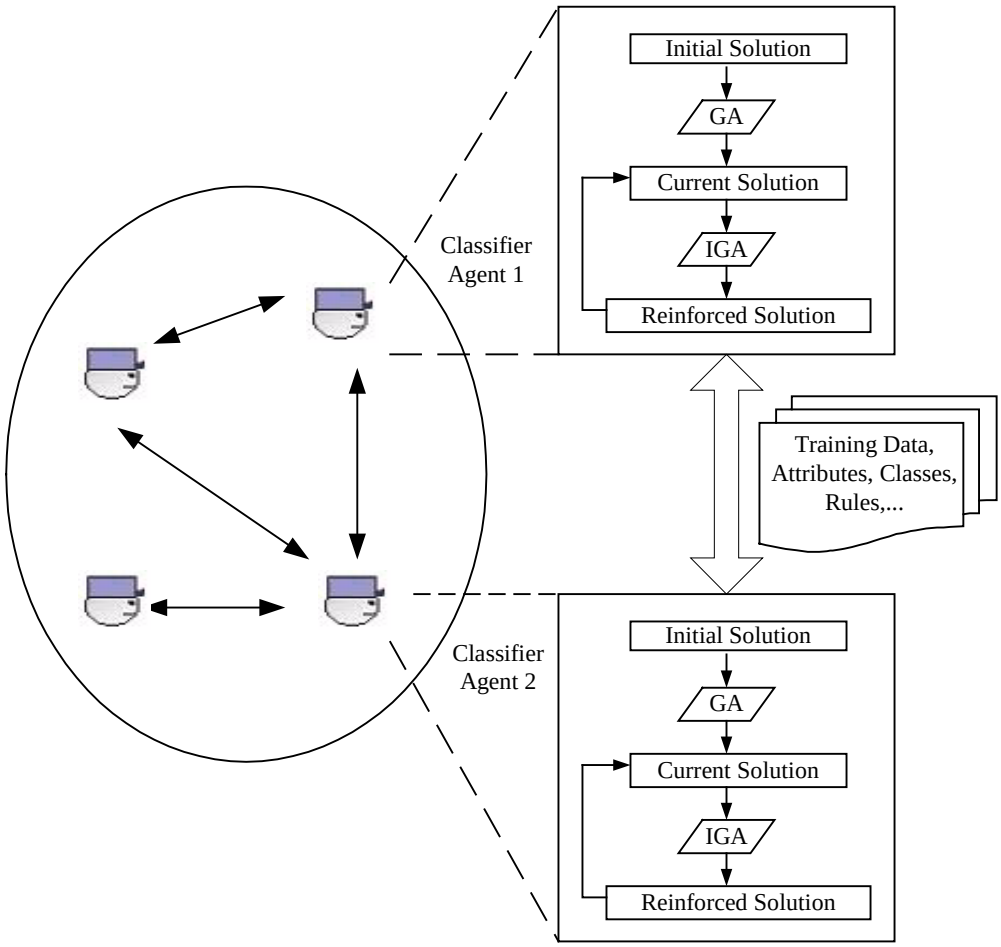


Figure 2.1: Incremental learning of classifier agents with GA and IGA

2.3 GA Approach for Rule-Based Classification

GAs are randomness-search procedures capable of adaptive search over a wide range of search spaces. Rule-based classification has already become a recognized application field for GAs. A typical GA is shown in Figure 2.2.

The task of classification is to assign instances to one out of a set of pre-defined classes, by discovering certain relationship among attributes. Let us assume a pattern classification problem is a *c*-class problem in an *n*-dimensional pattern space. And *p* real vectors $X_i = (x_{i1}, \ x_{i2}, \ ..., \ x_{in}), \ i = 1,2,..., p,$ are given as training patterns from the *c* classes $(c << p)$. Normally, a learning algorithm is applied to a set of training data with known classes to discover the relationship between the attributes and classes. The discovered rules can be evaluated by classification accuracy or error rate either on the training data or test data.

```
begin
  t:=0;
  initialize P(t);    //initialise a population of candidates
  evaluate P(t);    //evaluate each candidate using a fitness function
  while (not terminate-condition) do        //stopping criteria
      begin
        select P'(t) from P(t);   // selection mechanism
        crossover P'(t);     // crossover rate applied
        mutate P'(t);        // mutation rate applied
        combine P'(t) and P(t) to form P(t+1); // survivorsPercent applied
        evaluate P(t+1);
        t:=t+1;
      end
end
```

Figure 2.2: Pseudocode of a typical GA

For classification problems, the discovered rules are usually represented in the following IF-THEN form:

`IF <condition 1>&<condition 2>&...&<condition n> THEN <action>` (2.1)

Each rule has one or more conditions as the antecedent, an action statement as the consequent which determines the class category. There are various representation methods for the conditions and actions in terms of the rule properties (fuzzy or non-

fuzzy) and the attribute properties (nominal or continuous). In this thesis, the nonfuzzy IF-THEN rules are used, as the nonfuzzy rules are adequate to show the main theme of incremental learning and the encoding mechanism is simpler. The adoption of fuzzy rules will be a possible direction for future research.

2.3.1 Encoding Mechanism

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

$$R_i : \text{IF } (V_{1\min} \le x_1 \le V_{1\max}) \wedge (V_{2\min} \le x_2 \le V_{2\max}) \ldots \wedge (V_{n\min} \le x_n \le V_{n\max}) \text{ THEN } y = C \quad (2.2)$$

Where R_i is a rule label, n is the number of attributes, (x_1, x_2,… x_n) is the attribute set, and C is a class. V_{jmin} and V_{jmax} are the minimum and maximum bounds of the j-th attribute x_j respectively. We encode the rule R_i according to the following diagram:

Antecedent Element *1*			……	Antecedent Element *n*			Consequence Element
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 element will be regarded as an invalid element.

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

$$CR_j = \bigcup_{i=1,m} R_i \qquad j = 1,2,...,s \quad (2.3)$$

where m is the 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 the value range for each attribute and class is known *a priori*, V_{jmin}, V_{jmax},

and C can be encoded each with a character by finding their positions in the value ranges. Thus, the final chromosome can be encoded as a string consisting of characters. According to the above encoding mechanism, each chromosome will consist of L characters, where

$$L = m * (3 * n + 1) \tag{2.4}$$

If all the antecedent elements in a rule are inactive, this rule will be regarded as a non-contributing rule.

The above encoding mechanism is similar to the one presented in (Corcoran and Sen, 1994). The new distinctive features are the activeness bit and the representation method with characters. This encoding mechanism is suitable for classification problems whose attributes are real-valued. However, it can be easily extended to classification problems with nominal-valued attributes. For instance, we can build a table to facilitate the conversion between nominal values and continuous values. Thus, the main functions such as encoding, decoding, and mutation can be realized easily by table lookup.

2.3.2 Genetic Operators

Genetic operators such as crossover, mutation, and reproduction play important roles in GAs. One-point crossover is used in this thesis, as it is a standard and common crossover operator and proved to be effective in the literature (Michalewicz, 1996). Referring to the encoding mechanism, it is noted that crossover will not lead to inconsistency and thus can take place in any point of chromosome. On the contrary, the mutation operator has some constraints. The mutation point is randomly selected. According to the position of selected point, it can be determined whether the selected

element 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. This is implemented in such a way that the lower and upper bounds are never exceeded. Figure 2.3 shows the operations of crossover and mutation. The rates for mutation and crossover are specified as mutationRate and crossoverRate. For reproduction, the survival rate is simply set 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.

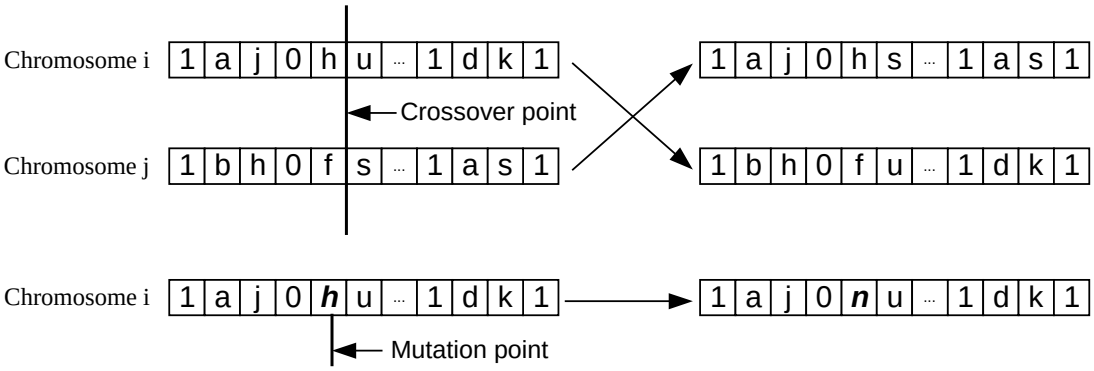


Figure 2.3: Crossover and mutation

Selection mechanism deals with the selection of chromosomes which will undergo crossover and mutation. Some selection mechanisms are deterministic such as *elitism* where just a certain percentage of the best chromosomes are selected. However, most research has shown that stochastic selection biased by fitness is more productive. There are many different ways to implement stochastic selection, and the one used here is called roulette wheel selection (Michalewicz, 1996). In this investigation, the

probability that a chromosome *i* will be selected for crossover and mutation is given by the chromosome's fitness divided by the total fitness of all the chromosomes:

$$P_i = \frac{F_i}{\sum_{k=1}^{p} F_k} \qquad (2.5)$$

where,

P_i : Probability of chromosome *i* selected;
F_i : Fitness of the chromosome *i*;
p : Total number of chromosomes.

This means, during selection, high-fitness chromosomes have a higher probability of producing offspring for the next generation than lower fitness chromosomes.

```
correctNumber:=0;
        for each instance
        { for each rule in the chromosome
                {decode rule antecedents;
                if (all rule antecedents are valid for the instance)
                  then cast a vote for the class reported in the rule ;
                }
            use a voting mechanism to determine the classPredicted;
        if (classPredicted==real class in the instance)
         then  correctNumber++;
        }
fitness:=correctNumber/totalNumber;   //totalNumber is the total number of instances
```

Figure 2.4: Pseudocode for evaluating the fitness of one chromosome

2.3.3 Fitness Function

The fitness of a chromosome reflects the accuracy rate achieved while the corresponding rule set is used for classification. The genetic operators use this information to evolve better chromosomes over generations. Figure 2.4 shows the pseudocode for fitness evaluation. 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:

$$f = \frac{C}{N} = \frac{number\ \ of\ \ instances\ \ correctly\ \ classified}{total\ \ number\ \ of\ \ instances} \qquad (2.6)$$

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. A voting mechanism is used to help resolve 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. (Our observation is that this case rarely happens, therefore it will not hurt the accuracy performance much.)

2.3.4 Stopping Criteria

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

2.4 Incremental Genetic Algorithms (IGAs)

IGAs share the same or similar settings with GAs in many aspects and have some difference in other aspects. This section illustrates the special features of IGAs. Figure 2.5 illustrates the pseudocode of IGAs. The main features of IGAs lie in the formation

of the initial population, integration of old and new chromosomes, biased genetic operators, and a revised fitness function. These features are elaborated in the following subsections.

```
Loop begin
 if new attributes are ready to be integrated
  t:=0;
  if group chromosomes in the old GA/IGA are available
      then select them as seeds;
          else select the chromosome with best fitness  in the old GA/IGA as the seed;
  if new chromosomes for new attributes are available to be integrated
      then integrate them with the old chromosomes;
          else expand old chromosomes with some randomly created elements;
  evaluate P(t); // P(t) is the resulting population from the above steps
  while (not terminate-condition) do        //stopping criteria
        begin
          select P'(t) from P(t);
          biased crossover P'(t);   //  biased crossover rate used
          biased mutation P'(t);    // biased mutation rate used
          combine P'(t) and P(t) to form P(t+1); // based on the survivorPercent rate
          evaluate P(t+1);
          t:=t+1;
        end
Loop end
```

Figure 2.5: Pseudocode of IGAs

2.4.1 Initial Population for IGAs

The formation of initial population is one of the main features of IGAs, in which the integration of old and new chromosomes/elements is one of the major contribution. Figure 2.6 shows how the new elements are inserted into an old rule to form a new rule. Note that it only shows the operation on a single rule for the purpose of simplicity. The other rules in the chromosome will undergo similar operations.

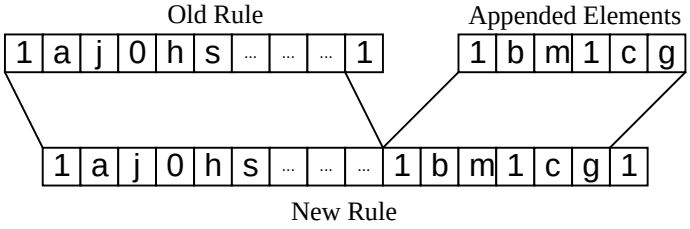


Figure 2.6: Formation of a new rule in a chromosome

There are several ways to construct new chromosome population in terms of the selection of old chromosome(s) and newly appended elements. For the old chromosome(s), either the best rule set (chromosome) is used as a seed for all the initial population of IGA or the whole population of chromosomes in the last generation of the old GA as seeds. To create new elements, two choices are available. Randomly created new elements can be appended to the old rule sets, if the new information acquired or exchanged from the other classifier agent only includes the new attributes and their value ranges. If the other classifier agent can provide more information such as the entire evolved rule set covering the new attributes, it will be more helpful to use the elements from such a rule set. Table 2.1 lists these choices and gives them distinct names for comparison later (IS in the table stands for initialization schemes).

Table 2.1: IGAs alternatives on the formation of a new population

New Elements / Old Elements	Randomly Created Elements	Elements from Acquired Rules
Best Chromosome	IS1	IS2
Group Chromosomes	IS3	IS4

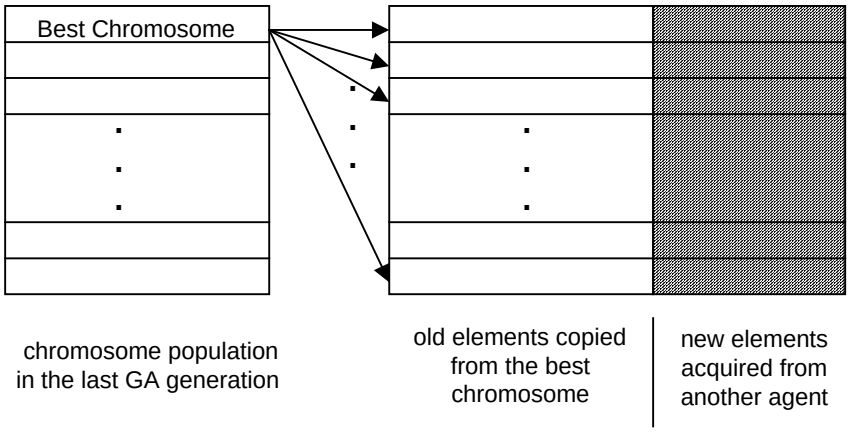


Figure 2.7(a): Illustration for integrating old chromosomes with new elements under IS2

```
IS1:
    for each newChrom[j] in the new population for IGA
    { bestChrom:=the best chromosome from the last generation of GA/IGA;
      bufferChrom:=bestChrom;
     for each rule i in bufferChrom
            {randomly create the activeness bit and bounds for each new attribute;
             create each element with the activeness bit and boundary values;
            insert all the created elements into bufferChrom;
            }
    newChrom[j]:=bufferChrom;
    }
```

```
IS2:
    for each newChrom[j] in the new population for IGA
    { bufferChrom=the best chromosome from the last generation of GA/IGA;
      incomingChrom:=a  chromosome randomly selected from the group
                        chromosomes coming from another agent;
      for each rule i in bufferChrom
            { curClass:=the class of rule i;
              analyze incomingChrom, and  place all new incoming rules having the same class
                    as curClass into a candidate pool;
             randomly select a rule from the candidate  pool;
             insert all  the elements for the new attributes in the selected rule into bufferChrom;
            }
      newChrom[j]:=bufferChrom;
    }
```

```
IS3:
    for each newChrom[j] in the new population for IGA
    { bufferChrom:=the chromosome[j] from the last generation of GA/IGA;
     for each rule i in bufferChrom
            { randomly create the activeness bit and bounds for each new attribute;
             create each element with the activeness bit and boundary values;
            insert all the created elements into bufferChrom;
            }
    newChrom[j]:=bufferChrom;
    }
```

```
IS4:
    for each newChrom[j] in the new population for IGA
    { bufferChrom:=the chromosome[j] from the last generation of GA/IGA;
      incomingChrom:=a chromosome randomly selected from the group
                        chromosomes coming from another agent;
     for each rule i in bufferChrom
            { curClass:=the class of rule i;
            analyze incomingChrom, and  place all new incoming rules having the same class as
                    curClass into a candidate pool;
            randomly choose a rule from the candidate  pool;
            insert all  the elements for the new attributes in the selected rule into bufferChrom;
            }
     newChrom[j]:=bufferChrom;
    }
```

Figure 2.7(b): Pseudocodes for integrating old chromosomes with new elements under IS1 - IS4

Figure 2.7(a) illustrates the formation of a new population using IS2 and Figure 2.7(b) shows the pseudocodes of how the new chromosomes are created by integrating the old and new elements under IS1 to IS4. For instance, IS2 copies from the last generation of GA/IGA the best chromosome into all the new chromosomes, and new elements are selected with a matching mechanism from the incoming chromosomes from the other agent.

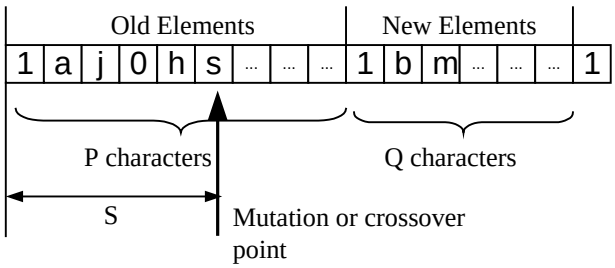
These four IGA approaches are applicable to different environments for incremental learning. If incremental learning happens in a stand-alone agent, this agent can only use IS1 and IS3. However, in a collaborative multi-agent environment, classifier agents gain more freedom on choosing which approach to use. They may choose one of the four approaches according to the environmental situation. We will have a detailed discussion on this later in Section 2.6 based on the experiment results.

2.4.2 Biased Mutation and Crossover

For IGAs, the mutation and crossover operators are biased with preference toward the new elements. Mutation and crossover points are still selected randomly. However, if the point chosen for mutation or crossover is located in the old chromosome part, the corresponding rate may be reduced with a reduction rate, as shown in Figure 2.8. That's why they are called biased rates, and α, β are called mutation and crossover reduction rate respectively.

The motivation behind this is that IGAs tend to preserve the structure of old elements and explore more on the combination between old and new elements. The old elements part still needs variation, but the rates applied can be comparatively lower, compared with the rates exerted on the new elements. Classifier agents can adjust the

values for α and β for different classification problems to achieve better solutions. We have more experiments and discussions on the setting of α and β in Section 2.5.3.



$$P_m^* = \begin{cases} \alpha \cdot P_m & 1 \le S \le P \\ P_m & P < S \le P + Q \end{cases} \qquad P_C^* = \begin{cases} \beta \cdot P_C & 1 \le S \le P \\ P_C & P < S \le P + Q \end{cases}$$

$$0 \le \alpha, \beta \le 1$$

Legends: S - mutation or crossover point selected;
P - length of the old elements; Q - length of the new elements;
α - mutation reduction rate; β - crossover reduction rate;
P_m - mutation rate applied to the new elements in IGA (mutationRate);
P_c - crossover rate applied to the new elements in IGA (crossoverRate);
P_m^* - biased mutation rate applied to the old elements in IGA;
P_c^* - biased crossover rate applied to the old elements in IGA.

Figure 2.8: Biased crossover and mutation rates

2.4.3 Fitness Function and Stopping Criteria for IGAs

The Fitness function is still the same as shown in (2.5). The difference in the fitness evaluation is that the evaluation set will be expanded to include the new attributes and new instances if available. Stopping criteria remain unchanged. The value for generation limit can be reduced, as IGAs will cost less time generally as it preserves existing knowledge. The detailed settings of IGAs are reported along with the results.

2.5 Experiment Results and Analysis

There are many benchmark data sets available for the research community of pattern classification and machine learning. The benchmark data sets used in this thesis are the

wine, glass, cancer, iris, yeast, diabetes, and diabates1 data. The first six are taken from the UCI machine learning repository (Blake and Merz, 1998), and the last one is taken from the PROBEN1 collection (Prechelt, 1994). They all are real-world problems. The basic information of each data set is shown in Table 2.2, which includes the number of attributes, classes, and instances. The detailed information of these data sets can be found in Appendix A.

Several classifier agents are implemented running on four benchmark data sets. First, the feasibility of our GA design is tested by running the GA approach separately. Then, the training and generalization performance of IGAs is evaluated. The different IGA approaches are also compared and the effect of biased mutation and crossover rates is evaluated.

Table 2.2: Details of benchmark data sets used in this thesis

Name of Data Sets	No. of Attributes	No. of Classes	No. of Instances
Wine	13	2	178
Glass	9	6	214
Cancer	9	2	699
Iris	4	3	150
Yeast	8	10	1484
Diabetes/Diabetes1	8	2	768

In order to simulate the process of incremental learning, the data sets are utilized in different ways for different experiments. The detailed information is presented with experiments. All experiments are completed on Pentium III 650MHz PCs with 128MB memory. The values reported in all tables are averaged over ten independent runs. (These apply to all the experiments presented in this thesis.)

2.5.1 Feasibility and Performance of Our GA Approach

The wine data is used to test the feasibility of our GA approach. All the 178 instances are used as training data to evolve the rule set. The GA parameters are chosen as follows: mutationRate=0.01, crossoverRate=1, survivorsPercent=0.5, ruleNumber=60, popSize=500, generationLimit=150. GA is run for 10 independent trials, and we get the final classification rate (CR) value 1.0 for eight times, and 0.994 for two times. Thus, the average CR is 0.999.

Corcoran and Sen (1994) used a real-coded genetic-based machine learning approach to evolve 60 nonfuzzy IF-THEN rules on the wine data. They used a population of 1500 and applied 300 generations with full replacement and reported the results as follows: best CR 100%, average CR 0.995, and worst CR 0.983. Ishibuchi et al. (1999) designed a fuzzy classifier with 60 fuzzy rules by means of an integer-coded GA and grid partitioning. They got the results for the wine data as follows: overall CR 0.994, average CR 0.985, and worst CR is 0.978.

Table 2.3: Comparison of various approaches on the wine data classification

	Corcoran and Sen (1994)	Ishibuchi et al. (1999)	Our GA approach
Rule Number	60	60	60
Generation	300	1000	150
Population	1500	100	500
Best CR	1.00	0.994	1.00
Average CR	0.995	0.985	0.999
Worst CR	0.983	0.978	0.994

As shown in Table 2.3, our approach uses the fewest generations while having a comparable population size. This experiment shows that our genetic approach to

classification problems is feasible and its performance is comparable to other approaches.

2.5.2 Training Performance of IGAs

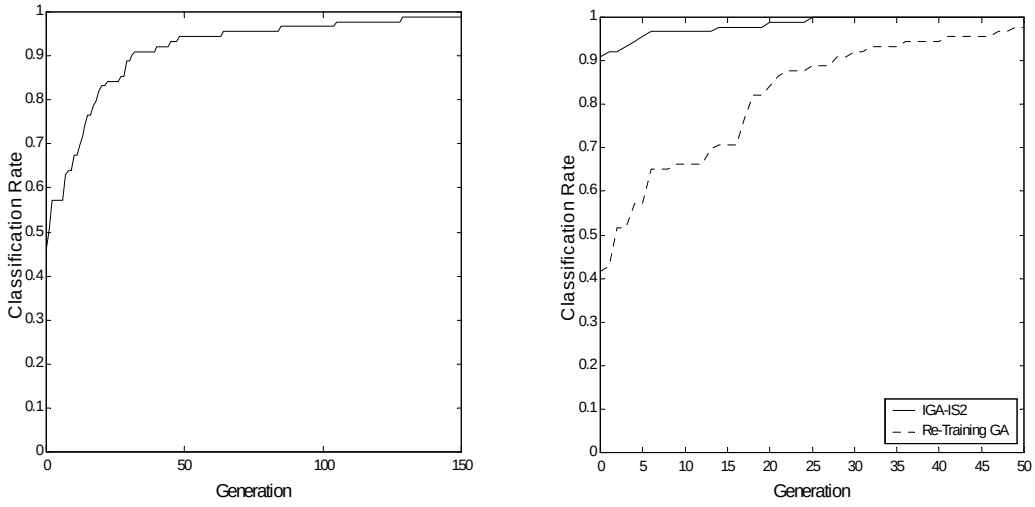
In order to demonstrate the properties of IGAs, the experiment steps are designed as follows.

- First, the whole instances in a data set are partitioned into two groups. One half is used as the old training data, and the other half is used to simulate the newly available data.
- Assume one classifier agent knows only partial attributes at first, and it uses the first half data with the partial attributes as the training data.
- After the agent acquires the new attributes or rule sets from the other agent, IGAs form the new population (There are four different approaches, as listed in Table 2.1). IGAs use the other half data with the complete set of attributes as the training data.
- The procedures of IGAs and biased genetic operators are applied as presented in Section 2.4.

Figure 2.9(a) shows the evolution of rule set with the first 10 attributes in the wine data set. The figure records the best CR in each generation, i.e. the highest fitness value achieved for each generation. It shows that CR grows from an initial value of 0.46 gradually, and finally reaches 0.99 at generation 150.

Figure 2.9(b) shows the IGA process with 13 attributes. IS2 is chosen for the formation of the initial population, which uses the best chromosome in the resultant rule set from (a), and combining it with the rule elements from the other agent. The

retraining GA, which trains the classifier from scratch again with 13 attributes, is also shown in the figure as a comparison.



(a) (b)

Notes:
1. for (a), mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, stagnationLimit=30, generationLimit=150.
2. for (b), α =0.5, β =0.4, generationLimit=50, the rest are the same as in the note 1.

Figure 2.9: Simulation results shows: (a) Classifier agent evolving rule sets with 10 attributes; (b) IS2 running to achieve rule sets with 13 attributes, compared to the retraining GA approach.

It is shown that the best CR from IS2 decreases from 0.99 to 0.91 immediately after the formation of initial population. This can be explained by the facts that new attributes are integrated into the new chromosomes and the new training data are used. Then, CR increases gradually, and reaches 1.0 with 25 generations only. In the case of retraining GA, it costs about 50 generations to reach a CR of 0.99. The simulation results show that IS2 has integrated successfully the new attributes, and evolves a new rule set within a much shorter time compared to retraining GA.

In order to show the advantages of IGAs and explore their performance on integrating different number of new attributes, more experiments have been conducted

on the wine data. Table 2.4 shows the experiment results with various attribute partitions. The upper part shows the results of GA running with the old attributes, and the middle part shows the IGA results on the expanded attributes. The bottom part shows the results of retraining using GA from scratch to accommodate the expanded attributes. Each column in the table represents a different combination of the number of old and new attributes (NOA and NNA). Attributes are used in the same order as they are in the original data set. For instance, NOA=8 means the first 8 attributes are used as old attributes, and NNA=3 means the remaining 3 attributes are used as new attributes.

Table 2.4: Comparison of the performance of IGA on the wine data with various attribute partitions

		NOA =10	NOA=8	NOA=6
GA	Initial CR	0.4574	0.4773	0.5056
	Generations	129.7	139.2	138
	Ending CR	0.9855	0.9844	0.9496
		NOA =10 NNA=3	NOA=8 NNA=5	NOA=6 NNA=7
IGA-IS2	Initial CR	0.8933	0.9212	0.8934
	Generations	33.3	21.8	23.2
	Ending CR	0.9978	1.0	1.0
Retrain. GA	Initial CR		0.4124	
	Generations		49.5	
	Ending CR		0.9865	

Legends:

NOA: number of old attributes; NNA: number of new attributes;

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, stagnationLimit=30, generationLimit=150 (GA), α =0.5, β =0.4, generationLimit=50 (IGA, Retraining GA).
2. "Initial CR" means the best classification rate achieved by the initial population; "Generations" means the number of generations needed to reach stopping criteria; "Ending CR" means the best classification rate achieved by the resulting population.
3. IS2 is used as the method for the formation of initial population.

It is shown that IS2 performs well in integrating various numbers of attributes. When it forms an initial population, the initial CR decreases a little. Finally, IS2 reaches a higher ending CR in much fewer generations. For example, in the case of NOA=6, NNA=7, the initial run of GA achieves an ending CR with 0.9496. IS2 gets an initial CR of 0.8934. Then with only 23 generations, it reaches an ending CR of 1.0. This means that IS2 recovers the information loss caused by missing attributes, and obtains new capability to use the information from the expanded attributes. As listed in the table, if retraining GA is used to retrain from scratch, it will cost about 50 generations, and the resulting CR is inferior to IS2. Therefore, IS2 obviously outperforms retraining.

It is also noted from the table that the ending CRs with incomplete attributes are inferior to the ending CR with complete attributes, i.e. the ending CRs with 6, 8, and 10 attributes can only reach 0.9496, 0.9844, and 0.9855 respectively, which are lower than 0.9980 achieved with 13 attributes, while they cost similar number of generations. It means the contributions made by these attributes are not negligible, although they are small.

In order to demonstrate the power of IGAs further, they are tested on another data set -- the glass data. Three different combinations of the number of old and new attributes are tried, and the results are shown in Table 2.5. This time, IS1 is employed instead of IS2 as the method for the formation of initial population. The results further demonstrate the power of IGAs, as it is verified again that IS1 can integrate successfully the new attributes, and achieve a new solution quickly. It is also shown that IS1 outperforms the retraining GA, as it achieves much better CR than the latter.

Table 2.5: Comparison of the performance of IGA on the glass data with various attribute partitions

		NOA=8	NOA=7	NOA =6
GA	Initial CR	0.367	0.3598	0.3653
	Generations	134.9	131.3	145.7
	Ending CR	0.6363	0.6372	0.6235
		NOA=8 NNA=1	NOA=7 NNA=2	NOA =6 NNA=3
IGA-IS1	Initial CR	0.429	0.425	0.4299
	Generations	78.9	78.6	75.1
	Ending CR	0.6441	0.6424	0.6423
Retrain. GA	Initial CR		0.3551	
	Generations		80.0	
	Ending CR		0.5636	

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, stagnationLimit=30, generationLimit=150 (GA), α =0.5, β =0.4, generationLimit=80 (IGA, retraining GA).
2. IS1 is used as the method for the formation of initial population.

Table 2.6: Comparison of the performance of IGA on the diabetes data

		NOA=7 (excluding attr. 2)	NOA=7 (excluding attr. 4)	NOA=7 (excluding attr. 6)	NOA=7 (excluding attr. 8)
GA	Initial CR	0.6428	0.6489	0.6448	0.6406
	Generations	223.5	207.4	214.7	233.5
	Ending CR	0.7151	0.7664	0.7645	0.7641
		NOA=7 NNA=1 (attr. 2)	NOA=7 NNA=1 (attr. 4)	NOA=7 NNA=1 (attr. 6)	NOA=7 NNA=1 (attr. 8)
IGA-IS1	Initial CR	0.6641	0.7391	0.7302	0.6969
	Generations	115.3	115.8	115.9	116.6
	Ending CR	0.7860	0.7899	0.7875	0.7896
Retrain. GA	Initial CR		0.6388		
	Generations		104.2		
	Ending CR		0.7737		

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30, generationLimit=250 (GA), α =0.5, β =0.4, generationLimit=120 (IGA, retraining GA).
2. IS1 is used as the method for the formation of initial population.

Table 2.6 reports the experiment results on the diabetes data. As shown in the table, NOA and NNA are kept unchanged as 7 and 1, but a different attribute is selected as the new attribute in each experiment, i.e., attribute 2, 4, 6, and 8 are selected as the new attribute respectively. For example, the last column means that attributes 1 to 7 are chosen as the old attributes, and attribute 8 as the new attribute for IS1. Apart from similar findings obtained like in the last two experiments, we get another more interesting finding. It is noted that if attribute 2 is excluded from the input attributes, the ending CR from using the other seven attributes is only 0.7151, which is apparently lower than the other three cases (about 0.765). In other words, CR has dropped about 0.05 more if attribute 2 is removed. It tells us that attribute 2 is an important attribute in the diabetes data, as it contributes more than attribute 4, 6, and 8. The results also show that IS1 recovers successfully the larger CR loss caused by the missing attribute (2), and achieves a similar ending CR as the other three settings.

2.5.3 Generalization Performance of IGAs

In the above experiments, the whole data set is used only for training. In order to test the generalization performance of the classifier agents, each data set is partitioned into three parts, namely, TRA, TRB, and TST, with respective percentage of 35%, 35%, and 30%. TRA is used to simulate the partial attributes, TRB is used to simulate the full set of attributes, and TST is used for all instances as unknown test data. Therefore, when the classifier knows only partial attributes, it uses TRA and TST as training data and test data respectively. When it learns the full set of attributes, TRB and TST are used instead.

Table 2.7: Comparison of the performance of IGAs on the wine data

NOA=8, NNA=5	**GA**	Initial CR		0.4429		
		Generations		106.2		
		T. time (s)		175.0		
		Ending CR		0.9556		
		Test CR		0.7788		
			Retraining GA	IS1	IS2	IS3	IS4
	Retrain. GA & IGAs	Initial CR	0.4365	0.6667	0.8000	0.6603	0.8175
		Generations	46.3	36.1	30.2	34.5	30.5
		T. time (s)	97.0	78.5	66.6	74.6	67.0
		Ending CR	0.9365	0.9952	1.0	1.0	0.9984
		Test CR	0.8192	0.8558	0.8615	0.8519	0.8769
NOA=9, NNA=4	**GA**	Initial CR		0.4444		
		Generations		101.0		
		T. time (s)		162.5		
		Ending CR		0.9825		
		Test CR		0.8096		
			Retraining GA	IS1	IS2	IS3	IS4
	Retrain. GA & IGAs	Initial CR	0.4365	0.7063	0.7810	0.7	0.8000
		Generations	46.3	41.1	35.7	42.1	31.1
		T. time (s)	97.0	81.9	70.8	83.5	61.9
		Ending CR	0.9365	0.9921	0.9921	0.9905	0.9952
		Test CR	0.8192	0.8558	0.8865	0.8635	0.8769
NOA=10, NNA=3	**GA**	Initial CR		0.4397		
		Generations		105.5		
		T. time (s)		176.9		
		Ending CR		0.9746		
		Test CR		0.8019		
			Retraining GA	IS1	IS2	IS3	IS4
	Retrain. GA & IGAs	Initial CR	0.4365	0.7508	0.8032	0.7397	0.8254
		Generations	46.3	42.4	35.8	37.4	32.3
		T. time (s)	97.0	85.5	72.7	74.8	64.9
		Ending CR	0.9365	0.9905	0.9984	0.9952	0.9968
		Test CR	0.8192	0.8558	0.8577	0.8327	0.8481

Notes:

1. The meaning of various IGAs can be referred to Table 2.1.
2. "T. time (s)" means the training time cost, in seconds;
   "Test CR" means the classification rate achieved by the resulting population on the test data. Other legends follow the explanations under Table 2.4.
3. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30, generationLimit=150 (GA), α =0.5, β =0.4, generationLimit=50 (IGA, retraining GA).

Table 2.8: Comparison of the performance of IGAs on the cancer data

NOA=6, NNA=3	**GA**	Initial CR			0.8344	
		Generations			89.9	
		T. time (s)			349.5	
		Ending CR			0.9746	
		Test CR			0.9678	
			Retraining GA	IS1	IS2	IS3	IS4
	Retrain. GA & IGAs	Initial CR	0.7660	0.9193	0.9422	0.9316	0.9426
		Generations	80.0	72.2	58.1	69.3	63.0
		T. time (s)	347.4	324.1	258.4	315.8	287.6
		Ending CR	0.9873	0.9807	0.9766	0.9811	0.9799
		Test CR	0.9479	0.9502	0.9521	0.9569	0.9583
NOA=4, NNA=5	**GA**	Initial CR			0.8582	
		Generations			91.4	
		T. time (s)			294.8	
		Ending CR			0.9689	
		Test CR			0.9578	
			Retraining GA	IS1	IS2	IS3	IS4
	Retrain. GA & IGAs	Initial CR	0.7660	0.9074	0.95	0.9012	0.9508
		Generations	80.0	73.2	61.5	68.7	65.3
		T. time (s)	347.4	302.7	260.0	293.8	277.6
		Ending CR	0.9873	0.9836	0.9779	0.9816	0.9799
		Test CR	0.9479	0.9649	0.9664	0.9597	0.9597

Notes:
1. The meaning of various IGAs can be referred to Table 2.1.
2. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30, generationLimit=200 (GA), α =0.5, β =0.4, generationLimit=80 (IGA, Retraining GA).

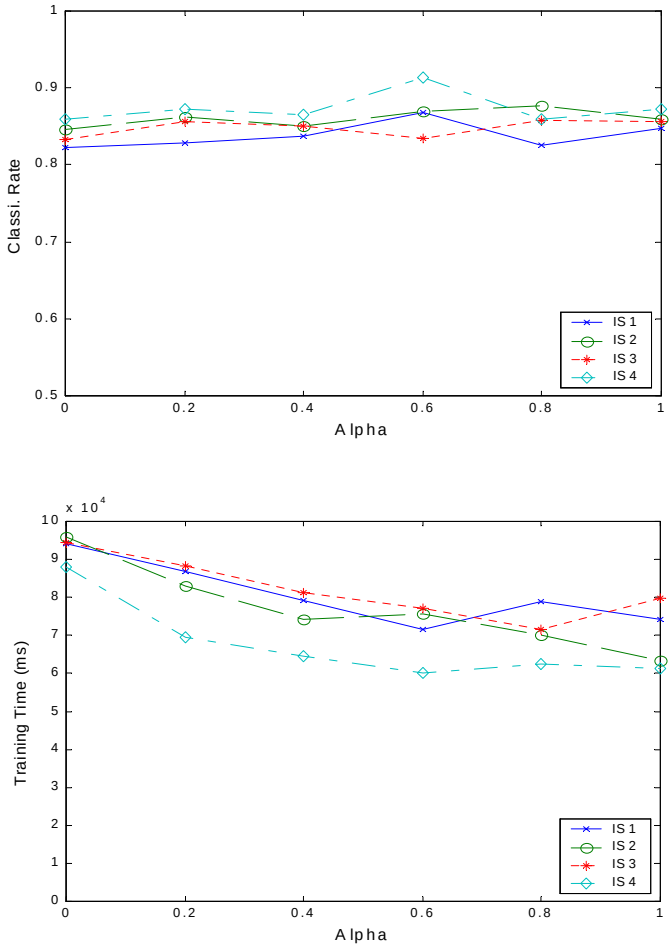
Table 2.7 shows the comparison among the four IGA approaches (listed in Table 2.1 earlier) using the wine data. GA is run with a certain number of attributes first (8, 9, 10 attributes as shown in the table). With the results of GA, four IGA approaches are employed. Therefore these four approaches have the same starting point, which is fair for comparison. The results show that IS2 and IS4 cost fewer generations and thus less training time to reach convergence than IS1 and IS3. This may be explained by recalling the method each approach uses to form the initial population. As IS2 and IS4 use the evolved rule set from the other agent, they can acquire more useful information from the rule set than IS1 and IS3 with the randomly created elements. This can also

be verified by observing the initial CR for IGAs, i.e., the initial CRs of IS2 and IS4 are apparently higher than those of IS1 and IS3. Comparing the resulting test CR of these four approaches, it is found that either IS2 or IS4 achieves the best test CR. This again shows that using evolved rule sets helps achieve better performance.

Table 2.7 also compares performance between the four IGAs and the retraining GA. It is found that all IGAs cost fewer generations (shorter training time) and achieve better performance (training and test CRs) than the retraining GA, which further demonstrates the advantages of IGAs.

Table 2.8 compares the performance of various IGAs with another data set – the cancer data. Similar findings are obtained as those from Table 2.7. It is shown again that the four approaches meet the requirements of incremental learning with different performance. IS2 or IS4 again achieves the best test CR among the four approaches and needs fewest generations (least training time) to reach convergence. The comparison between IGAs and retraining GA also shows the similar findings as with wine data. IGAs outperform retraining GA in training time and test CR.

Finally, several experiments are conducted with different settings of the mutation and crossover reduction rates α and β to explore their effects on the performance of IGAs. When one rate is evaluated, the other rate is fixed as 1.0. Figure 2.10 and 2.11 show the results on the wine data with different values of α and β respectively. $\alpha = 0$ or $\beta = 0$ means there is no operation (mutation or crossover) on the old elements, and $\alpha = 1$ or $\beta = 1$ means there is no bias on mutation or crossover between the old elements and new elements.

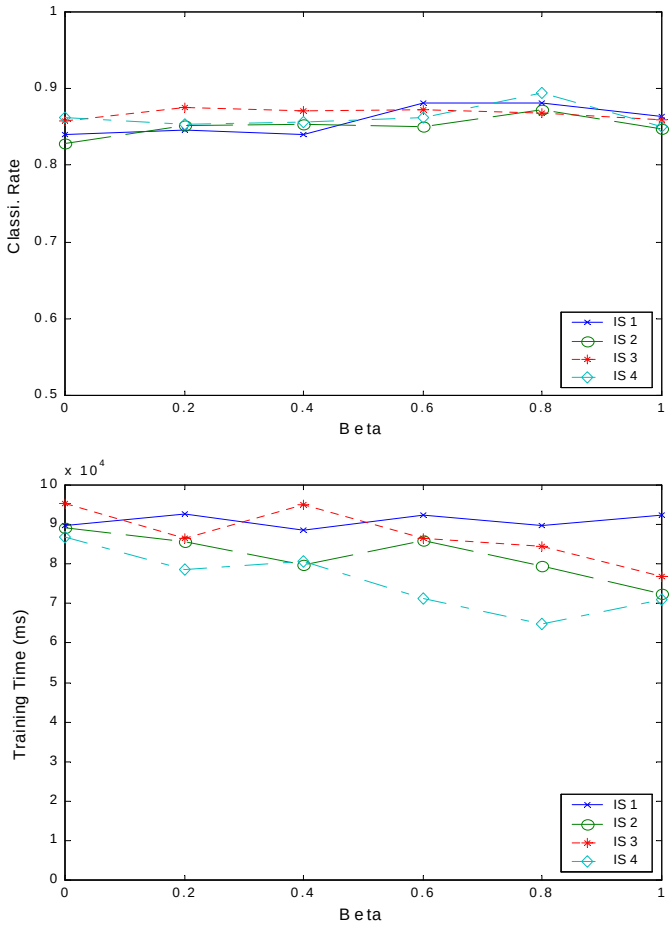


Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30, generationLimit=150 (GA), generationLimit=50 (IGA).
2. NOA=10, NNA=3.

Figure 2.10: Effect of mutation reduction rate α on the performance of IGAs (test CR and training time) with the wine data

It is found that α and β really affect the performance of IGAs. Both Figures show that if IGAs are used with $\alpha = 0$ or $\beta = 0$, they need the longest training time and achieve lower test CRs compared to other values for α and β. This tells us that the extremely biased rates ($\alpha = 0$ or $\beta = 0$) are not suitable for IGAs and the old elements still need some genetic operations during IGAs. It is also shown in both figures that the

test CR increases and training time decreases in a general trend, when the values of α and β increase. It is noted that $\alpha = 1$ or $\beta = 1$ is not the best parameter. In contrast, the best parameter values for α and β are between 0.6 to 0.8, depending on the type of IGAs. This result supports the use of reduced crossover and mutation rates on the old elements.



Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30, generationLimit=150 (GA), generationLimit=50 (IGA).
2. NOA=10, NNA=3.

Figure 2.11: Effect of crossover reduction rate β on the performance of IGAs (test CR and training time) with the wine data

2.5.4 Analysis and Explanation

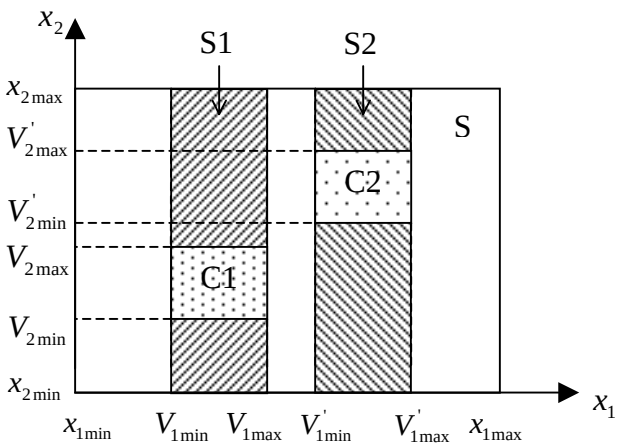


Figure 2.12: Analysis model for a simplified classification problem

It is found from the experiment results that IGA approaches outperform the retraining GA. The advantages of IGA approaches mainly exist in two aspects, namely, shorter training time and higher classification rates. In this section, we give some analysis and explanation on the inner mechanisms leading to the advantages of IGA approaches.

Figure 2.12 presents a model for a simplified classification problem with two attribute (x_1, x_2) and two classes (C_1, C_2). This simplified model and following analysis can be easily extended to higher-dimensional spaces. S denotes the possible area confined by the value range of ($x_{1\min}$, $x_{1\max}$) and ($x_{2\min}$, $x_{2\max}$), which is also the maximum searching area for GA. C_1 and C_2 are the areas covering the training instances belonging to the two class categories respectively. When the retraining GA evolves the rule set using two attribute (x_1 and x_2) together, it searches the area S directly. However, IGA approaches consist of two steps. The first step is a one-dimensional search along the axis x_1, trying to find the boundary information for both

classes, i.e. $V_{1\min}$, $V_{1\max}$, $V'_{1\min}$, $V'_{1\max}$. The second step inherits these boundary information, and continue searching the boundary information for the two attributes (further searching $V_{2\min}$, $V_{2\max}$, $V'_{2\min}$, $V'_{2\max}$). However, with the help of the inherited information, the second search step can be confined in the areas $S1$ and $S2$ (the shadow area plus the grid area for each as shown in the figure). It is easy to see that the search space is largely reduced from S to $S1 \bigcup S2$. Even the effort of the one-dimensional search in the first sep is added, the IGA approaches still search smaller area than the retraining GA. This explains why the IGA approaches need less training time. That is, as the IGA approaches inherit old information, they stand on a better starting point.

Figure 2.12 also provides some information for the reason of the improvement on the classification rates. As the initial population for IGAs is created using the boundary information of x_1, they are located in the area $S1$ or $S2$. Because they are close to the best solutions C_1 or C_2, it is more likely that the IGA approaches converge to either solution. However, the retraining GA needs to search a larger area S, thus it may miss the best solutions occasionally. As shown in the Figure 2.12, it is relatively more difficult for the retraining GA to derive such solution contour. As a result, the partitioning in the attribute domain brings along some advantages. In general, the interference among intervened attributes makes GA search more difficult. When a larger attribute domain is partitioned, the interference among attributes can be reduced. Therefore, it is easier to map partial attributes to classes, which makes GA search easier and more accurate.

We continue to analyze the contributions of two reduction rates α and β. Still learned from Figure 2.12, IGA approaches should stick to the $S1$ neighborhood for the old attribute x_1, while exploring more for the boundary information on the new attribute x_2. As the exploration is more focused on the new attribute, the training time can be further saved. This analysis supports the use of reduction rates, i.e. $\alpha < 1$ and $\beta < 1$. Meanwhile, as the real classification problems are more complicated than the simplified one in the Figure 2.12, the two areas $S1$ and $S2$ may have irregular contour shape and may overlap sometimes. Therefore, the exploration on the boundary information of the old attribute should not be stopped entirely. This implies $\alpha > 0$ and $\beta > 0$. Thus, we have $0 < \alpha < 1$ and $0 < \beta < 1$, and the optimal values for the reduction rates exist in between. The experiment results in Figures 2.10 and 2.11 have confirmed this, and it is found the optimal values for α and β differ according to the IGA approaches used. With the selection of α and β, the generation cost to achieve the stopping criteria can be reduced, thus the training time will be saved accordingly.

The advantages of IGAs and the justification for α and β can also be explained with the schema analysis and building block hypothesis (Holland, 1975; Goldberg 1989). A schema is a similarity template describing a subset of strings with similarities at certain string positions. It is postulated that an individual's high fitness is due to the fact that it contains good schemata. Short and high-performance schemata are combined to form building blocks with higher performance expected. Building blocks are propagated from generation to generation, which leads to a keystone of the GA approach. Research on GA has proved that it is beneficial to preserve building blocks during the evolution process (Goldberg 1989). IGAs inherit the old chromosomes from

the previous results, where the building blocks likely reside. The integration of these building blocks into the initial population for IGAs provides a solid foundation for the following evolutions. Also, the smooth preservation of these building blocks during the following evolutions also boosts the classification performance. This justifies the use of α and β. When the crossover and mutation rates are reduced in the old elements portion, the building blocks inside will undergo less genetic evolution pressure and thus increase their survival chance.

2.6 Discussions and Refinement

The literature in GA has provided extensive discussions on how traditional GA prevents from falling local minima (Holland, 1975; Goldberg 1989; Michalewicz, 1996). It has been shown that "the crossover leads the program towards better solutions while mutation prevents it from getting stuck with local minimum" (Goldberg 1989). Crossover biases the search towards promising regions of the search space by sticking together useful building blocks, and mutation serves to prevent premature loss of population diversity by randomly sampling new points in the search space. Compared with other methods such as hill-climbing, GA is extremely unlikely to be trapped in local minima, as crossover and mutation bring in variance and new offspring (Michalewicz, 1996). The selection of crossover and mutation rates is important. The rule of thumb is that the mutation rate is kept small, while crossover is much larger. We have chosen the commonly-use rates for these two rates, i.e. 1.0 for the crossover rate and 0.01 for the mutation rate. We have also carefully observed the results of GA/IGA experiments, and found that they never fall into local minima.

Several real-world data sets have been employed. They have different properties including the number of attribute, classes, and training instances. Therefore, some data sets, e.g. the cancer data, are easier to classify, making their classification rates close to 1.0, while other data sets, such as the diabetes data and glass data, are more difficult to classify, and their final classification rates are about 0.6. According to the results reported, the improvement on the classification rates with these data sets is still large (about 14%), apart from the obvious improvement on the training time. Therefore, we can draw the following conclusions from the overall experiment results: our IGA approaches both improve the classification rates and save the training time. The saving of training time is obvious, while the level of improvement on the classification rates depends on the data sets' properties. For some data sets whose classification rates are close to 1.0, the improvement still exist but may become small.

There can be many variations for the algorithms and experiment settings to deal with different environmental situations. This section discusses some possibilities which include more IGA alternatives, the expansion of population size to facilitate the integration of new and old elements, the algorithms for a single agent to accommodate new attributes, and the special case when new patterns are available with new attributes only.

Four IGA alternatives have been proposed and evaluated in this chapter. Actually, there can be more options for the integration of old and new elements. For example, the elements from the acquired rules can also be available as the best chromosome, group chromosomes, or even some percentage of the group chromosomes. There still has room for future research to explore these combinations.

As discussed earlier, before IS2 and IS4 are employed, the old and new elements have already been evolved in two classifier agents. When they are integrated, there are many options in producing the offspring by integration. In the current experiments, same population size is chosen for GAs and IGAs. In order to explore more on integration, the population size is doubled to accommodate more resulting offspring chromosomes from the integration of old and new elements. Then the offspring chromosomes are sorted in a descending order of fitness. The fitter half will survive as the initial population for IS2 or IS4. The other half will be discarded. The experiments with this refinement have shown that the resulting test CR can be improved in most cases. For instance, the test CR for wine data can be improved by about 2%. IS2 and IS4 still outperform IS1 and IS3. Therefore, the integration mechanism really affects the final classifier performance. As for the training time, it will be a little longer because of the longer integration process. However, it can be neglected, as the succeeding training time will dominate the whole IGA process.

From the results of IS1 to IS4, it is found that the performance of IS2 and IS4 are generally better than IS1 and IS3. We have considered the application of IGAs in a multi-agent environment, which means IS2 and IS4 are used only when two or more classifier agents are exchanging information on new attributes and instances. However, IS2 and IS4 can also be used in the situation of one single agent. When such an agent learns that new attributes are available, it can create the elements for these attributes first, then independently evolve these elements, as if it was done in other agents. Finally, these elements can be integrated with the old ones using IS2 or IS4. With this method, the final performance will be better than that of randomly creating elements with IS1 or IS3. Therefore, the IGA algorithm can be refined for a single agent to

achieve better performance by replacing the process of randomly creating new elements with a separate evolution process for new elements before they are integrated, as shown in Figure 2.13.

```
Loop begin
if new attributes are ready to be integrated
   t:=0;
   if group chromosomes in the old GA/IGA are available
        then select them as seeds;
             else select the chromosome with best fitness  in the old GA/IGA as the seed;
   if new chromosomes for new attributes are available to be integrated
        then integrate them with the old chromosomes;
             else  randomly create new elements for new attributes;
                   evolve the new elements using conventional GA;
                   integrate the evolved new elements with the old chromosomes;
    evaluate P(t); // P(t) is the resulting population from the above steps
   while (not terminate-condition) do        //stopping criteria
         begin
            select P'(t) from P(t);
            biased crossover P'(t);    //  biased crossover rate used
            biased mutation P'(t);      // biased mutation rate used
            combine P'(t) and P(t) to form P(t+1);   // based on the survivorPercent rate
            evaluate P(t+1);
            t:=t+1;
         end
Loop end
```

Figure 2.13: Refined IGAs with separate evolution of new elements (ref. Fig. 2.5)

In the above experiments, it is assumed that new training patterns including all attributes come along with new attributes so that these patterns can be used to train the rule set with all attributes. This is likely in most realistic applications. For instance, a researcher may find a new symptom which is likely to contribute to the diagnosis of a certain disease. Then, new data associated with the old and new symptoms may be collected for further research. However, sometimes newly collected data may only contain information on the new attributes, without information on the old attributes. Therefore, when the old and new attributes are integrated, there is no on-hand training pattern to train the rule set for the whole set of attributes. As an alternative solution,

classifier agents may integrate the old and new training patterns together to form a new training pattern set according to the class categories. Then, IGAs can evolve a new rule set on the new training patterns.

2.7 Conclusion

In this chapter, incremental genetic algorithms (IGAs) have been proposed for incremental learning of classifier agents. IGAs eliminate the need to re-evolve the rule set from scratch in order to accommodate new incoming attributes. Using IGAs, a classifier agent can fully utilize current knowledge and quickly evolve a new solution.

The main features of IGAs, such as the incremental evolution of new attributes, formation of new population, integration of chromosomes, and biased rates for genetic operators, have been elaborated. Four real-world data sets were used to evaluate the performance of IGAs. The results show IGAs can be successfully used for incremental learning and speeds up the learning process as compared with normal GAs.

Four IGA approaches have been proposed in this chapter, and their performance has been compared. Classifier agents can choose one of them according to various conditions such as whether the group chromosomes from the old solution are preserved or the evolved rule sets from other agents are available. Experimental results show that utilizing the group chromosomes from the old solution and the evolved rule sets from other agents can achieve better performance. However, there are some tradeoffs for agents to determine which type of IGAs to use for incremental learning. For example, as IS2 and IS4 need to use the whole population from the last GA/IGA run instead of just one rule set, classifier agents need more memory for temporary storage, which will cost extra resource.

Chapter 3

Incremental Genetic Algorithms for New Class Acquisition

3.1 Introduction

We have considered the incremental learning from the perspective of accommodating new attributes in the previous chapter, and IGAs have been proposed with different initialization schemes to integrate the old chromosomes and new elements for new attributes. As discussed earlier in section 2.2, a classifier agent is actually exposed to the changing environment, and it needs to evolve its solution by adapting to various changes. There are three categories of changes for classification problems. This chapter continues to explore the utilization of IGAs from another perspective - incremental learning of new classes, and it is still considered in the context of multi-agent environment.

Agents are both self-interested and social. They have their own goals, but they also seek for collaboration. The interaction with other agents should in some way or another help each individual agent fulfill one or more of its goals. The motivation for collaboration can arise for the purpose of some temporary tasks, e.g. dealing with specific classification problem, or for long-term objectives, e.g. co-evolution. Classifier agents not only passively respond to the changing environment, but also actively interact, collaborate, and evolve autonomously. In this chapter, we still

simplify the multi-agent environment as a scenario of two agents, simple yet adequate to illustrate the collaboration among agents. Because this chapter is focused on designing incremental algorithms for acquiring and integrating new classes, the difference between the scenarios of multi-agent and two-agent has no effect on the algorithms involved.

IGA approaches are revised for acquiring new classes. Similar to the case of learning new attributes, four types of IGAs for new class acquisition are proposed and compared with retraining GAs in terms of performance. The experiment results on benchmark data sets and their analysis are reported to demonstrate the power of IGAs for new class acquisition.

```
Loop begin
if new classes are ready to be integrated
   t:=0;
   if group chromosomes in the old GA/IGA are available
        then select them as seeds;
             else select the chromosome with best fitness  in the old GA/IGA as seeds;
   if new chromosomes for new classes are available to be integrated
        then integrate them with the old chromosomes;
             else expand old chromosomes with some randomly created rules;
   evaluate P(t); // P(t) is the resulting population from the above steps
   while (not terminate-condition) do        //stopping criteria
          begin
             select P'(t) from P(t);
             crossover P'(t
             mutation P'(t);
             combine P'(t) and P(t) to form P(t+1); // based on the survivorsPercent rate
             evaluate P(t+1);
             t:=t+1;
          end
Loop end
```

Figure 3.1: Pseudocode of IGAs for new class acquisition

3.2 IGAs for New Class Acquisition

In order to be used for acquiring new classes, IGAs presented in the last chapter should be revised accordingly. In this section, only the different or revised components are presented, while the other common components share the presentation in Chapter 2. In addition, the biased rates for crossover and mutation will not be used in new class acquisition. Figure 3.1 shows the pseudocode of revised IGAs for new class acquisition.

There are two possible situations for new class acquisition, i.e. class expansion (CE) and rule integration (RI). In the first situation, an agent learns that some new classes are available for classification, and new data with the new classes are also provided for training. Then, the agent can expand the rule set solution autonomously. In the second situation, the agent acquires evolved rules about the new classes from another agent, besides the above-mentioned information. Then, it integrates new rules with its old rules.

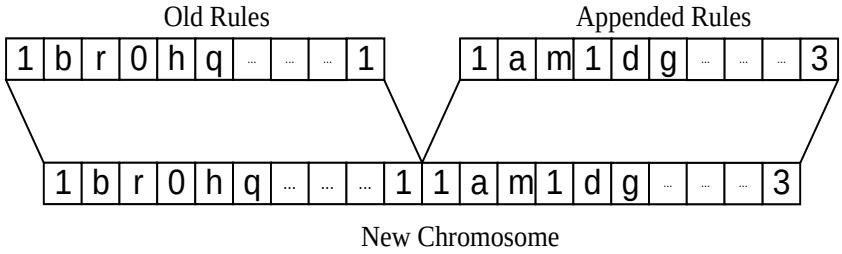


Figure 3.2: Formation of a new chromosome in IGAs with CE or RI

The main revision of IGAs lies in the formation of initial population and the integration of old and new elements. Figure 3.2 shows how the newly rules (chromosome) are concatenated with old rules (chromosome) to form a new chromosome in IGAs. Recalling the method used in Chapter 2 for integrating the new

elements of new attributes (cf. Figure 2.6), there are some difference between these two figures. First, Figure 2.6 only shows the formation of a new rule, which is only a segment of chromosome, while Figure 3.2 shows a complete chromosome. Second, in Figure 2.6, the new elements are selected with a matching mechanism and inserted into old rules so that the class category of the new rule is inherited from the old one and remains unchanged. While in Figure 3.2, the new rules are a piece of complete rule sets consisting of attribute and class elements. Therefore, it can be appended directly into the old chromosome without any matching mechanism.

There are many ways to construct new chromosomes in terms of the selection of old chromosome(s) and newly appended rules. We still follow similar options used in Chapter 2, and select four approaches as the research targets. For the choice of old chromosome(s), either the best rule set (chromosome) or the whole population in the last generation of old GA is used as seed(s) for all the initial population of IGA. For the new rules, two choices are selected, i.e., CE and RI. As discussed earlier, CE means the newly appended rules are randomly created, and RI means these rules are provided by the other agent instead. Table 3.1 lists these approaches, and assigns them with distinct names.

Table 3.1: IGAs alternatives on the formation of a new population for new class acquisition

New Rules / Old Rules	CE (Class Expansion)	RI (Rule Integration)
Best Chromosome	CE1	RI1
Group Chromosome	CE2	RI2

```
CE1:
for each newChrom[j]
    {  oldChrom:= the best chromosome from the last generation of GA/IGA;
       determine the total number of classes p for IGA;
       allocate the rule number for each class (r1 ... rp);
       parse oldChrom in terms of class category, and place rules into
            respective pools (1... p);
       determine the number of rules in each pool (n1…np);
       for each class k
          {  if (nk==0) then randomly create rk rules;
             else { if (rk<=nk) then randomly select rk rules from pool k;
                    else select all rules in pool k and randomly create rk-nk rules;
                  }
             append rk rules to bufferChrom;
          }
       newChrom[j]:=bufferChrom;
    }
```

```
RI1:
for each newChrom[j]
    {  oldChrom:= the best chromosome from the last generation of GA/IGA;
       incomingChrom:=the chromosome acquired from another agent;
       determine the total number of classes p for IGA;
       allocate the rule number for each class (r1 ... rp);
       parse oldChrom in terms of class category, and place rules into
            respective pools (1... p);
       parse incomingChrom, and place rules into respective pools (1.. p);
       determine the number of rules in each pool (n1…np);
       for each class k
          {  if (nk==0) then randomly create rk rules;
             else { if (rk<=nk) then randomly select rk rules from pool k;
                    else select all rules in pool k and randomly create rk-nk rules;
                  }
             append rk rules to bufferChrom;
          }
       newChrom[j]:=bufferChrom;
    }
```

Figure 3.3: Pseudocodes for the formation of initial population under CE1 and RI1

Figure 3.3 and 3.4 show the pseudocodes for forming a new initial population with the four methods. The percentages of new and old rules are not fixed when forming a new population. Instead, a random rule-selection scheme is applied. That is, the old and new rules are reshuffled to different rule pools in terms of their class categories, and they are selected randomly from respective pools to become a member of the new chromosome. Therefore, both old and new rules have the same probability to be

selected. If there are no rules available for certain classes as in the case of CE1 and CE2 and the corresponding pools are thus empty, the rules for new classes are created randomly.

```
CE2:
for each newChrom[j]
    {   oldChrom:= the chromosome[j] from the last generation of GA/IGA;
        determine the total number of  classes p for IGA;
        allocate the rule number for each class (r1 ... rp);
        parse oldChrom in terms of class category, and place rules into
              respective pools (1... p);
        determine the number of rules in each pool (n1…np);
        for each class k
           {   if (nk==0) then randomly create rk rules;
               else { if (rk<=nk) then randomly select rk rules from pool k;
                      else select all rules in pool k and randomly create rk-nk rules;
                    }
               append rk rules to bufferChrom;
           }
        newChrom[j]:=bufferChrom;
    }
```

```
RI2:
for each newChrom[j]
    {   oldChrom:= the chromosome[j] from the last generation of GA/IGA;
        incomingChrom:=the chromosome acquired from another agent;
        determine the total number of classes p for IGA;
        allocate  the rule number for each class (r1 ... rp);
        parse oldChrom in terms of class category, and place rules into
              respective pools (1... p);
        parse incomingChrom, and place rules into respective pools (1.. p);
        determine the number of rules in each pool (n1…np);
        for each class k
           {   if (nk==0) then randomly create rk rules;
               else { if (rk<=nk) then randomly select rk rules from pool k;
                      else select all rules in pool k and randomly create rk-nk rules;
                    }
               append rk rules to bufferChrom;
           }
        newChrom[j]:=bufferChrom;
    }
```

Figure 3.4: Pseudocodes for the formation of initial population under CE2 and RI2

It is possible that the newly acquired classes may overlap with the old ones. There are many options in tackling these situations. For example, classifier agents may

ignore the incoming rules for the existing classes, and only absorb those rules for the new classes. In our approaches, all incoming rules including those for the existing classes are used, as these rules have been evolved and can make contribution for the succeeding evolution. A new set of classes is determined first and rules are selected for these classes based on a random-selection scheme as mentioned above. Moreover, the allocated rule number for each class is identical when forming the new initial population, since it is unknown initially which class is harder to learn and requires more rules. It will not affect the final solution as the evolution process will adjust the distribution of rules to all classes with the use of genetic operators such as mutation and crossover. In other words, each class may end up in different number of rules which is more or less than the allocated rule number.

3.3 Experiment Results and Analysis

Three data sets are chosen for experiments in this chapter. They are wine data, iris data, and glass data, and only the training performance of classifier agents on these data sets is examined. In order to simulate the process of new class acquisition, these data sets are utilized in a specific way by partitioning them into two sets. The first set contains the data whose classes have already been known by the agent, and the other set contains the data whose classes are unknown initially. Then, the first set is used to train a rule set with GAs, and the other set is used to simulate incremental learning using IGAs. As the known classes are varied for different experimental settings and data sets, the exact partitioning of training data is thus varied.

The results reported are all averaged over ten independent runs. The parameters for GAs and IGAs, such as mutationRate, crossoverRate, generationLimits, etc, are given

under the results. The whole evolution process is recorded for GAs and IGAs, but we are only interested in some indicative results, which include initial classification rate (CR), number of generations, training time, and ending CR. (Their definitions can be referred to those tables in Chapter 2.)

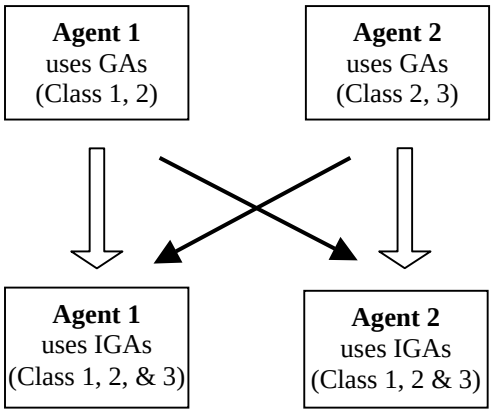


Figure 3.5: Illustration of experiments on new class acquisition

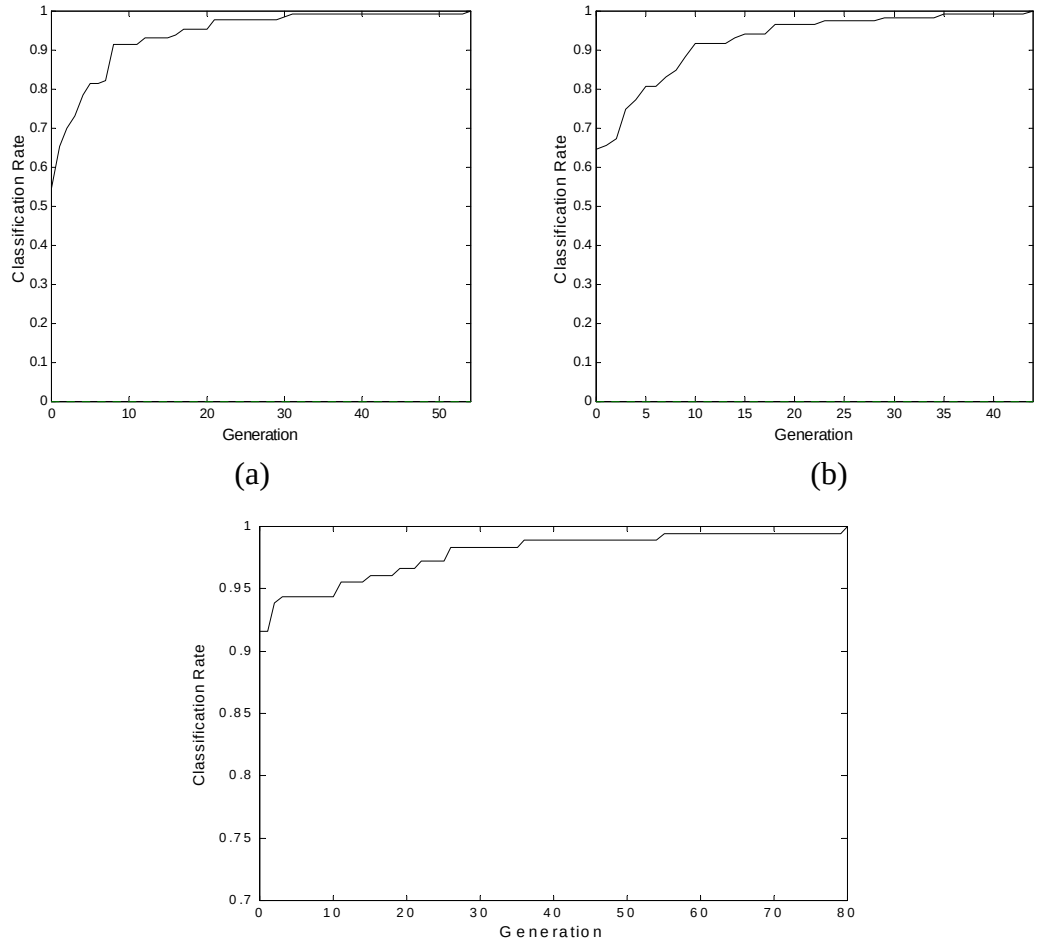
3.3.1 The Wine Data

Figure 3.5 illustrates experiments on class acquisition with detailed steps explained as follows. The wine data are used as an example to illustrate the design of experiments. It has 13 attributes and 3 classes. The experiments on the other data sets can be easily determined.

Step 1: Training data are prepared for two agents. It is assumed that both agents know 13 attributes at first, but agent 1 only knows class 1 and 2, and agent 2 only knows class 2 and 3. Therefore, different training data are provided to two agents.

Step 2: Both agents use GAs to evolve the rule sets on the currently known attributes and classes.

Step 3: Two agents exchange information, and then each knows the complete class set (class 1, 2 & 3). IGAs will be used to evolve the rule set on 3 classes. Note that there are four alternatives for IGAs (c.f., Table 3.1), and the whole training data will be provided for IGAs. For simplicity, we only focus on the IGAs in the first agent.



Note:
1. For GA and IGA, mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, generationLimit=150, stagnationLimit=30.

Figure 3.6: Simulation shows: (a) GA results in agent 1 with class 1 & 2; (b) GA results in agent 2 with class 2 & 3; (c) IGA (RI1) results in agent 1 with class 1, 2, & 3.

Table 3.2: Comparison of the performance of IGAs on the wine data with various class settings

		Agent 1 (Class=1& 2)			Agent 2 (Class=2 & 3)	
GA	Initial CR	0.6054			0.6353	
	Generations	48.8			38.2	
	T. time (s)	291.8			279.6	
	Ending CR	0.9992			1.0	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.4539	0.7421	0.9112	0.7618	0.9140
	Generations	135.9	138.0	82.7	101.6	66.2
	T. time (s)	1385.7	1520.6	969.2	1120.3	761.0
	Ending CR	0.9893	0.9955	0.9983	0.9955	0.9983
		Agent 1 (Class=1 & 3)			Agent 2 (Class=2 & 3)	
GA	Initial CR	0.7075			0.6059	
	Generations	12.2			44.8	
	T. time (s)	65.8			309.6	
	Ending CR	1.0			1.0	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.4539	0.5584	0.7916	0.5983	0.8152
	Generations	135.9	148.8	104.4	147.3	114.7
	T. time (s)	1385.7	1531.5	1181.6	1502.0	1269.1
	Ending CR	0.9893	0.9899	0.9983	0.9916	0.9933
		Agent 1 (Class=1 & 3)			Agent 2 (Class= 1 & 2)	
GA	Initial CR	0.6729			0.5906	
	Generations	13.3			60.3	
	T. time (s)	66.4			449.0	
	Ending CR	1.0			0.9983	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.4539	0.6554	0.8658	0.6860	0.8677
	Generations	135.9	145.6	117. 7	135.3	90.7
	T. time (s)	1385.7	1564.2	1353.5	1469.6	1016.6
	Ending CR	0.9893	0.9894	0.9944	0.9956	0.9988

Notes:

1. For GA and IGA, mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, generationLimit=150, stagnationLimit=30.

Figure 3.6 shows the results of a typical run of GA and IGA (RI1 is employed here). Figure 3.6(a) and 3.6(b) show that it takes 54 generations for GA in agent 1 to

evolve CR from 0.55 to 1.0, and 44 generations for GA in agent 2 to advance CR from 0.65 to 1.0. Figure 3.6(c) shows that CR decreases from 1.0 to 0.92 immediately after the formation of initial population. This can be explained by the fact that new classes are integrated into the new chromosomes, and the training data have incorporated the new class. Then, CR increases gradually, and reaches 1.0 again with 80 generations. It means that IGA has successfully integrated the new classes and evolved a new rule set.

More experiments with different settings of classes are conducted, and the performance of different IGA approaches is compared. Table 3.2 summarizes the results of three experiments with different class combinations. For each experiment, GA runs in parallel for agent 1 and 2. Upon getting the results from two GAs, IGAs will begin to work with different approaches. Therefore, these approaches have the same starting point that is fair for comparison.

As mentioned earlier, four IGA approaches are available on forming the new population, in terms of the way on choosing the old rule sets (chromosomes) and the way on combining them with new classes. They are noted as CE1, CE2, RI1, RI2, as shown in Table 3.1. In order to compare them to the conventional approach, a retraining GA experiment is also conducted, which abandons the current solution and starts the evolution process from scratch.

It is noted that all approaches can successfully integrate the new classes learned and evolve a new rule set. When observing the initial CR in the IGA part of all the three experiment results, it is found that the initial CRs of RI1 and RI2 are apparently higher than those of CE1 and CE2, and the retraining GA has the lowest initial CR. This can be explained by recalling the method that each approach uses to form the initial population. As RI1 and RI2 acquire the evolved rule set from the other agent and use it

to form a new population, they can obtain more useful information from the rule set than CE1 and CE2 using the randomly created rules. The retraining GA is in the most disadvantaged position, as it works out the new population with a totally random method.

When comparing the performance of GA/IGA, the ending CR is used as the main benchmark, while the number of generations/training time used is also considered. It is found that the performance of the four IGA approaches are generally better than the retraining GA in terms of ending CR and training time. Among the four IGA approaches, RI1 and RI2 have better performance than CE1 and CE2, which testifies that agents can benefit from the evolved rule sets acquired from other agents. Furthermore, RI2 achieves the best performance in the third experiments, RI1 wins in the second experiment, and they tie in the first experiment. This hints that group chromosomes used in RI2 may help achieve better performance.

3.3.2 The Iris Data

Table 3.3 shows the results of three experiments on the iris data, and the performance of retraining GA and four IGA approaches is compared.

It has been reported for the iris data that the first class is linearly separable from the other two, but the other two are not linearly separable from each other (Blake and Merz, 1998). This assertion has been testified in our experiment results, as shown in the GA part of Table 3.3. For example, in the first experiments, agent 1 with class 1 and 2 can achieve an ending CR of 1.0, while agent 2 with class 2 and 3 can only achieve an ending CR of 0.953. The results from other two experiments also support

the fact that class 2 and 3 cannot be fully classified and class 1 can be fully separated from the other two classes.

Table 3.3: Comparison of the performance of IGAs on the iris data with various class settings

		Agent 1 (Class=1& 2)			Agent 2 (Class=2 & 3)	
GA	Initial CR	0.8110			0.6515	
	Generations	6.6			66.3	
	T. time (s)	2.3			23.2	
	Ending CR	1.0			0.9530	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.5222	0.7487	0.8987	0.6953	0.9137
	Generations	81.8	72.6	43.7	74.9	44.9
	T. time (s)	38.0	35.9	21.4	36.7	22.3
	Ending CR	0.9612	0.9610	0.9670	0.9627	0.9653
		Agent 1 (Class=1 & 3)			Agent 2 (Class=2 & 3)	
GA	Initial CR	0.8440			0.6560	
	Generations	6.8			60.8	
	T. time (s)	2.1			19.0	
	Ending CR	1.0			0.9520	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.5222	0.7073	0.8707	0.6933	0.8800
	Generations	81.8	78.8	53.4	76.5	44.6
	T. time (s)	38.0	37.2	26.0	38.0	22.9
	Ending CR	0.9612	0.9660	0.9687	0.9600	0.9673
		Agent 1 (Class=1 & 3)			Agent 2 (Class= 1 & 2)	
GA	Initial CR	0.8490			0.7960	
	Generations	7.1			8.1	
	T. time (s)	2.2			2.6	
	Ending CR	1.0			1.0	
		Agent 1 (Class=1, 2 & 3)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.5222	0.7260	0.8020	0.6967	0.834
	Generations	81.8	70.7	67.2	75.6	60.7
	T. time (s)	38.0	31.7	31.1	36.1	29.3
	Ending CR	0.9612	0.9600	0.9627	0.9593	0.9627

Notes:

1. For GA and IGA, mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=6, popSize=100, generationLimit=100, stagnationLimit=30.

The comparison among IGA approaches and retraining GA leads to similar findings as those drawn from the wine data. RI1 or RI2 is again the best approach among IGAs in terms of initial CR, training time, and ending CR. The only exception is that the ending CR of retraining GA is sometimes slightly better than some of the IGA approaches, but the training time for retraining GA is still the longest. This may be because the iris data is easier to be classified, thus some IGA approaches may not show much advantage over retraining GA in terms of CR. But they still show advantage in terms of training time.

3.3.3 The Glass Data

Table 3.4 shows the results of three experiments on the glass data. As the glass data has more classes and instances, it is noted that the CRs achieved are apparently lower than the other two data sets. This means the glass data is harder to be classified.

The results presented in the GA part also give us some information regarding the internal interference of data. For instance, when GA is used to classify the data with class 3, 4, 5, and 6, it can achieve an ending CR of about 0.95. In contrast, the ending CR of GA with class 1, 2, 3, and 4 can only achieve a much lower value of 0.64. This means it is more difficult to classify the latter group of classes than the former group, which also indicates there is more interference among the data belonging to the latter group of classes.

Through the comparison between the four IGA approaches and the retraining GA, similar findings can be obtained. All the four IGA approaches achieve higher ending CR than the retraining GA with much greater improvement. RI1 or RI2 is still the best approach in terms of the ending CR.

Table 3.4: Comparison of the performance of IGAs on the glass data with various class settings

		Agent 1 (Class=1, 2 & 3)		Agent 2 (Class=3, 4, 5 & 6)		
GA	Initial CR	0.4871		0.4368		
	Generations	261.1		233.4		
	T. time (s)	2125.6		1016.4		
	Ending CR	0.7577		0.9515		
		Agent 1 (Class=1, 2, 3, 4, 5, & 6)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.3424	0.5318	0.5869	0.5332	0.5874
	Generations	245.4	291.3	230.9	274.1	241.9
	T. time (s)	2492.2	3186.0	2566.0	3085.3	2737.8
	Ending CR	0.6278	0.7341	0.7360	0.7276	0.7383
		Agent 1 (Class=1, 2, 3 & 4)		Agent 2 (Class=3, 4, 5 & 6)		
GA	Initial CR	0.4256		0.4265		
	Generations	206.6		168.2		
	T. time (s)	1849.7		691.1		
	Ending CR	0.6358		0.9368		
		Agent 1 (Class=1, 2, 3, 4, 5, & 6)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.3424	0.5164	0.5799	0.5117	0.5757
	Generations	245.4	214.4	208.1	182.1	193.5
	T. time (s)	2492.2	2375.0	2321.1	2092.2	2202.1
	Ending CR	0.6278	0.6930	0.7220	0.6715	0.7070
		Agent 1 (Class=1, 2, 3, 4 & 5)		Agent 2 (Class=3, 4, 5 & 6)		
GA	Initial CR	0.4024		0.4363		
	Generations	217.2		150.0		
	T. time (s)	1905.9		607.1		
	Ending CR	0.6547		0.9314		
		Agent 1 (Class=1, 2, 3, 4, 5, & 6)				
		Retraining GA	CE1	RI1	CE2	RI2
IGAs	Initial CR	0.3424	0.5535	0.6111	0.5571	0.6059
	Generations	245.4	170.8	192.2	192.6	199.3
	T. time (s)	2492.2	1929.6	2096.0	2157.6	2249.5
	Ending CR	0.6278	0.6926	0.7144	0.7056	0.7269

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=500, generationLimit=300, stagnationLimit=100.

3.4 Conclusion

This chapter has explored the incremental learning of classifier agents through new class acquisition. Following the basic design of GAs and IGAs presented in the last chapter, IGAs have been further adapted to tackle the tasks of acquiring new class. Based on the different situations of class expansion and rule integration, four approaches have been derived and examined. Classifier agents can choose one of them according to various conditions such as whether group chromosomes from the old solution are preserved or the evolved rule sets from other agents are available. Three benchmark data sets are used to evaluate these approaches, and their performance is compared. The experimental results show that GAs and IGAs can be successfully used for new class acquisition, and classifier agents can benefit from collaborative learning with other agents.

Chapter 4

Continuous Incremental Genetic Algorithms

4.1 Introduction

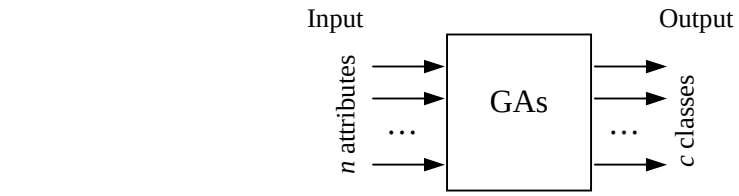
In previous chapters, IGAs are proposed for incremental learning of new attributes and classes. When leaning new attributes, it is assumed that only one set of new attributes is introduced, and they are treated in a batch mode. Actually IGAs can also be extended to run in continuous mode, which means new attributes can be introduced one by another.

This chapter proposes continuous incremental genetic algorithms (CIGAs) to address continuous incremental learning and training of input attributes for classifier agents. Rather than using input attributes in a batch as with normal GAs, CIGAs learn attributes one after another. The resulting classification rule sets are also evolved incrementally to accommodate the new attributes. IGAs are still used as the basic algorithm to accommodate one or more new attributes. Based on IGAs, CIGAs are designed as iterative algorithms for continuous incremental learning and training.

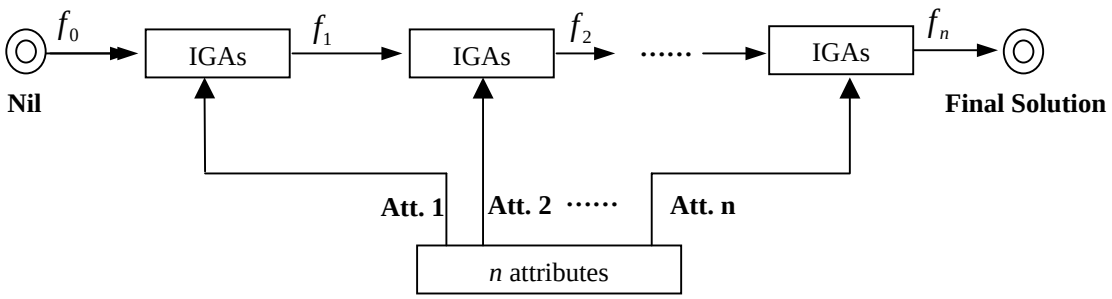
With the use of four different schemes in IGAs, four approaches of CIGAs are explored. CIGAs are evaluated with four benchmark classification data sets, and their performance is compared with normal GAs. The simulation results show that CIGAs can be used successfully for continuous incremental training of classifier agents and can achieve better performance than normal GAs using batch-mode training.

4.2 Continuous Incremental Genetic Algorithms (CIGAs)

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



(a) Normal GA-based solution



(b) Illustration of CIGAs

Figure 4.1: Illustrations of normal GAs and CIGAs

Figure 4.1 illustrates the concepts of normal GAs and CIGAs. As shown in Figure 4.1(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. CIGAs are significantly different. As shown in Figure 4.1(b), they consist of several stages, each dealing with one new attribute by employing IGAs to

accommodate the new attribute. They may start from nil knowledge. With the continuous introduction of new attributes, the rule set can be incrementally evolved step by step and achieve the final solution. CIGAs can also stop at any middle stage, and the solution at that time can be used to solve the problem with a subset of attributes.

Step 1: Initialization. Set *i*=1. Initialize parameters for CIGAs.

Step 2: *i*-th iteration. Introduce the *i*-th attribute.

Step 3: IGAs use IS1 or IS3 to integrate the new elements for the *i*-th attribute, forming the new initial population. (*i*=1 means the introduction of the first attribute. As there is no old solution, the new elements for the first attribute are simply used as the initial population.)

Step 4: Evolve the chromosomes with IGAs until the stopping criteria are met, using the training patterns including attributes from the 1st to the *i*-th.

Step 5: If *i* is equal to *n* (which is the total number of attributes), the training process will be stopped. Otherwise, increase *i* by 1. Repeat the procedure from step 2 to step 5.

Figure 4.2: Algorithms for CIGA1 and CIGA3

Following the notations presented above, we denote each stage of CIGAs as:

$$f_i : \quad (f_{i-1}, X_i) \to C \qquad\qquad i = 1,2,...,n \qquad\qquad (4.1)$$

where, f_i is a solution for the problem with i attributes, and f_0 is a special case - nil solution. X_i is the vector of training patterns with i attributes, and C is the set of output classes. In each stage i, X_{i-1} is expanded into X_i by introducing the new i-th

attribute, and the old solution f_{i-1} is also used to help evolve a new solution f_i. Therefore, $(f_0, f_1, ..., f_n)$ provides the corresponding solutions with the incremental introduction of new attributes, and f_n is the final solution accommodating the whole set of attributes.

Figure 4.2 shows the algorithms for CIGA1 and CIGA3, and their names are derived from their usage of IS1 and IS3 in step 3. In step 3, classifier agents have known the new attribute and old solution. By employing IS1 or IS3, new elements will be created randomly for this new attribute and integrated with old chromosomes, as presented in Section 2.4. The choice of best chromosome or group chromosomes differentiates between CIGA1 and CIGA3.

4.3 Experiments with CIGA1 and CIGA3

Four benchmark data sets, wine data, glass data, cancer data, and yeast data, are used to evaluate the performance of CIGAs in this chapter. Each data set is equally partitioned into two parts. One half is for training, and the other half is for testing. In addition, the biased rates for crossover and mutation will not be used in experiments.

Only one attribute is introduced during each iteration in CIGAs, and the sequence of introducing attributes is the same as in the original benchmark data set. As the first step, CIGA1 and CIGA3 are evaluated on the glass and yeast data sets.

Table 4.1 shows the performance of classifier on the glass data, using CIGA1, CIGA3, and normal GA. The whole training process is recorded for CIGA1 and CIGA3. Each column records the initial CR, which is the classification rate achieved on the training data after initially integrating the new attribute. After some generations

(training time), the classifier achieves an ending CR on the training data, and a test CR on the test data.

Table 4.1: Performance comparison on the glass data - CIGA1, CIGA3, and normal GA

CIGA1	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.3505	0.4879	0.5318	0.6131	0.6383	0.6467	0.6664	0.672	0.6907
Generations	54.1	50.5	57	48.3	38	45.4	40	41.4	36.7
T. time (s)	41.5	52.1	71.2	69	60.8	76.9	72.8	78.9	73.5
Ending CR	0.4869	0.5299	0.6131	0.6383	0.6458	0.6664	0.672	0.6907	0.6963
Test CR	0.3542	0.3654	0.3523	0.3645	0.3692	0.4009	0.3963	0.4215	0.4178
CIGA3	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.3673	0.5393	0.5757	0.6533	0.6776	0.6981	0.7131	0.7234	0.7383
Generations	58.4	53.9	57	47.2	49.8	43.1	44	38.7	32.6
T. time (s)	47.6	58.1	74.8	69.4	79.7	73.2	79.9	74.7	64.1
Ending CR	0.5383	0.5729	0.6533	0.6776	0.6981	0.7131	0.7234	0.7383	0.7402
Test CR	0.3636	0.3729	0.3813	0.3897	0.3841	0.4028	0.415	0.4336	0.429

Summary	**Normal GA**	**CIGA1**	**CIGA3**
Initial CR	0.3308	0.3505	0.3673
Generations	167.3	411.4	424.7
T. time (s)	354.6	596.7	621.5
Ending CR	0.5897	0.6963 (18.1%)	0.7402 (25.5%)
Test CR	0.3953	0.4178 (5.7%)	0.429 (8.5%)

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30;
2. For each stage of CIGAs, generationLimit=60; For Normal GA, generationLimit=540.
   (In order to be fair, the generationLimit for normal GA is equal to the sum of those for each stage, other experiments in this chapter follow the same mechanism.)

The bottom part of Table 4.1 provides a summary of CIGA1 and CIGA3. The generations and training time consumed in all stages are summed up. The initial CR for CIGA1 and CIGA3 in the summary table is actually the initial CR with attribute 1, which is different from the initial CR for the normal GA (with the whole set of attributes). The ending CR and test CR are the final training and test CRs with the whole set of attributes introduced. They are also compared with the normal GA with the improvement percentage computed.

Figure 4.3 shows the comparison of performance among CIGA1, CIGA3, and normal GA in terms of training and test CRs. As normal GA uses the whole set of attributes in batch, there are only two points for it in the figure. On the contrary, CIGA1 and CIGA3 each has a curve to show their continuous, incremental training process. From Table 4.1 and Figure 4.3, it is found that the training CR rises steadily with the introduction of each new attribute, while the change of test CR is relatively small. It is also shown that CIGA3 outperforms CIGA1 in terms of both training and test CRs. Recalling the difference between CIGA1 and CIGA3, it can be confirmed that the group chromosomes used by IS3 in CIGA3 may contain more useful information than the best chromosome used by IS1 in CIGA1. For the comparison between CIGAs and the normal GA, it is noted that CIGAs outperform normal GA in terms of training CR and test CR, with a significant improvement around 6% - 25%.

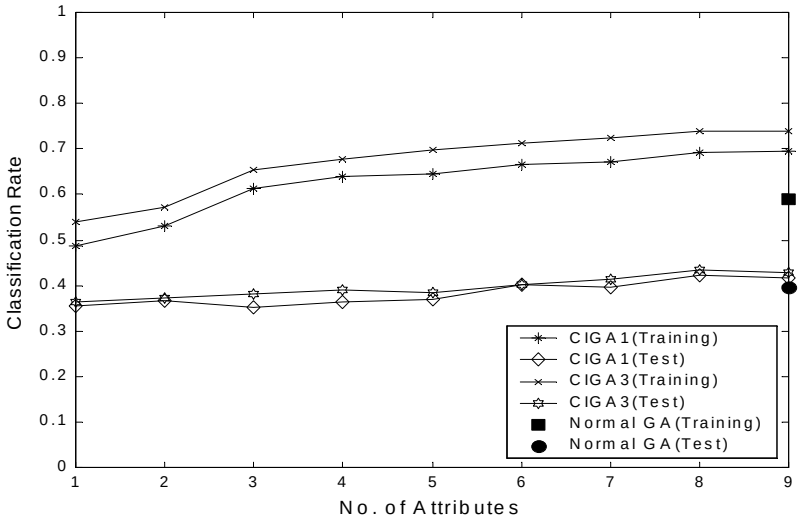


Figure 4.3: Comparison of CIGA1, CIGA3, and normal GA on the glass data

The classifier performance on the yeast data is recorded in Table 4.2, and also shown in Figure 4.4. The performance comparison illustrates that CIGA1 and CIGA3

outperform normal GA in terms of training and test CRs. The improvement is also significant, with an increase of 8%-11%. Furthermore, CIGA3 still outperforms CIGA1.

Table 4.2: Performance comparison on the yeast data – CIGA1, CIGA3, and normal GA

CIGA1	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8
Initial CR	0.2725	0.3084	0.3186	0.3499	0.3588	0.3664	0.3714	0.3752
Generations	55.8	49.1	52.6	48.8	47.6	40.5	40.7	43.7
T. time (s)	156.6	186	242	257.7	269.2	228.5	237.8	262.6
Ending CR	0.3082	0.3164	0.3491	0.3588	0.3664	0.3714	0.3752	0.3803
Test CR	0.3081	0.314	0.3372	0.3419	0.346	0.3505	0.3528	0.3561
CIGA3	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8
Initial CR	0.2662	0.3093	0.3155	0.3635	0.3751	0.3778	0.3801	0.3809
Generations	52	46.7	51.7	56	46.6	45.3	37	36.3
T. time (s)	158.3	179.9	242.3	291.5	268.1	267.4	230	231.6
Ending CR	0.3092	0.3155	0.3635	0.3751	0.3778	0.3801	0.3809	0.3817
Test CR	0.307	0.3074	0.3488	0.3549	0.3562	0.3559	0.3553	0.3559

Summary	**Normal GA**	**CIGA1**	**CIGA3**
Initial CR	0.2127	0.2725	0.2662
Generations	111.7	378.8	371.6
T. time (s)	628.2	1840.4	1869.1
Ending CR	0.3414	0.3803 (11.4%)	0.3817 (11.8%)
Test CR	0.3284	0.3561 (8.4%)	0.3559 (8.4%)

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30;
2. For each stage of CIGAs, generationLimit=60; For Normal GA, generationLimit=480.

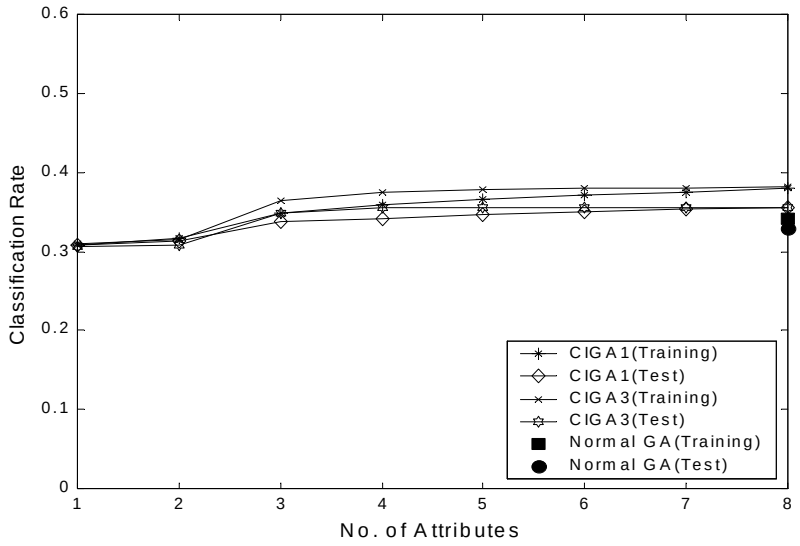


Figure 4.4: Comparison of CIGA1, CIGA3, and normal GA on the yeast data

4.4 Experiments with CIGA2 and CIGA4

Experiments with CIGA1 and CIGA3 have shown that they outperform normal GA. In this section, further refinement for CIGAs is explored. Recalling that IGAs have four initialization schemes, and IS1 and IS3 have been used to integrate the randomly created new elements. This motivates us to design new algorithms of CIGAs by utilizing IS2 and IS4, and thus their names are assigned as CIGA2 and CIGA4 respectively. In order to employ IS2 or IS4, a new step is needed to evolve separately new elements for a new attribute. Figure 4.5 shows the details of CIGA2 and CIGA4 algorithms. Note that the new step 3 is inserted into the algorithms as a separate evolution process, and with this step, a group of chromosomes are evolved and thus available for the integration process in IS2 or IS4.

Step 1: Initialization. Set i = 1. Initialize parameters for CIGAs.

Step 2: i-th iteration. Introduce the i-th attribute.

Step 3: Evolve the i-th attribute alone, using the training patterns for the i-th attribute.

Step 4: IGAs use IS2 or IS4 to integrate the new elements for the i-th attribute, forming the new initial population. (i=1 means the introduction of the first attribute. As there is no old solution, the new elements for the first attribute are simply used as the initial population.)

Step 5: Evolve the chromosomes with IGAs until the stopping criteria are met, using the training patterns including attributes from the 1st to the i-th.

Step 6: If i is equal to n (which is the total number of attributes), the training process will be stopped. Otherwise, increase i by 1. Repeat the procedure from step 2 to step 6.

Figure 4.5: Algorithms for CIGA2 and CIGA4

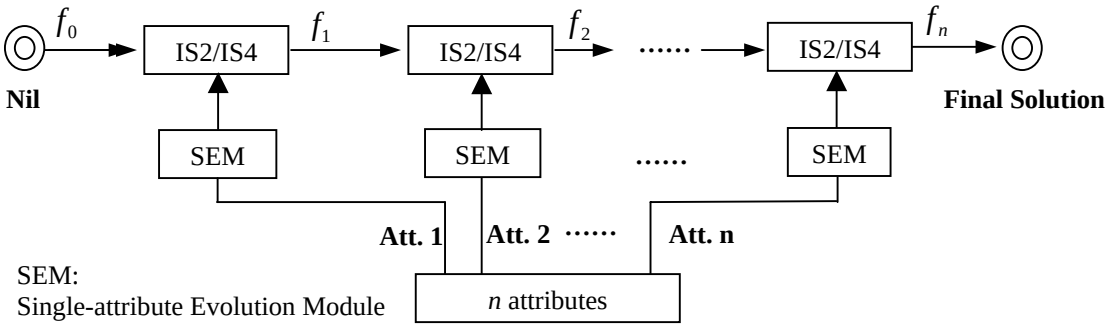


Figure 4.6: Illustration of CIGA2 and CIGA4

Figure 4.6 shows the illustration for CIGA2 and CIGA4, which is refined from Figure 4.1(b). Note that SEMs (Single-attribute Evolution Modules) are inserted to evolve separately a single attribute, and the function of these modules corresponds to step 3 in Figure 4.5.

Most experiment settings of CIGA2 and CIGA4 are the same as those for CIGA1 and CIGA3, except the new settings for SEMs. Considering that it is relatively easy to evolve a single attribute, the parameters for SEMs are chosen as half of the values used in the normal iteration to reduce the time cost. This means: generationLimit, ruleNumber, and popSize are set as half of the normal iteration values.

First, experiments with CIGA2 and CIGA4 are conducted on the wine and cancer data. Table 4.3 and Figure 4.7 show the performance comparison on the wine data, using CIGA2, CIGA4, and normal GA. It is found that CIGA2 and CIGA4 outperform normal GA in terms of training CR and test CR, with an improvement around 1% - 4%. Furthermore, CIGA4 outperforms CIGA2 in terms of both training and test CRs. Recalling the difference between CIGA2 and CIGA4, it can be confirmed again that the group chromosomes used by IS4 in CIGA4 may contain more useful information than

the best chromosome used by IS2 in CIGA2. This result conforms to those obtained from the experiments of CIGA1 and CIGA3.

Table 4.3: Performance comparison on the wine data - CIGA2, CIGA4, and normal GA

CIGA2	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att.7
Initial CR	0.5809	0.7112	0.8112	0.836	0.8596	0.8888	0.9191
Generations	39.4	40	39.7	40	40	38.3	38.9
T. time (s)	26.5	43.6	52.3	60.7	66.4	68.8	74.6
Ending CR	0.664	0.8404	0.8618	0.8854	0.9022	0.9213	0.9461
Test CR	0.6596	0.6787	0.6528	0.6584	0.6528	0.7079	0.7742

CIGA2 (contd.)	Add Att.8	Add Att. 9	Add Att. 10	Add Att. 11	Add Att. 12	Add Att. 13
Initial CR	0.9236	0.9404	0.9517	0.9708	0.9742	0.9787
Generations	37	36.4	34.6	28.3	25.7	15.4
T. time (s)	76	79.7	79.6	68.9	66.6	43.6
Ending CR	0.9562	0.964	0.9708	0.9831	0.9899	0.9955
Test CR	0.809	0.8258	0.8517	0.8472	0.8517	0.8573

CIGA4	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att.7
Initial CR	0.5843	0.7798	0.8371	0.8528	0.8787	0.9067	0.9315
Generations	38.5	40	38.4	40	40	39.9	39
T. time (s)	26.7	44.4	50.2	58.8	64.8	70.7	74.1
Ending CR	0.6764	0.8528	0.8663	0.8978	0.9146	0.9393	0.964
Test CR	0.682	0.7112	0.691	0.6978	0.7225	0.7787	0.8461

CIGA4 (contd.)	Add Att.8	Add Att. 9	Add Att. 10	Add Att. 11	Add Att. 12	Add Att. 13
Initial CR	0.9461	0.9584	0.9562	0.9798	0.9787	0.9798
Generations	38.8	34.8	35.3	20.7	23.2	18.1
T. time (s)	79.9	75.3	80.5	52.1	60.2	49.9
Ending CR	0.9719	0.9787	0.9831	0.991	0.9921	0.9955
Test CR	0.8483	0.8292	0.8393	0.8506	0.8607	0.8607

Summary	**Normal GA**	**CIGA2**	**CIGA4**
Initial CR	0.4034	0.5809	0.5843
Generations	144.7	453.7	446.7
T. time (s)	356.7	807.3	787.6
Ending CR	0.9876	0.9955 (0.8%)	0.9955 (0.8%)
Test CR	0.8303	0.8573 (3.3%)	0.8607 (3.7%)

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=200, stagnationLimit=30;
2. For each stage of CIGAs, generationLimit=40; For Normal GA, generationLimit=520.

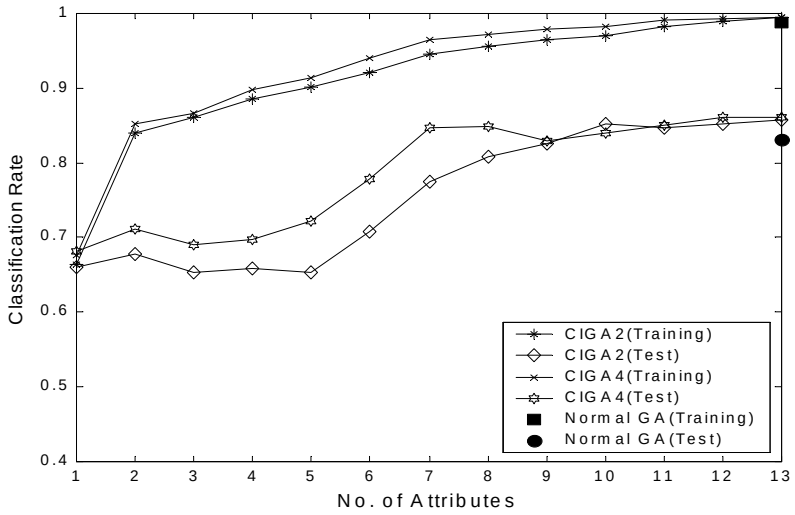


Figure 4.7: Comparison of CIGA2, CIGA4, and normal GA on the wine data

Table 4.4: Performance comparison on the cancer data - CIGA2, CIGA4, and normal GA

CIGA2	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.8138	0.9203	0.9355	0.9381	0.9499	0.9562	0.9582	0.9619	0.9582
Generations	31.8	52.2	50.9	50	54.5	48.9	45.2	42.3	40.7
T. time (s)	37	94.8	110.6	121.9	141.8	132.8	127.9	126.7	125.3
Ending CR	0.8711	0.9358	0.9513	0.957	0.9642	0.9668	0.9682	0.9691	0.9693
Test CR	0.8539	0.9527	0.9602	0.9544	0.9527	0.9582	0.9573	0.9582	0.9596
CIGA4	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.8324	0.9272	0.941	0.9438	0.951	0.9573	0.9633	0.9599	0.9633
Generations	32.1	46.3	45.9	47	56.1	51.1	43.3	42.1	39.8
T. time (s)	39.9	86.6	101.4	113.9	143.5	138	127.9	131	125.6
Ending CR	0.8711	0.9372	0.9513	0.9564	0.9639	0.967	0.9685	0.9685	0.9696
Test CR	0.8539	0.9479	0.9553	0.9521	0.9487	0.9593	0.9587	0.9639	0.9699

Summary	**Normal GA**	**CIGA2**	**CIGA4**
Initial CR	0.7324	0.8138	0.8324
Generations	136.6	416.5	403.7
T. time (s)	381.2	1018.8	1007.8
Ending CR	0.967	0.9693 (0.2%)	0.9696 (0.3%)
Test CR	0.9619	0.9596 (-0.2%)	0.9699 (0.8%)

Notes:

1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%, ruleNumber=30, popSize=100, stagnationLimit=30;
2. For each stage of CIGAs, generationLimit=60; For Normal GA, generationLimit=540.

Table 4.4 shows the classifier performance on the cancer data. The nine attributes of the cancer data are introduced sequentially. The whole incremental training process is recorded for CIGA2 and CIGA4. With a comparison to normal GA, it is found that the training and test CRs for these three approaches are similar, with not much improvement or deterioration. Figure 4.8 also shows visually that their performance is very close. The curves in the figure also show that test CR may exceed training CR in some stages during the training process.

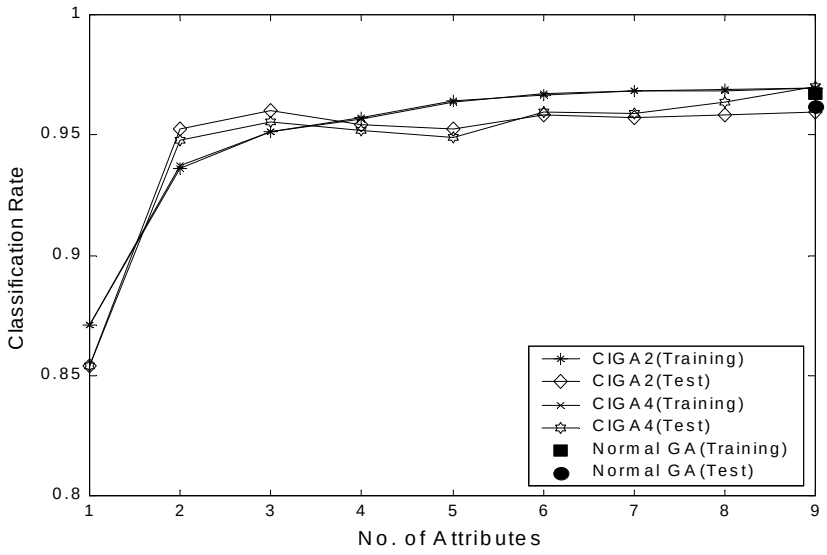


Figure 4.8: Comparison of CIGA2, CIGA4, and normal GA on the cancer data

In order to compare the performance of the four types of CIGAs, experiments with CIGA2 and CIGA4 on the glass and yeast data are conducted, and their results are compared with those of CIGA1 and CIGA3. Table 4.5 and Table 4.6 only list the respective summary of the evolution process. The detailed information of each stage of CIGA2 and CIGA4 can be found in Appendix B, while those of CIGA1 and CIGA3 can be found in Table 4.1 and Table 4.2 respectively. Table 4.5 and Figure 4.9 compare the performance of four types of CIGAs on the glass data in two aspects, training CR

and test CR respectively. It is shown that all types of CIGAs outperform normal GA in terms of both CRs. Among the four types, CIGA4 is the best approach, and CIGA1 and CIGA3 are inferior to CIGA2 and CIGA4 in both CRs. This tells us that the separate evolution of each new attribute does contribute to better performance.

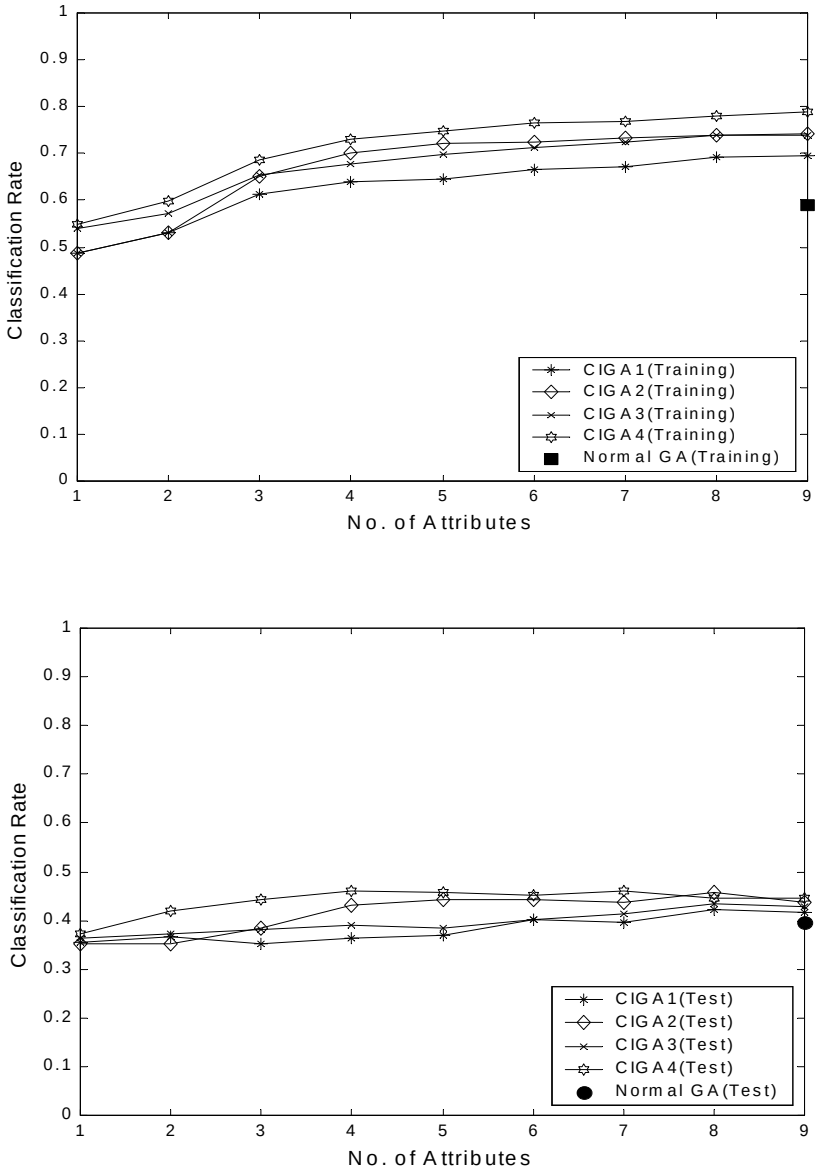


Figure 4.9: Performance comparison of CIGAs on the glass data

Table 4.5: Performance comparison of CIGAs on the glass data

Summary	Normal GA	CIGA1	CIGA2	CIGA3	CIGA4
Initial CR	0.3308	0.3505	0.3598	0.3673	0.3467
Generations	167.3	411.4	454	424.7	496.3
T. time (s)	354.6	596.7	779.2	621.5	874.6
Ending CR	0.5897	0.6963 (18.1%)	0.7421 (25.8%)	0.7402 (25.5%)	0.7879 (33.6%)
Test CR	0.3953	0.4178 (5.7%)	0.4374 (10.7%)	0.429 (8.5%)	0.4458 (12.8%)

Notes: The experiment setting is the same as that for Table 4.1.

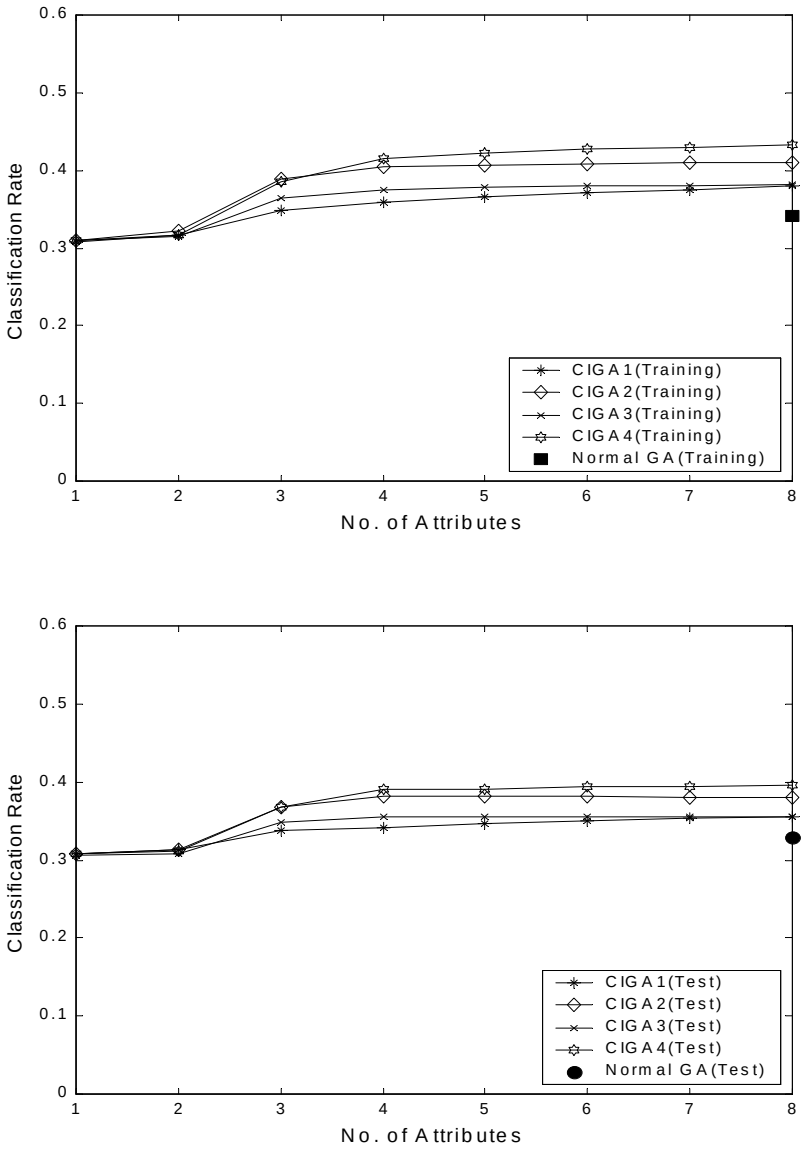


Figure 4.10: Performance comparison of CIGAs on the yeast data

Table 4.6 and Figure 4.10 show the comparison of performance of CIGAs on the yeast data. Similar results have been obtained regarding their training and test performance. CIGA4 is again the best approach among the four choices, and CIGA2 and CIGA4 outperform CIGA1 and CIGA3 in both CRs.

Table 4.6: Performance comparison of CIGAs on the yeast data

Summary	Normal GA	CIGA1	CIGA2	CIGA3	CIGA4
Initial CR	0.2127	0.2725	0.2667	0.2662	0.2612
Generations	111.7	378.8	409.3	371.6	429.5
T. time (s)	628.2	1840.4	2475.3	1869.1	2655.5
Ending CR	0.3414	0.3803 (11.4%)	0.4097 (20%)	0.3817 (11.8%)	0.4326 (26.7%)
Test CR	0.3284	0.3561 (8.4%)	0.3803 (15.8%)	0.3559 (8.4%)	0.396 (20.6%)

Notes: The experiment setting is the same as that for Table 4.2.

4.5 Comparison to other methods

In the above experiments, our CIGA approaches are compared with the normal GA using the benchmark data sets. It is found that the use of CIGAs (especially CIGA4) can improve the performance of classifiers in terms of both training and test CRs. In this section, we compare further the results of CIGA4 with those reported in the literature.

For the wine data, Corcoran and Sen (1994) used a real-coded GA-based machine learning approach, and achieved an average training rate of 99.5%. Ishibuchi et al. (1999) 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 (2000) used GA-fuzzy classifiers which achieve a classification rate of 98.3%. Comparing to these results in the literature, CIGA4 achieves the highest rate as 99.55% (c.f. Table 4.3).

Regarding the cancer data, Wolberg and Mangasarian (1990) reported a test rate of 95.9% with the use of hyperplanes, and Zhang (1992) reported 93.7% with the use of 1-nearest neighbor. CIGA4 achieves 96.99% (c.f. Table 4.4), which also outperforms these two. For the glass data, Falco et al. (2002) used genetic programming frameworks to discover classification rules, and reported a training rate of 64.61%, while CIGA4 achieves a better rate of 78.79% (c.f. Table 4.5).

4.6 Discussions

From the experiments on the four benchmark data sets, we have an overall picture on CIGAs. First, with CIGAs, classifier agents are equipped with a capability to integrate new attribute(s). Therefore, classifier agents can evolve their solutions incrementally in any situation where the introduction of new attributes may be frequent. Furthermore, four types of CIGAs are available to accommodate new attributes, so that the classifier agents can choose one type to cope with the dynamic environment.

With the evaluation of classifier performance on the benchmark data sets and comparison to normal GAs, it is found that the use of CIGAs (especially CIGA4) can improve the performance of classifiers in terms of both training and test CRs. For some problems such as glass and yeast, the improvement versus normal GA is significant. For problems such as wine and cancer, the improvement is relatively smaller. This result may be explained by the fact that the final accuracy for these data achieved with GA is already high. This means these data are easier to be classified, and therefore CIGAs may not have much room to demonstrate their advantages against normal GA on these data.

Based on the experiment results, CIGA4 generally outperforms CIGA2 in both training CR and test CR. This tells us that group chromosomes generally contain more information than the best chromosome alone. Especially, when these chromosomes are used as seeds to create the initial population for the next round of evolution, group chromosomes allow more degree of heterogeneity. Also, separate evolution of each new attribute in advance contributes to the improvement on performance, as demonstrated by the results that CIGA2 and CIGA4 outperform CIGA1 and CIGA3.

However, there are some tradeoffs here. First, the utilization of group chromosomes from the old solution depends on its availability. For example, a training process is interrupted in the middle, and the group chromosomes in the final solution are not preserved. Therefore, group chromosomes are not available when the training process is resumed later. This suggests that, in order to cope with some unexpected circumstances, it may be necessary to use a combination of different types of CIGAs. Second, the training time for CIGA2 and CIGA4 will be generally longer as they need an extra evolution process for each new attribute. For example, it is found from Table 4.5 that the training time of CIGA4 on the glass data increased by 40.7% as compared to CIGA3 (increased from 621.5s to 874.6s), and the training time of CIGA2 used for the yeast data increased by 34.5% as compared to CIGA1 (c.f. Table 4.6, increased from 1840.4s to 2475.3s).

4.7 Conclusion

This chapter has proposed continuous incremental genetic algorithms (CIGAs) to address continuous incremental learning and training tasks of GA-based classifier agents. Rather than using input attributes in batch as done by normal GAs, CIGAs learn

input attributes one after another. The classification rule sets are also expanded and evolved incrementally to accommodate the new attributes.

Four benchmark data sets are used to evaluate the performance of classifier agents, and they are compared with normal GAs. The simulation results show that CIGAs can be used successfully for continuous incremental training of classifier agents and can achieve better performance than normal GAs using batch-mode training. Four types of CIGAs are also compared to explore further on the effect of different initialization schemes and separate evolution of new attributes. The results show that the use of old group chromosomes (solutions) may help achieve better training and test CR than the use of a single best old chromosome (solution). Separate evolution of new attributes also helps achieve better performance with the cost of extra training time.

Chapter 5

Class Decomposition for GA-based Classifier Agents

5.1 Introduction

In the previous chapters, GAs have been used as the basic algorithms for classification tasks, and IGAs have been developed for incremental learning. However, when GAs are applied to larger-scale real-world classification problems, they still suffer 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 sub-tasks based on certain techniques. Normally, a decomposition approach divides a task into smaller and simpler sub-tasks, supervises the learning of each sub-task, 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 (Anand et al., 1995; Guan and Li, 2002a; Jenkins and Yuhas, 1993; Lu and Ito, 1999). However, most of them are used in Artificial Neural Networks (ANN), very few find their applications in GAs, especially GA-based classification. In

this chapter, we aim to explore the use of class decomposition in GAs and evaluate its performance on classification problems.

The class decomposition approach presented in this chapter 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 form the final solution.

GAs and IGAs are further explored with the introduction of class decomposition. 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.

5.2 Class Decomposition in GA-based Classification

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 (cf. Figure 4.1). GA with class decomposition is significantly different. As shown in Figure 5.1, it generally consists of three steps. Firstly, a problem is divided into k sub-problems in terms of classes. Then, k GA modules are constructed for these sub-problems, and GA in each module will be responsible for evolving a sub-solution. Finally, these sub-solutions are integrated to further obtain the final solution for the original problem. The details for each step are presented in the following subsections.

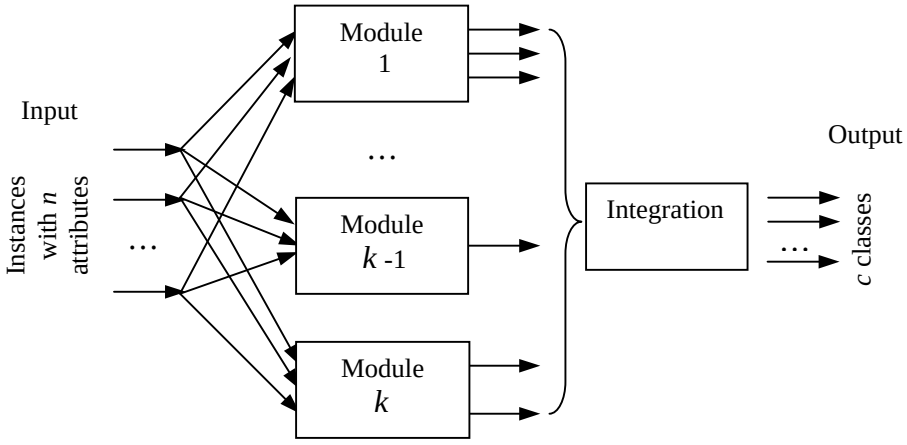


Figure 5.1: Illustration of GA with class decomposition

5.2.1 Class Decomposition

The first step is to decompose a classification problem with a high-dimensional class space into a set of sub-problems with low-dimensional class spaces, in terms of class categories.

Following the previous notations, let us assume a classification problem has *c* classes in the *n*-dimensional pattern space, and *p* vectors $X_i = (x_{i1}, x_{i2}, ..., x_{in})$, $i = 1,2,..., p, \; p >> c$, are given as training patterns. The original classification problem can be denoted as:

$$f : \quad X \rightarrow C \tag{5.1}$$

where, $X \in R^n$ is the set of instances with *n* attributes, and $C \in R^c$ 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:

$$\xi = \{(X_i, C_i)\}_{i=1}^{p} \tag{5.2}$$

Assume the *c*-class problem is divided into k sub-problems, each has c_j ($j = 1,2,...,k$) classes. Denoting the class set for each sub-problem as $C^{(j)}$, we have:

$$C = C^{(1)} \bigcup C^{(2)} \bigcup ... \bigcup C^{(k)} \tag{5.3}$$

where $C \in R^c$, and $C^{(j)} \in R^{c_j}$, and the *j*-th sub-problem can be formulated as finding a certain f_j with a satisfactory classification rate on $C^{(j)}$:

$$f_j : \quad X \rightarrow C^{(j)} \tag{5.4}$$

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.

5.2.2 Parallel Training

With the division of *k* sub-problems, agents can construct *k* GA modules and run them in parallel, as shown in Figure 5.1. 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.

All the training data are fed to each module, but the class categories for each module are different. We denote:

$$\overline{C}^{(j)} = C - C^{(j)}, \quad j = 1,2,...,k \tag{5.5}$$

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

$$\xi_j = \left\{\left(X_q, C_q^{(j)}\right)\right\}_{q=1}^{M} \bigcup \left\{\left(X_q, \overline{C}_q^{(j)}\right)\right\}_{q=M+1}^{p} \tag{5.6}$$

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

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 $C^{(j)}$ correctly, but also ensure that training data for the classes in $\overline{C}^{(j)}$ will not be wrongly classified into the classes in $C^{(j)}$. In other words, for those classes in $\overline{C}^{(j)}$, GA will just distinguish them from the classes in $C^{(j)}$, not necessary to differentiate them in between. As a result, GA in each module will converge more quickly.

These k GA modules are totally independent, because the classes have been fully partitioned without overlapping into several modules. After each module gets a copy of the training patterns, they can be 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 k modules.

5.2.3 Integration

Although each GA module has evolved a portion of the solution, their sub-solutions cannot be simply aggregated as the final one. As discussed earlier, each GA module only classifies the classes in $C^{(j)}$, but not the classes in $\overline{C}^{(j)}$. Therefore, when the sub-solutions are combined together, there may still exist some conflicts among the sub-solutions. For example, rules from different modules may classify an instance into several classes. In order to resolve these conflicts and further improve the classification

accuracy, some intelligent decision rules are employed. The detailed integration process is explained as follows.

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

These rules are based on heuristics. The philosophy behind can be explained as follows. For rule one, when class categories tie for one instance, we trust more the rules for the class achieving higher accuracy. Therefore, the class module achieving higher classification rate is selected. For rule two, when a class module achieves the lowest classification rate, it means that the class is the most difficult to classify, and thus it most likely has more instances incorrectly classified. Therefore, it has high probability that the instance not being classified to any classes belongs to the class whose module achieves the lowest classification rate.

5.3 Experiment Results and Analysis

Several classifier agents have been implemented running on five benchmark data sets to evaluate the class decomposition approach. The data sets chosen are the wine data, iris data, glass data, cancer data, and diabetes data.

The evolution of each module and the integration process are recorded, and some indicative metrics, including initial classification rate (CR), generations, training time, and ending CR, are shown as results. The CR in each generation is the best rate achieved by the whole population. The experiments in Sections 5.3.1 and 5.3.2 use the whole data set as the training data, while the experiments in Section 5.3.3 use tenfold cross validation and the detailed partitioning on data sets are elaborated in that section.

5.3.1 Results and Analysis – GA Based Class Decomposition

Figure 5.2 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.

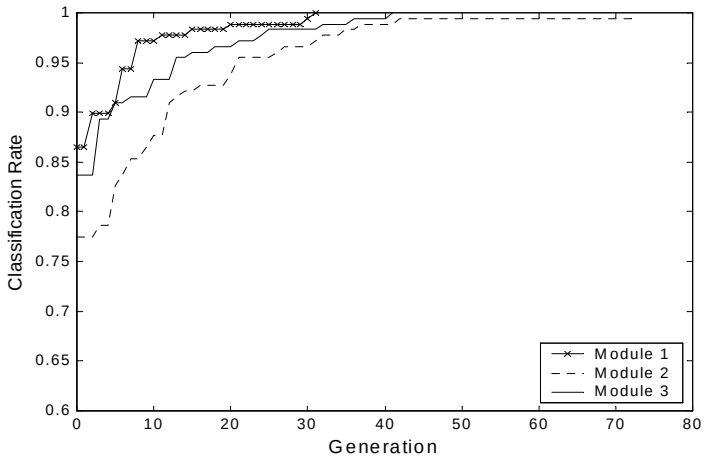


Figure 5.2: The evolution process in three class modules on the wine data

Table 5.1 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. The wine problem is partitioned 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.

Table 5.1: 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. 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.
5. 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.

It is found 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 5.2.3 are employed, an ending CR of 0.9978 is achieved. Then, the whole process including the parallel module training and the integration process is summarized on generation, training time, and

ending CR, and shown in the bottom part of the table. Note that when computing the whole generation and training time, two values are provided as explained in the Note 4 under Table 5.1, 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, no value is listed in the summary.

Comparing the performance of GA with class decomposition to the normal GA, it is found that 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 635.3s to 169.5s for parallel implementation, to 368.0s for serial implementation).

Table 5.2: 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.

Table 5.2 shows the results on the iris data, with the same table structure as Table 5.1. The iris problem is divided into 3 modules, each with one class. The summary part

of Table 5.2 shows 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.

The diabetes problem is divided into two modules each of which deals with one class. From Table 5.3, 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.

Table 5.3: 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=200, generationLimit=200, stagnationLimit=30.

As the glass data have more class categories (6 classes), we have tried different approaches in terms of different partitions on classes. Table 5.4 shows the result of an experiment where the whole problem is partitioned into 3 modules, each with 2 classes. Another two approaches are also tried, i.e., 2-module and 6-module partitioning, which

decompose the original problem into 2 and 6 modules respectively. Table 5.5 shows their comparison.

Table 5.4: 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 5.5: 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

It is found that the class decomposition approaches in all experiments achieve higher ending CR than the normal GA approach. If the three class decomposition approaches are compared with each other, it is found that with the increase of modules used, CR scores higher from 0.6276 to 0.7033 (improved by 12.1%), then to 0.8037

(improved by 28.1%), and training time decreases from 244.7s to 189.9s, then to 185.5s in the case of parallel implementation. This tells us that a finer-grained class decomposition approach will achieve a higher CR and need less training time, 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 (Guan and Li, 2002a), 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. It is found the result of our 2-module approach is comparable to their approach, while our 3-module and 6-module approaches perform even better.

5.3.2 Results and Analysis – IGA Based Class Decomposition

This section steps further to explore the application of class decomposition in IGAs. The IGAs used in this section are dealing with new class acquisition, which have been presented in Chapter 3. It means that an agent already has a solution and needs to evolve a new solution to accommodate new classes.

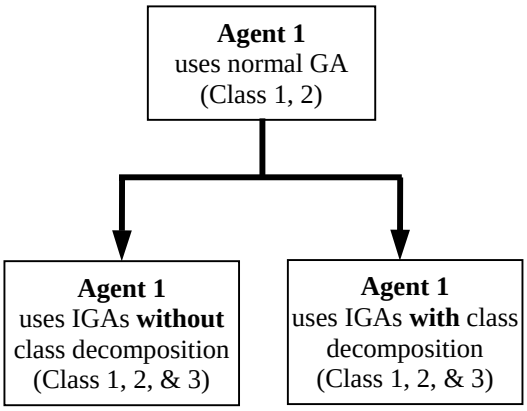


Figure 5.3: Illustration of experiments on IGAs with/without class decomposition

Figure 5.3 illustrates the design of experiments on IGAs with/without class decomposition. Assuming a target classification problem has 3 classes, the detailed experiment steps are explained as follows. (It is easy to derive similar experiment settings for other problems with different number of classes.)

- Assume that an agent only knows classes 1 and 2 at first. It uses normal GA to evolve a rule set on the currently known attributes and classes.
- When the agent knows another class – class 3, it uses IGAs to evolve a new rule set. As there is only one agent, either CE1 or CE2 is used. (The details of CE1 and CE2 can be referred to Section 3.2 and Table 3.1.)
- The class decomposition approach is optionally used with IGAs.

Table 5.6, 5.7, and 5.8 summarize the results of experiments on the wine, iris, and glass data respectively. For each data set, normal GA runs with incomplete set of classes first. With the solutions from GA, IGAs (CE1 and CE2) work on a complete set of classes with or without the class decomposition.

Table 5.6: Comparison of performance of IGAs 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)	
		CE1	CE2	CE1	CE2
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 IGAs 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 5.1.

Table 5.7: Comparison of performance of IGAs 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)	
		CE1	CE2	CE1	CE2
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 IGAs without class decomposition, ruleNumber=6, popSize=100, generationLimit=100, stagnationLimit=30;
3. For IGAs with class decomposition, the parameters are the same as those listed in Table 5.2.

Table 5.8: Comparison of performance of IGAs 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)	
		CE1	CE2	CE1	CE2
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 IGAs without class decomposition, ruleNumber=30, popSize=500, generationLimit=150, stagnationLimit=50;
3. For IGAs with class decomposition, the parameters are the same as those listed in Table 5.4.

It is found from these tables that the approaches with class decomposition perform better than those without class decomposition, for both CE1 and CE2. The former always achieves higher CR and spends less training time in both serial and parallel

implementation. For CE1 in Table 5.7, 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 35.9s to 31.3s 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 5.8 that the ending CR of CE1 improves by 0.047 (i.e. 6.9%), and CR of CE2 improves by 0.053 (i.e. 7.9%).

5.3.3 Generalization Performance and Comparison to Related Work

The generalization performance of evolved rule sets 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 5.9, 5.10, and 5.11 respectively.

Table 5.9: Generalization 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 5.1, except that tenfold cross validation is used.

It is found 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.

Table 5.10: Generalization 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 5.2, except that tenfold cross validation is used.

Table 5.11: Generalization 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.

The performance of our approach is compared with other conventional methods. Table 5.12 shows the classification error rates for different methods on the iris data

(error rate=1-clssifcation rate). The error rates for the first six methods were reported in (Weiss and Kulikowski, 1991), and the last two items are the results from our approach. It is found 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.

Table 5.12: 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**

Note: Error rate=1-clssifcation rate (CR);

For the wine data, Corcoran and Sen (1994) 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. (1999) 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 (2000) 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, Wolberg and Mangasarian (1990) reported a test rate of 95.9% with the use of hyperplanes, and Zhang (1992) 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.

5.4 Conclusion

This chapter has proposed 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 sub-solutions obtained from them are integrated to obtain the final solution by resolving conflicts.

The approach of class decomposition has been explored with the use of GA and IGAs. Four benchmark data sets have been used to evaluate the performance of class decomposition. The results show that class decomposition can help achieve higher classification accuracy with training time reduced.

Chapter 6

Feature Selection for Modular GA-based Classifier Agents

6.1 Introduction

In the previous chapter, a class decomposition approach is proposed to improve the performance of GA-based classifier. This chapter continues to explore the methods of improving performance from the perspective of feature selection. Feature selection plays an important role in finding relevant or irrelevant features in classification. As discussed in Chapter 1, not all features are equally important for a classification problem. Some of them may be redundant or even irrelevant. Better performance may be achieved by discarding some features (Verikas and Bacauskiene, 2002). In other circumstances, we may aim to reduce the dimensionality of input space to save some computation effort, although classification accuracy may be slightly deteriorated.

Principal component analysis (PCA) and linear discriminant analysis are two traditional techniques used to reduce dimensionality by creating new features that are linear combinations of the original ones (Fukunaga, 1990). Fisher's linear discriminant (FLD) is the most popular goodness-score function used in feature selection. It is simple in computation and does not need strict assumptions in the distribution of features. Generally, various combinations of features in the original feature space can be evaluated with the goodness-score function by excluding some features in the

feature space. Because all possible combinations of the features should be tried, the computation effort of such techniques is very high. In order to reduce computation time, some search algorithms are developed, such as knock-out and backtrack tree (Lerner et al., 1994; Gonzalez and Perez, 2001).

In this chapter, a feature selection technique based on relative importance factor (RIF) is presented, which is based on the optimal transformation weights from Fisher's linear discriminant function. The RIF technique can detect features that are irrelevant to the classification problem and remove them from the feature space to improve classification performance in terms of accuracy and complexity. RIF is integrated into the modular GA-based scheme (i.e. GA with class decomposition) by employing it in finding a suitable feature subset for each class module. RIF has been used successfully for modular neural network classifiers (Guan and Li, 2002b). It is aimed here to explore the application of feature selection in the GA domain, which appears to be missing in the literature. A modular-GA based classification approach will be more effective for RIF-based feature selection, as it is easier to find the irrelevant features (IRFs) in individual class, eliminating the interference from other classes. Three benchmark data sets are used to evaluate the performance of RIF. The experiment results show that RIF can help achieve higher classification accuracy with the feature space dimension reduced.

One issue should be clarified here is the use of two terms "attribute" and "feature". Feature selection is a jargon used in the research community, and one feature may consist of several related attributes. In this chapter, one feature corresponds to one attribute, thus they hold the same meaning. In order to conform to the common usage

of feature selection, the term “feature” is used in this chapter instead of the “attribute” in previous chapters.

6.2 Relative Importance Factor (RIF) Feature Selection

Fisher’s linear discriminant (FLD) algorithm projects an *n*-dimensional feature space to a *c-1* dimensional feature space by the function $y_i = w^t x_i$, in the direction *w* that maximizes the criterion function $J(w) = \frac{w^t S_B w}{w^t S_W w}$, where S_B is called as the between-class scatter matrix, and S_W the within-class scatter matrix (Duda and Hart, 2000).

As our feature selection technique is employed in each class module which only distinguishes two class categories, i.e., $C^{(j)}$ and $\overline{C}^{(j)}$ (cf. Section 5.2), the projected feature space is one-dimensional (projected on one line) in this situation. Hence, the transformation matrix *w* that maximizes the criterion function *J(w)* is a vector $w = [w_1 \quad w_2 \quad ... \quad w_n]^t$. The elements in the transformation vector *w* can be viewed as weights for different features in the original feature space respectively. Thus, the feature selection technique can be simplified based on one observation: in an optimal transformation vector *w* of the FLD, a larger w_i means that the i_{th} feature is likely to be more relevant to the module and a smaller w_i means the i_{th} feature is likely to be less relevant to the module. This observation forms the basis of the proposed RIF technique.

However, the weights obtained directly from the transformation vector *w* are not normalized. In order to derive a common feature selection metric across different sets of features in different problems, a relative importance factor (RIF), $r = [r_1 \quad r_2 \quad ... \quad r_n]^t$, is proposed instead of using the transformation vector *w* directly

for feature selection. The RIF is obtained through the following two steps (Guan and Li, 2002b):

I. Normalize the length of the transformation vector *w*.

Since RIF is evaluating the relative importance of features, the relative weights of the features in the transformation vector *w* is more important, which can be obtained through normalization:

$$w' = \frac{w}{\sqrt{\sum_{i=1}^{n} (w_i)^2}} \qquad (6.1)$$

where w_i is the weight of the *i*th feature in w, w' is the normalized transformation vector, and *n* is the number of features.

II. Render the importance factor independent from the number of features.

Since different problems have different numbers of features in their feature spaces, it is necessary to make the RIF values independent of the number of features in the feature space. This is achieved by the following function:

$$r = \frac{n}{\sum_{i=1}^{n} |w'_i|} w' \qquad (6.2)$$

Combining (6.1) and (6.2), RIF values can be obtained from the transformation vector *w* directly as:

$$r = \frac{n}{\sum_{i=1}^{n} \left| \frac{w_i}{\sqrt{\sum_{i=1}^{n} (w_i)^2}} \right|} * \frac{w}{\sqrt{\sum_{i=1}^{n} (w_i)^2}} = \frac{n}{\sum_{i=1}^{n} |w_i|} w \qquad (6.3)$$

As a result, the elements of *r* represent the normalized importance of different features, which are independent from the magnitude of *w* and the number of features in the feature space.

The proposed RIF technique requires much less computation time. Assume there are *n* input features in the original feature space. In order to obtain the relative importance of each feature, *n* FLD computations with *n-1* features included is needed each time using traditional knock-out techniques. With the simplified method, the relative importance of each feature in the module (RIF value) can be obtained in one computation with all *n* features included.

RIF values are used as the feature selection tool in our modular GA-based classification. The feature selection technique can be summarized as follows:

Step 1: Calculate the Fisher's transformation vector *w* with respect to all features in the input feature space for each class module.

Step 2: Calculate the RIF value for each feature by using formula (6.3).

Step 3: Set a threshold value *T1*. If the RIF value of a feature is less than *T1*, it can be considered as an irrelevant feature (IRF).

Step 4: Remove IRFs from each module. A new set of features for each class module will be selected.

Step 5: Modular GA-based classification is then performed based on the new feature set for each class module, as presented in Chapter 5.

6.3 Experiment Results and Analysis

Several classifier agents are implemented running on three benchmark data sets to evaluate our approaches. The data sets chosen are the wine data, glass data, and

diabetes1 data. Each data set is partitioned into two parts with an equal number of instances. One half is for training, and the other half is for testing. The training data are used to train the rule sets, and the test data are used to evaluate the generalization performance of resulting rule sets.

Each experiment follows the five-step process to determine the IRFs and evaluate the performance of classifiers with those IRFs removed. Then, by comparing to the performance of a classifier with the complete feature set, it can be examined whether the performance of modular classifiers have improved or degraded as a result of removing IRFs.

Table 6.1: RIF value for each feature in different class modules - wine data

RIF	Class=1	Class=2	Class=3
Feature 1	1.8773	1.0938	0.9735
Feature 2	0.1974	0.4703	0.4643
Feature 3	3.9760	3.1118	2.8734
Feature 4	0.4459	0.1332	0.1047
Feature 5	**0.0026**	**0.0005**	**0.0014**
Feature 6	0.9748	**0.0398**	0.1324
Feature 7	1.7762	1.4848	1.6627
Feature 8	0.1335	2.8563	2.8634
Feature 9	0.6493	0.2832	0.2381
Feature 10	0.1636	0.5319	0.5601
Feature 11	0.9828	2.6315	2.6018
Feature 12	1.8124	0.3605	0.5223
Feature 13	**0.0082**	**0.0023**	**0.0017**

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. The threshold value is chosen as T1=0.1; those values below the threshold are highlighted.

Table 6.1 shows the RIF value for each feature of the wine data in different class modules. If the threshold value is chosen as T1=0.1, feature 5 and 13 are regarded as common IRFs in class module 1, 2 and 3, while feature 6 is regarded as an IRF in class module 2 only. Therefore, feature 5 and 13 are removed from the feature set for

module 1 and 3, and feature 5, 6, and 13 are removed from the feature set for module 2. Table 6.2 shows the comparison of the classifier performance with/without feature selection on the wine data. It is found that the test CRs are improved in all modules as a result of removing all IRFs. For example, the test CR of module 2 gets an improvement from 0.8371 to 0.8657 by 3.4%. In addition, the overall test CR is improved with an increase from 0.8652 to 0.8831 by 2.1%.

Table 6.2: Performance of the classifier with/without feature selection - wine data

		Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)
Using All Features	Initial CR	0.8876	0.7618	0.8685
	Generations	23.3	48.6	38.5
	T. time (s)	31.2	64.3	50.1
	Training CR	0.9989	1.0	0.9921
	Test CR	0.9033	0.8371	0.8703
		Integration		
	Training CR	0.9966		
	Test CR	0.8652		
Removing all IRFs from each module	Initial CR	0.8899	0.7787	0.8708
	Generations	42.7	55.7	32.1
	T. time (s)	51.7	68.1	39.7
	Training CR	0.9944	0.9933	0.9955
	Test CR	0.9067	0.8657	0.8833
		Integration		
	Training CR	0.9933		
	Test CR	0.8831		

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

It is noted that the number of generations and training time needed for each module become either shorter (for module 3) or longer (for module 1 and 2), after the IRFs are removed. This means that the classifier with a reduced feature set either converges quickly or needs more generations to reach a higher performance. Furthermore,

module 2 obtains the greatest improvement, which is mainly due to the removal of three features.

Table 6.3 shows the RIF values for each feature of the glass data in different class modules. The threshold value is chosen as 0.1. It is found that different features are regarded as IRFs in different class modules, as highlighted in the table. All IRFs are removed from each class module. The performance of the classifier trained with the complete set of features and the one with IRFs removed are shown in Table 6.4 and 6.5 respectively.

Table 6.3: RIF value for each feature in different class modules - glass data

RIF	Class=1	Class=2	Class=3	Class=4	Class=5	Class=6
Feature 1	6.5212	6.8322	8.9045	6.8777	8.0957	8.2007
Feature 2	0.3349	0.3327	**0.0066**	0.2782	0.2675	0.1111
Feature 3	0.3951	0.2882	**0.0002**	0.2596	**0.0984**	**0.0981**
Feature 4	0.2575	0.2605	**0.0288**	0.2204	**0.0311**	**0.0843**
Feature 5	0.3708	0.3064	**0.0231**	0.2289	0.1945	0.1293
Feature 6	0.3709	0.3183	**0.0106**	0.5747	0.1378	0.1062
Feature 7	0.3427	0.3011	**0.0064**	0.3070	0.1389	**0.0815**
Feature 8	0.3656	0.3241	**0.0040**	0.2076	**0.0158**	0.1826
Feature 9	**0.0413**	**0.0365**	**0.0156**	**0.0458**	**0.0204**	**0.0062**

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. The threshold value is chosen as T1=0.1; those values below the threshold are highlighted.

Comparing the corresponding module elements in Table 6.4 and 6.5, it is noted that the ending CR for each module is either improved or degraded slightly after IRFs are removed from the six modules, i.e., the test CRs of module 1, 2, and 3 have improved, while the test CRs of module 4, 5, 6 have degraded. However, the overall test CR is still improved from 0.4224 to 0.4944 (17%) after the integration process. This tells us that that removing IRFs may result in performance deterioration in some modules, which also means the selection of a suitable threshold is crucial, but it may still be beneficial to the overall performance.

Table 6.4: Performance of the classifier with the complete set of features - glass data

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)	Module 4 (Class=4)	Module 5 (Class=5)	Module 6 (Class=6)
Initial CR	0.7308	0.7224	0.9187	0.9523	0.9664	0.9561
Generations	125.2	128.1	67.0	50.1	30.4	42.0
T. time (s)	89.6	93.7	40.5	29.7	18.2	32.7
Training CR	0.9421	0.9178	0.9299	0.9944	0.9963	0.9972
Test CR	0.6776	0.6196	0.8832	0.9299	0.9411	0.9449
	Integration (Class=1, 2, 3, 4, 5, 6)					
Training CR	0.7738					
Test CR	0.4224					

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=5, popSize=100, generationLimit=150, stagnationLimit=50;

Table 6.5: Performance of the classifier with all IRFs removed - glass data

	Module 1 (Class=1)	Module 2 (Class=2)	Module 3 (Class=3)	Module 4 (Class=4)	Module 5 (Class=5)	Module 6 (Class=6)
Initial CR	0.7346	0.7121	0.9150	0.9570	0.9636	0.9701
Generations	127.5	123.8	50.1	24.8	59.3	39.0
T. time (s)	87.3	83.9	33.6	14.2	34.0	29.0
Training CR	0.9243	0.9075	0.9160	0.9991	0.9822	0.9953
Test CR	0.7056	0.6785	0.9243	0.9168	0.9234	0.9252
	Integration (Class=1, 2, 3, 4, 5, 6)					
Training CR	0.7720					
Test CR	0.4944					

Notes:
1. mutationRate=0.01, crossoverRate=1, survivorsPercent=50%;
2. For each module, ruleNumber=5, popSize=100, generationLimit=150, stagnationLimit=50;

Table 6.6: RIF value for each feature in different class modules – diabetes1 data

RIF	Class=1/ Class=2
Feature 1	0.8291
Feature 2	2.8045
Feature 3	0.6738
Feature 4	**0.0366**
Feature 5	0.3618
Feature 6	2.1049
Feature 7	0.8168
Feature 8	0.3725

Notes:
1. Each row in the table records the RIF value for each feature under each class module;
2. The threshold value is chosen as T1=0.1; those values below the threshold are highlighted.

Table 6.7: Performance of the classifier with different set of features – diabetes1 data

Using All Features	Module 1 (Class=1)	Module 2 (Class=2)
Initial CR	0.6966	0.6852
Generations	179.1	186.4
T. time (s)	366.5	358.9
Training CR	0.8542	0.8234
Test CR	0.7336	0.7279
	Integration (Class=1, 2)	
Training CR	0.8388	
Test CR	0.7365	

Removing Feature 4	Module 1 (Class=1)	Module 2 (Class=2)
Initial CR	0.6958	0.7047
Generations	179.4	159.7
T. time (s)	387.6	316.6
Training CR	0.8552	0.8167
Test CR	0.7349	0.7385
	Integration (Class=1, 2)	
Training CR	0.8411	
Test CR	0.7477	

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

The RIF values are still used to determine the IRFs of the diabetes1 data. Since the diabetes1 data have only 2 classes, each feature has the same RIF value in the two class modules as shown in Table 6.6. The threshold is chosen as T1=0.1, and feature 4 is regarded as the IRF for both modules.

Feature 4 is removed from both modules and the resulting performance of classifier is reported in Table 6.7, which compares the classifier performance under two scenarios, i.e., the special case when feature 4 is removed and the normal case when all features are used for classification. It is noted that test CRs are improved for both modules, and training CR is improved for module 1. As for the final training CR and test CR, they all are improved after feature 4 is removed from the feature set. These results on the diabetes1 data have again shown that the effect of removing IRFs successfully reduces the feature space dimension and helps improve the classifier performance.

As the diabetes1 data has only two classes, and feature 4 is the common IRF for both class modules, a general non-modular GA approach with RIF feature selection

technique is also applicable. An experiment with the non-modular GA approach has been conducted to contrast with the modular GA approach, and the results are shown in Table 6.8.

Table 6.8: Performance of the non-modular GA classifier - diabetes1 data

	Using All Features	**Removing Feature 4**
Initial CR	0.6273	0.6393
Generations	178.1	184.0
T. time (s)	581.6	616.8
Training CR	0.7568	0.7747
Test CR	0.6961	0.7289

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

It is still found that removing feature 4 (IRF) improves both the training CR (from 0.7568 to 0.7747 by 2.4%) and test CR (from 0.6961 to 0.7289 by 4.7%) with the non-modular GA approach. If comparing the corresponding results of these two approaches, it is shown that the performance of the non-modular approach is inferior to that of the modular approach in terms of the final training CR and test CR, which shows that class decomposition approach can improve the classifier performance.

6.4 Discussions

6.4.1 Reduction in Rule Set Complexity

As the IRFs are removed from the feature space, the resulting rule sets for a classification problem become shorter and more concise, i.e, the complexity is reduced. When these reduced rule sets are used to classify data, it is apparent that the classifier can achieve a higher throughput.

```
1. IF (0.81<=X2<=1.01) THEN Class=1
2. IF (0.71<=X6<=0.88) AND (0.07<=X8<=0.51) THEN Class=1
3. IF (0.53<=X6<=0.59) AND (0.36<=X7<=0.88) THEN Class=1
4. IF (0.45<=X2<=0.47) AND (0.49<=X3<=0.64) AND (0.61<=X4<=0.70) THEN Class=1
5. IF (0.43<=X2<=0.63) AND (0.28<=X3<=0.83) AND (0.22<=X4<=0.40) AND
      (0.45<=X6<=0.87) AND (0.01<=X7<=0.42) AND (0.12<=X8<=0.32) THEN Class=1
6. IF (0.00<=X1<=0.12) AND (0.83<=X2<=0.91) AND (0.38<=X3<=0.99) AND
      (0.46<=X4<=0.46) AND (0.46<=X5<=0.55) AND (0.14<=X6<=0.70) AND
      (0.46<=X7<=0.75) THEN Class=1
7. IF (0.03<=X1<=0.76) AND (0.79<=X2<=0.92) AND (0.18<=X4<=0.86) THEN Class=1
8. IF (0.16<=X3<=0.64) AND (0.32<=X4<=0.47) AND (0.57<=X7<=0.65) THEN Class=1
9. IF (0.38<=X1<=0.87) AND (0.45<=X2<=1.00) AND (0.18<=X7<=0.39) THEN Class=1
10. IF (0.18<=X5<=0.53) AND (0.48<=X7<=0.64) THEN Class=1
11. IF (0.59<=X2<=0.78) AND (0.41<=X6<=0.84) AND (0.35<=X8<=0.64) THEN Class=1
12. IF (0.50<=X2<=0.75) AND (0.89<=X3<=0.91) AND (0.71<=X6<=0.95) AND
       (0.44<=X7<=0.72) THEN Class=1
13. IF (0.14<=X1<=0.42) AND (0.41<=X7<=1.01) THEN Class=1
14. IF (0.09<=X1<=0.30) AND (0.57<=X4<=0.60) AND (0.70<=X5<=0.73) AND
       (0.10<=X6<=0.34) AND (0.35<=X7<=0.37) THEN Class=1
15. IF (0.44<=X3<=0.82) AND (0.24<=X5<=0.54) AND (0.30<=X7<=0.72) AND
       (0.36<=X8<=0.83) THEN Class=1
```

Figure 6.1: Rule set for module 1 with all features - diabetes1 data

```
1. IF (0.59<=X2<=0.98) AND (0.27<=X3<=0.71) AND (0.47<=X7<=0.58) THEN Class=1
2. IF (0.36<=X1<=0.55) AND (0.70<=X2<=0.97) AND (0.54<=X3<=0.91) THEN Class=1
3. IF (0.12<=X3<=0.96) AND (0.18<=X5<=0.51) AND (0.38<=X8<=0.75) THEN Class=1
4. IF (0.14<=X2<=0.97) AND (0.82<=X3<=0.97) AND (0.62<=X5<=1.00) AND
      (0.90<=X7<=0.91) AND (0.27<=X8<=0.93) THEN Class=1
5. IF (0.22<=X3<=1.00) AND (0.24<=X5<=0.33) AND (0.33<=X6<=0.73) AND
      (0.25<=X7<=0.80) AND (0.11<=X8<=0.30) THEN Class=1
6. IF (0.54<=X2<=0.68) AND (0.39<=X6<=1.00) AND (0.20<=X8<=0.59) THEN Class=1
7. IF (0.46<=X2<=0.95) AND (0.58<=X5<=0.91) THEN Class=1
8. IF (0.57<=X1<=0.91) AND (0.03<=X2<=0.17) AND (0.59<=X5<=0.75) THEN Class=1
9. IF (0.65<=X3<=0.78) AND (0.27<=X7<=0.42) AND (0.24<=X8<=0.99) THEN Class=1
10. IF (0.31<=X1<=0.43) AND (0.37<=X3<=0.73) AND (0.42<=X5<=0.50) AND
       (0.27<=X7<=0.29) AND (0.15<=X8<=0.93) THEN Class=1
11. IF (0.02<=X6<=0.11) THEN Class=1
12. IF (0.81<=X2<=1.00) THEN Class=1
13. IF (0.48<=X6<=0.50) AND (0.04<=X7<=0.70) AND (0.37<=X8<=0.69) THEN Class=1
14. IF (0.71<=X1<=0.76) AND (0.86<=X3<=0.95) AND (0.30<=X6<=0.34) THEN Class=1
15. IF (0.43<=X1<=0.70) AND (0.09<=X5<=0.73) THEN Class=1
```

Figure 6.2: Rule set for module 1 with feature 4 removed – diabetes1 data

Figure 6.1 and 6.2 show the resulting rule sets for class module 1 of the diabetes1 data before and after feature selection respectively - removing feature 4 (cf. Table 6.6 and 6.7). It can be seen that feature 4 (X4 in the rule set) does not appear in the second rule list, as it has been removed from the feature space. The total number of attribute elements can be used to evaluate the improvement on complexity. It is found that the number of elements is reduced from 50 elements in Figure 6.1 to 45 elements in Figure

6.2, with a reduction rate as 10%. Therefore, with a reduced feature space, the rule set complexity is also reduced.

The complexity of the rule sets for the wine and glass data is also measured. As a result, a reduction rate of 17.9% and 25% is achieved respectively. It is found that the rule set for glass data achieves the highest reduction rate, as more IRFs are removed from the feature space.

6.4.2 Comparison to the Application of RIF in Neural Networks

RIF has also been applied successfully to neural networks for feature selection. According to the results reported in (Guan and Li, 2002b), the diabetes1 data are also used to test the effect of RIF in neural networks. When feature 4 (IRF) is removed, the classifier achieves a classification error of 23.96% on the test data (25% of the whole data), which is equal to a test CR of 0.7604. According to the results reported in Table 6.7, the modular GA-approach with RIF achieves a test CR of 0.7477 on the test data (50% of the whole data). In order to be fair, we also use the 25% of the whole data as test data, and achieve a test CR of 0.7531. We can conclude that the performance of our modular GA approach with RIF is comparable to that of neural networks.

6.4.3 Other Issues of RIF

The integration of RIF feature selection with a modular GA approach with class decomposition brings forth some advantages. First, as each module is only responsible for one class, it is easier to use RIF values to find the IRFs in that particular class, eliminating the interference from other classes. Second, RIF requires relatively small computation cost compared to other feature selection techniques such as the knock-out

technique. It is based on the statistic distribution of features in the input feature space and needs only one calculation of FLD transformation weights. Furthermore, RIF is independent from the learning algorithms, and it can also be used with other soft computing techniques such as neural network and other types of classifiers such as Bayes classifiers.

The selection of a good threshold value for RIF is an important issue. In most cases, if a larger threshold value is used, more features can be removed and complexity can be further reduced. However, too large a threshold value may induce information loss, so that classification accuracy can be affected.

The feature selection techniques presented have different effects on the training and test performance of the classifiers. From the experiment results, it is found that training CR sometimes degrades a little while test CR improves. However, test CR generally improves more than training CR degrades. The test performance is of more importance because it represents the generalization capability of a classifier, thus it is worth using the proposed feature selection techniques to reduce the feature space dimension.

6.5 Conclusion

This chapter has employed a new feature selection technique based on relative importance factor (RIF) to find irrelevant features (IRFs) in the input domain of a classification problem. RIF is employed in modular GA-based classifier agents, which use the class decomposition approach presented in the last chapter. RIF is used to detect the IRFs in each class module. By removing these features, it aims to improve classification accuracy and reduce the dimensionality of the classification problems.

Three benchmark classification data sets have been used to evaluate the proposed approaches. The experiment results show that RIF can be used as a simple and yet effective feature selection technique to determine irrelevant features and help achieve higher classification accuracy with the feature space dimension reduced. In the meantime, the complexity of the resulting rule sets is also reduced which means the modular classifiers with IRFs removed will be able to classify data with a higher throughput.

Chapter 7

Conclusions and Further Research

7.1 Conclusions

This thesis has explored incremental learning of classifier agents. Traditional incremental learning algorithms are generally based on neural networks and machine learning techniques, and few touch on the use of evolutionary algorithms. The approaches presented in this thesis have embodied classifier agents with genetic algorithms, and further explored incremental learning with specially designed algorithms.

On the basis of normal GAs, incremental genetic algorithms (IGAs) have been proposed for incremental learning of classifier agents. Various learning approaches for IGAs have been designed to cope with different changes in either single-agent or multi-agent environments. These approaches have been evaluated by examining the performance of classifier agents with benchmark classification data sets. Specifically, IGAs have been exploited in two circumstances. One is that classifier agents incrementally learn new attributes, and the other one tackles the case when the new classes are incrementally available. IGAs have been adapted to these two circumstances, and various optional initialization schemes have been proposed and compared. The simulation results presented in Chapter 2 and 3 have shown that IGAs

can be successfully used for incremental learning of new attributes and classes and speeds up the learning process as compared to traditional GAs.

As a further research on incremental learning, continuous incremental genetic algorithms (CIGAs) have been proposed for continuous incremental learning where new attributes are introduced one by one. Different approaches of CIGAs have been evaluated, and their performance has been compared with normal GAs. The simulation results have shown that CIGAs can be used to continuously learn new attributes and evolve new rule sets to accommodate the new attributes, and the final performance achieved is also better than normal GAs.

Finally, as further steps to enhance the performance of classifier agents, two additional approaches, i.e., class decomposition and feature selection, have been explored. The class decomposition approach partitions a classification problem into several class modules in the class domain. These modules are trained in parallel and independently, and the results obtained are integrated to form the final solution by resolving conflicts. GAs and IGAs have been evaluated with the use of class decomposition. The simulation results have shown that the class decomposition approach helps GAs and IGAs achieve a higher classification rate with training time reduced.

For feature selection, a new technique, relative importance factor (RIF), has been employed to find irrelevant features in the input domain. By removing these features, the classification accuracy has been improved and the dimensionality of the classification problems reduced. The complexity of the resulting rule sets has also been reduced, and it means classifier agents are able to classify data with a higher throughput.

A number of approaches have been proposed in this thesis. In summary, IGAs and CIGAs have been proposed for incremental learning/training, while class decomposition and feature selection have been proposed to improve performance. They have been used for different purposes. Table 7.1 lists the rules of thumb for the selection among IGAs and CIGAs. Furthermore, class decomposition can be used with GAs and IGAs, especially when a classification problem is complex. For example, the instance number reaches some thousand, or the number of attributes or classes is around ten or more. If the reduction in the feature space is allowed or it is as an objective itself, the proposed RIF method may be a good choice.

Table 7.1: Rules of thumb for the selection of IGA and CIGA approaches

Rule 1: If new attributes or classes are available in a time-varying manner, IGAs should be considered.

Rule 2: If both group chromosomes from the old solutions and evolved rule sets for new attributes/classes are available, IS4/RI2 should be chosen.

Rule 3: If only the best chromosome from the old solutions and evolved rule sets for new attributes/classes are available, IS2/RI1 should be chosen.

Rule 4: If only the group chromosomes from the old solutions are available, IS3/CE2 should be chosen.

Rule 5: If only the best chromosome from the old solutions is available, IS1/CE1 should be chosen.

Rule 6: If a data set is presented with all attributes, classes and instances being available, CIGAs are recommended for incremental training.

Rule 7: If the resource (e.g. computer memory) is enough for the single-attribute evolution module, CIGA4 is recommended. Otherwise, CIGA3 is recommended.

7.2 Future Research

While this thesis has made some contribution on incremental evolution of classifier agents with new approaches of incremental genetic algorithms and other methods to improve their performance, several issues are still open. Continuing research would allow us to further expand the findings of this thesis.

First, the rule-based approach for classification problems can be extended to fuzzy rule-based classification systems, by adding the membership function and grade of certainty. Fuzzy logic permits vague information, knowledge and concepts to be used in an exact mathematical manner. This enables qualitative and imprecise reasoning statements to be incorporated within rule-bases so producing simpler, more intuitive and better behaved models. A GA-fuzzy approach is a possible direction for our future work.

Second, when classifier agents exchange attributes, classes, or training data, some overlapping can happen. For example, the exchanged training data among agents may have some overlapping. Therefore, a classifier agent needs to analyze the incoming data, and combine them with its own current data using certain methods. Furthermore, if elements/rules exchanged among agents have some identical parts, agents may need more advanced mechanisms to detect them and integrate them properly. In addition, in a multi-agent environment, interaction among agents can be more complex than discussed in this thesis. To implement complete collaborative incremental learning for classifier agents, many issues should be carefully considered. For example, the communication channel among agents should be securely established and message

format should be carefully predefined. Furthermore, the interaction among agents can be continuous so that agents can realize frequent exchange for co-evolution.

Third, for the approaches of class decomposition and feature selection, there is also room for improvement. For class decomposition, a few kinds of intelligence can be embedded. For example, classifier agents can have some intelligence on determining the decomposition pattern which includes the size of modules, the allocation of classes into modules. Agents can autonomously try on different combinations to find the optimal decomposition pattern. On the other hand, human-agent interaction may provide a better way to boost the intelligence of agents and establish a control and feedback channel between users and agents. Currently, the RIF feature selection is built on a complete class decomposition, which means the output classes are partitioned in a non-overlapping manner, i.e. each module only tackles one class. Alternatively, agents can have some degree of overlapping in class decomposition, which will be more robust in the presence of faults and may lead to further improvement on classification accuracy.

Finally, reducing training time is always an important pursuit in designing online systems, as they need to react to the environment promptly. This thesis has proved that incremental genetic algorithms and the other approaches can save training time compared to normal GAs. In order to further reducing training time, future research may be focused on the adoption of parallel genetic algorithms and parallel implementation of multi-agent systems.

References

Abul, O., Polat, F., and Alhajj, R. Multiagent reinforcement learning using function approximation, IEEE Trans. on Systems, Man, and Cybernetics, Part C, 30 (4), 485-497, 2000.

Adeli, H. and Hung, S.L. Machine learning: neural networks, genetic algorithms, and fuzzy systems, New York: Wiley, 1995.

Anand, R., Mehrotra, K., Mohan, C.K., and Ranka, S. Efficient classification for multiclass problems using modular neural networks, IEEE Trans. on Neural Networks, 6 (1), 117-124, 1995.

Apté, C., Hong, S.J., Hosking, J., Lepre, J., Pednault, E., and Rosen, B. Decomposition of heterogeneous classification problems**.** In Proc. of Second International Symposium Advances in Intelligent Data Analysis, Reasoning about Data, Lecture Notes in Computer Science, vol. 1280, Springer, 17-28, 1997.

Back, T., Hammel, U., and Schwefel, H.P. Evolutionary computation: comments on the history and current state, IEEE Trans. on Evolutionary Computation, 1 (1), 3-17, 1997.

Back, T., Hoffmeister, F., and Schwefel, H.P. A survey of evolution strategies. In Proc. of the 4th International Conference on Genetic Algorithms, San Diego, CA, 2-9, 1991.

Bala, J., Huang, J., Vafaie, H., DeJong, K., and Wechsler, H. Hybrid learning using genetic algorithms and decision tress for pattern classification. In Proc. of the 14th

International Joint Conference on Artificial Intelligence, Montreal, Canada, 719-724, 1995.

Bandyopadhyay, S. and Pal, K. Pixel classification using variable string genetic algorithms with chromosome differentiation, IEEE Trans. on Geoscience and Remote Sensing, 39 (2), 303-308, 2001.

Battiti, R. Using mutual information for selecting features in supervised neural net learning, IEEE Trans. on Neural Networks, 5 (4), 537-550, 1994.

Blake, C.L. and Merz, C.J. 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.

Bradshaw, J.M. Software Agent, MA: MIT Press, 1997.

Cantu-Paz, E. Markov chain models of parallel genetic algorithms, IEEE Trans. on Evolutionary Computation, 4 (3), 216-226, 2000.

Cantu-Paz, E. Efficient and Accurate Parallel Genetic Algorithms, Boston, MA: Kluwer Academic, 2000.

Caragea, D., Silvescu, A., and Honavar, V. Towards a theoretical framework for analysis and synthesis of distributed and incremental learning agents. In Proc. of Workshop on Distributed and Parallel Knowledge Discovery, Boston, US, 2000.

Corcoran, A.L. and Sen, S. Using real-valued genetic algorithm to evolve rule sets for classification. In Proc. of the 1st IEEE Conference on Evolutionary Computation, Orlando, US, 120-124, 1994.

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

Dalché-Buc, F. and Ralaivola, L. Incremental learning algorithms for classification and regression: local strategies. In Proc. of American Institute of Physics Conference, 2001.

Dash, M. and Liu, H. Feature selection for classification, Intelligent Data Analysis, 1 (3), 131-156, 1997.

DeJong, K.A. Learning with genetic algorithms: an overview, Machine Learning, 3, 121-138, 1988.

DeJong, K.A. and Spears, W.M. Learning concept classification rules using genetic algorithms. In Proc. of the 1991 International Joint Conference on Artificial Intelligence, 651-656, 1991.

Duda, R.O., Hart, P.E., and Stork, D.G. Pattern Classification, New York: Wiley, 2nd Edition, 2000.

Enee, G. and Escazut, C. Classifier systems evolving multi-agent system with distributed elitism. In Proc. of the 1999 Congress on Evolutionary Computation, 1740-1746, 1999.

Engelbrecht A.P. and Brits, R. A clustering approach to incremental learning for feedforward neural networks. In Proc. of Int. Joint Conf. Neural Network, vol. 3, 2019-2024, 2001.

Etzioni, O. and Weld, D.S. Intelligent agents on the Internet: fact, fiction, and forecast, IEEE Expert, 10 (4), 44-49,1995.

Falco, I.D., Cioppa, A.D., and Tarantino, E. Discovering interesting classification rules with genetic programming, Applied Soft Computing, 1, 257-269, 2002.

Ferber, J. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 1999.

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

Fogel, D.B. Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, Piscataway: IEEE Press, 1995.

Fogel, D.B. Fogel, L.J. and Atmar, J.W. Meta-evolutionary programming. In Proc. of the 25th Aslimar Conf. on Signals, Systems and Computers, Maple Press, 540-545, 1991.

Franklin, S., and Graesser, A. Is it an agent or just a program? a taxonomy for autonomous agents. In Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages, New York: Springer-Verlag, 1996.

Fu, L., Hsu, H., and Principe, J.C. Incremental backpropagation learning networks, IEEE Trans. on Neural Networks, 7 (3), 757-761, 1996.

Fukunaga, K. Introduction to Statistical Pattern Recognition, 2nd ed., Boston: Academic Press, 1990.

Giraud-Carrier, C. A note on the utility of incremental learning, AI Communications, 13 (4), 215-223, 2000.

Giraud-Carrier, C. and Martinez, T. An incremental learning model for commonsense reasoning. In Proc. of the Seventh International Symposium on Artificial Intelligence, 134-141, 1994.

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

Gonzalez, A. and Perez, R. Selection of relevant features in a fuzzy genetic learning algorithm, IEEE Trans. on Systems, Man and Cybernetics, Part B, 31 (3), 417-425, 2001.

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

Guan, S. U., Ngoo, C.S., and Zhu, F.M. Handy broker - an intelligent product-brokering agent for m-commerce applications with user preference tracking, Electronic Commerce Research and Applications, Elsevier Science, 1(3-4), 314-330, 2002.

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

Guan, S.U. and Li, S.C. Parallel growing and training of neural networks using output parallelism, IEEE Trans. on Neural Networks, 13 (3), 542-550, 2002. (2002a)

Guan, S.U. and Li, P, Feature selection for modular neural network classifiers, accepted, Journal of Intelligent Systems, 12 (3), 173-200, 2002. (2002b)

Guan, S.U. and Liu, J. Incremental neural network training with an increasing input dimension, submitted to Journal of Intelligent Systems, 2002.

Guan, S.U., Zhu, F.M., and Ko, C.C. Agent fabrication and authorization in agent-based electronic commerce. In Proc. of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce, Wollongong, Australia, 528-534, 2000.

Guan, S.U. and Zhu, F.M. Agent fabrication and its implementation for agent-based electronic commerce, International Journal of Information Technology & Decision Making, 1 (3), 473-489, 2002. (2002a)

Guan, S.U. and Zhu, F.M. Learning of classifier agents based on incremental genetic algorithms. In Proc. of 2002 IASTED International Conference on AI and Applications, Málaga, Spain. (2002b)

Guan, S.U. and Zhu, F.M. Incremental learning of classifier agents with genetic algorithms, under second review, IEEE Trans. on Evolutionary Computation, 2002. (2002c)

Guan, S.U. and Zhu, F.M. Continuous incremental training for GA-based classifiers, submitted to IEEE Trans. on Systems, Man and Cybernetics, Part C., 2002. (2002d)

Guan, S.U. and Zhu, F.M. Evolutionary e-commerce agents with ontology exchange. In Proc. of IASTED International Conference on Internet and Multimedia Systems and Applications, 47-51, 2002. (2002e)

Guan, S.U. and Zhu, F.M. Incremental learning of collaborative classifier agents with new class acquisition - an incremental genetic algorithm approach, International Journal of Intelligent Systems, 18 (11), 1173-1193, 2003.

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

Guan, S.U. and Zhu, F.M. and Li, P, Modular feature selection using relative importance factors, International Journal of Computational Intelligence and Applications, 4 (1), 1-19, 2004. (2004b)

Haynes, T.D. and Wainwright, R.L. A simulation of adaptive agents in a hostile environment. In Proc. of the ACM Symposium on Applied Computing, ACM Press, 1995.

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

Holland, J.H. Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based systems. In Machine Learning: An Artificial Intelligence Approach, ed. by R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, Los Altos, CA: Morgan Kaufmann, 1986.

Ishibuchi, H., Murata, T., and Nakashima, T. Genetic-algorithm-based approaches to classification problems. In Fuzzy Evolutionary Computation, ed. by W. Pedrycz, Boston: Kluwer Academic, 127-154, 1997.

Ishibuchi, H., Nakashima, T., and Murata, T. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems, IEEE Trans. on Systems, Man and Cybernetics, Part B, 29 (5), 601-618, 1999.

Jenkins, R.E. and Yuhas, B.P. A simplified neural network solution through problem decomposition: the case of the truck backer-upper, IEEE Trans. on Neural Networks 4 (4), 718-720, 1993.

Jennings, N.R., Corera, J.M., and Laresgoiti, I. Developing industrial multi-agent systems. In Proc. of the First International Conference on Multi-agent Systems, 423-430, 1995.

Kang, H. Evolvable cellular classifiers. In Proc. of the 2000 Congress on Evolutionary Computation, 464-470, 2000.

Koller, D. and Sahami, M. Toward optimal feature selection. In Proc. of the 13th International Conference on Machine Learning, 284-292, Italy, July 1996.

Koza, J.R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, Cambridge, Mass.: MIT Press, 1992.

Kwak, N. and Choi, C.H. Input feature selection for classification problems, IEEE Trans. on Neural Networks, 13 (1), pp. 143-159, 2002.

Lange, S. and Grieser, G. On the power of incremental learning, Theoretical Computer Science, 288 (2), 277-307, 2002.

Lanzi, P.L., Stolzmann, W. and Wilson, S.W. (ed.) Learning Classifier Systems: from Foundations to Applications, Berlin: Springer, 2000.

Lerner, B., Levinstein, M., Rosenberg, B., Guterman, H., Dinstein, L., and Romem, Y. Feature selection and chromosome classification using a multilayer perceptron neural network. In Proc. of IEEE International Conference on Neural Networks, vol. 6, 3540-3545, 1994.

Lesser, V.R. Multiagent systems: an emerging subdiscipline of AI, ACM Computing Surveys, 27 (3), 340-342, 1995.

Liu, J. Autonomous Agents and Multi-Agent Systems: Explorations in Learning, Self-Organization and Adaptive Computation, River Edge, N.J.: World Scientific, 2001.

Liu, D., Chang, T.S., and Zhang, Y. A new learning algorithm for feedforward neural networks. In Proc. of the 2001 IEEE International Symposium on Intelligent Control, México City, 2001.

Lu, B.L. and Ito, M. Task decomposition and module combination based on class relations: a modular neural network for pattern classification. IEEE Trans. on Neural Networks 10 (5), 1244-1256,1999.

Maes, P. Agent that reduce work and information overload, Communication of the ACM, 37 (7), 31-40, 1994.

Masulli, F. and Valentini, G. Parallel non-linear dichotomizers. In Proc. of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, vol. 2, 29-33, 2000.

Melab, N. and Talbi, E. A parallel genetic algorithm for rule mining. In Proc. of 15th International Parallel and Distributed Processing Symposium, 1347-1352, 2001.

Merelo, J.J., Prieto, A., and Moran, F. Optimization of classifiers using genetic algorithms. In Advances in the Evolutionary Synthesis of Intelligent Agents ed. by M. Patel, V. Honavar, and K. Balakrishnan, Cambridge, Mass.: MIT Press, 2001.

Michalewicz, Z. Genetic Algorithms + Data Structures =Eevolution Programs, 3rd ed. New York : Springer-Verlag, 1996.

Michie, D. Problem decomposition and the learning of skills. In Proc. of the European Conference on Machine Learning, Springer-Verlag, 17-31, 1995.

Namatame, A. and Sasaki, T. Competitive evolution in a society of self-interested agents. In Proc. of IEEE World Congress on Computational Intelligence, 1998.

Nwana, H.S. Software agents: an overview, Knowledge Engineering Review, 11 (3), 205-244, 1996.

Nwana, H.S. and Azarmi, N. (ed.) Software Agents and Soft Computing: Towards Enhancing Machine Intelligence, Concepts and Applications, Lecture Notes in Computer Science 1198, Springer 1997.

Osorio, F.S. and Amy, B. INSS: A hybrid system for constructive machine learning, Neurocomputation, 28, 191–205, 1999.

Pal, S.K., De, R.K., and Basak, J. Unsupervised feature evaluation: a neuro-fuzzy approach, IEEE Trans. on Neural Networks, 11 (2), 366-376, 2000.

Polikar, R., Udpa, L., Udpa, S.S., and Honavar, V. Learn++: an incremental learning algorithm for supervised neural networks, IEEE Trans. on System, Man, and Cybernetics, Part C, 31 (4), 497-508, 2001.

Prechelt, L. PROBEN1: A set of neural network benchmark problems and benchmarking rules, Technical Report 21/94, Department of Informatics, University of Karlsruhe, Germany, 1994.

Ratsaby, J. Incremental learning with sample queries, IEEE Trans. on Pattern Analysis and Machine Intelligence, 20 (8), 883-888, 1998.

Rokach, L. and Maimon, O. Improving supervised learning by feature decomposition. In Proc. of Second International Symposium on Foundations of Information and Knowledge Systems, 178-196, 2002.

Schwefel, H.P. and Rudolph, G. Contemporary evolution strategies. In Proc. of Third International Conf. Artificial Life, LNAI 929, Berlin: Springer, 893-907, 1995.

Setiono, R. and Liu, H. Neural network feature selector, IEEE Trans. on Neural Networks, 8 (3), 654-662, 1997.

Setnes, M. and Roubos, H. GA-Fuzzy modeling and classification: complexity and performance, IEEE Trans. on Fuzzy Systems, 8 (5), 509-522, 2000.

Sheth, B. and Maes, P. Evolving agents for personalized information filtering. In Proc. of the Ninth Conference on Artificial Intelligence for Applications, 345-352, 1993.

Smith, R.E., Bonacina, C., Kearney, P.E., and Merlat, W. Embodiment of evolutionary computation in general agents, Evolutionary Computation, 8 (4), 475-493, 2000.

Smith, R.E., Merlat, W., and Kearney, P. Evolutionary adaptation in autonomous agent systems - a paradigm for the emerging enterprise, BT Technology Journal, 17 (4), 157-167, 1999.

Smith, S.F. A learning system based on genetic adaptive algorithms, Ph.D. thesis, University of Pittsburgh, Pittsburgh, 1980.

Srikanth, R., George, R., Warsi, N., Prabhu, D., Petry, F.E., and Buckles, B.P. A variable-length genetic algorithm for clustering and classification, Pattern Recognition Letters, 16, 789-800, 1995.

Stone, P. and Veloso, M. Towards collaborative and adversarial learning: A case study for robotic soccer, International Journal of Human Computer Studies, 48 (1), 83-104, 1998.

Su, L., Guan, S.U., and Yeo, Y.C., Incremental self-growing neural networks with the changing environment, Journal of Intelligent Systems, 11 (1), 43-74, 2001.

Takadama, K., Terano, T, and Shimohara, K. Learning Classifier Systems Meet Multiagent Environments. In Advances in learning classifier systems: third international workshop, ed. by P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Berlin: Springer, 192-212, 2001.

Tsumoto, S. and Tanaka, H. Incremental learning of probabilistic rules from clinical databases based on rough set theory. In Proc. of AMIA annual fall symposium, 198-202, 1997.

Verikas, A. and Bacauskiene, M. Feature selection with neural networks, Pattern Recognition Letters, 23 (11), 1323-1335, 2002.

Vuurpijl, L.G. and Schomaker, L.R.B. Multiple-agent architectures for the classification of handwritten text. In Proc. of the International Workshop on Frontiers in Handwriting Recognition, Taejon, Korea, 335-346, 1998.

Watson, R.A. and Pollack, J.B. Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In Proc. of Sixth International Conference on Parallel Problem Solving from Nature, 425-434, 2000.

Weile, D.S. and Michielssenm, E. The use of domain decomposition genetic algorithms exploiting model reduction for the design of frequency selective surfaces, Computer Methods in Applied Mechanics and Engineering, 18 (6), 439-458, 2000.

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

Wolberg, W.H. and Mangasarian, O.L. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. In Proc. of the National Academy of Sciences, 9193-9196, 1990.

Wooldridge, M. and Jennings, N. R. Agent theories, architectures, and languages. In Intelligent Agents, ed. by Wooldridge and Jennings, Springer Verlag, 1-22, 1995.

Yamauchi, K., Yamaguchi, N., and Ishii, N. Incremental learning methods with retrieving of interfered patterns, IEEE Trans. on Neural Networks, 10 (6), 1351-1365, 1999.

Yang, J. and Honavar, V. Feature subset selection using a genetic algorithm, IEEE Intelligent Systems, 13 (2), 44-49, 1998.

Zadeh, L.A. The Roles of Fuzzy Logic and Soft Computing in the Conception, Design and Deployment of Intelligent Systems. In Software Agents and Soft Computing, Lecture Notes in Computer Science 1198, Springer, 183-190, 1997.

Zhang, J. Selecting typical instances in instance-based learning. In Proc. of the Ninth International Conference on Machine Learning, 470-479, 1992.

Zhu, F.M. and Guan, S.U. Evolving software agents in e-commerce with GP operators and knowledge exchange. In Proc. of the 2001 IEEE Systems, Man, and Cybernetics Conference, Tucson, USA, 3297-3302, 2001. (2001a)

Zhu, F.M. and Guan, S.U. Towards evolution of software agents in electronic commerce. In Proc. of the Congress on Evolutionary Computation 2001, Seoul, Korea, 1303-1308, 2001. (2001b)

Zhu, F.M., Guan, S.U., and Yang, Y. SAFER e-commerce: secure agent fabrication, eolution & roaming for e-commerce. In Internet Commerce and Software Agents: Cases, Technologies and Opportunities, ed. by S.M. Rahman and R.J. Bignall, PA: Idea Group, 190-206, 2000.

Appendix A

Information on Benchmark Data Sets

This appendix provides detailed information of the benchmark data sets used in this thesis. They are wine, glass, cancer, iris, yeast, diabetes, and diabates1 data. The first six data sets are available in the UCI machine learning repository (Blake and Merz, 1998), and the last one is available in the PROBEN1 collection (Prechelt, 1994). They all are real-world problems.

A.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, and 178 instances. The 13 continuous attributes are alcohol, malic acid, ash, alkalinity of ash, magnesium, total phenols, flavanoids, nonflavanoids phenols, proanthocyaninsm color intensity, hue, OD280/OD315 of diluted wines and proline. The class distribution is as follows: 59 instances for class 1, 71 instances for class 2, and 48 instances for class 3.

A.2 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 non-float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation. This data set consists of 214 instances with 9 continuous attributes from 6 classes.

The 9 continuous attributes are refractive index, sodium, magnesium, aluminum, silicon, potassium, calcium, barium and iron. The 6 classes and the distribution of 214 instances are as follows: 70 instances for float processed building windows, 17 instances for float processed vehicle windows, 76 instances for non-float processed building windows, 13 instances for containers, 9 instances for tableware, and 29 instances for headlamps.

A.3 Cancer Data

The cancer problem diagnoses whether a breast cancer is benign or malignant. It has 9 attributes, 2 classes, and 699 instances. All attributes are continuous, and they are clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. Among the 699 instances, 458 instances are benign (65.5%), and 241 are malignant (34.5%).

A.4 Iris Data

The iris data set 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. There are 50 instances for each of the three classes.

A.5 Yeast Data

The yeast problem predicts the protein localization sites in cells. It has 8 attributes, 10 classes, and 1484 instances. The 8 attributes are McGeoch's method for signal sequence recognition (mcg), von Heijne's method for signal sequence recognition (gvh), score of the ALOM membrane spanning region prediction program (alm), score of discriminant analysis of the amino acid content (mit), presence of "HDEL" substring (erl), peroxisomal targeting signal in the C-terminus (pox), score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins (vac), score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins (nuc). The class distribution is as follows:

Class Category	Instances
CYT (cytosolic or cytoskeletal)	463
NUC (nuclear)	429
MIT (mitochondrial)	244
ME3 (membrane protein, no N-terminal signal)	163
ME2 (membrane protein, uncleaved signal)	51
ME1 (membrane protein, cleaved signal)	44
EXC (extracellular)	35
VAC (vacuolar)	30
POX (peroxisomal)	20
ERL (endoplasmic reticulum lumen)	5

A.6 Diabetes and Diabetes1 Data

The diabetes and diabetes1 problems diagnose diabetes of Pima Indians, and they come from different source. Both of them have 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-hour serum insulin, body mass index, diabetes pedigree function, and age. 500 instances are tested negative for diabetes, and 268 instances are positive.

Appendix B

Results of CIGA2 and CIGA4 on the Glass and Yeast Data

This appendix provides the detailed results of CIGA2 and CIGA4 on the glass and yeast data. The summaries of these results have been used in Table 4.5 and Table 4.6 respectively.

Performance comparison on the glass data - CIGA2 and CIGA4

CIGA2	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.3598	0.4729	0.5056	0.6449	0.6710	0.7084	0.6953	0.6991	0.6860
Generations	59.2	49.7	60	54.4	50.6	40	48	47.5	44.6
T. time (s)	45.8	60	88.1	91.6	94.5	81.6	103.1	107.7	106.8
Ending CR	0.4879	0.5299	0.6514	0.7009	0.7206	0.7252	0.7346	0.7402	0.7421
Test CR	0.3505	0.3505	0.3841	0.4308	0.4430	0.4421	0.4355	0.4579	0.4374
CIGA4	Add Att. 1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8	Add Att. 9
Initial CR	0.3467	0.5374	0.5710	0.6607	0.7103	0.6991	0.7449	0.7056	0.7308
Generations	57.7	59.2	59.8	55	57.6	56.9	47.8	52.9	49.4
T. time (s)	47.5	74.8	91.3	95.5	108.2	113.1	105.8	121.9	116.5
Ending CR	0.5477	0.5981	0.6869	0.7299	0.7467	0.7645	0.7692	0.7804	0.7879
Test CR	0.3710	0.4196	0.4439	0.4598	0.4579	0.4505	0.4598	0.4458	0.4458

Summary	**CIGA2**	**CIGA4**
Initial CR	0.3598	0.3467
Generations	454	496.3
T. time (s)	779.2	874.6
Ending CR	0.7421	0.7879
Test CR	0.4374	0.4458

Notes: The experiment setting is the same as that for Table 4.1.

Performance comparison on the yeast data – CIGA2 and CIGA4

CIGA2	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8
Initial CR	0.2667	0.3046	0.3425	0.3786	0.3933	0.4012	0.4023	0.4035
Generations	56.3	58.7	59.5	55.9	47.6	42.9	45.1	43.3
T. time (s)	158.1	271.4	342.5	352.1	331.6	311.7	350.3	357.6
Ending CR	0.3102	0.3217	0.3887	0.4042	0.4061	0.408	0.4092	0.4097
Test CR	0.3082	0.3127	0.3683	0.3814	0.3811	0.3823	0.3809	0.3803
CIGA4	Add Att.1	Add Att. 2	Add Att. 3	Add Att. 4	Add Att. 5	Add Att. 6	Add Att. 7	Add Att. 8
Initial CR	0.2612	0.3061	0.3333	0.3763	0.4111	0.4046	0.4221	0.4129
Generations	58.2	50.1	57.7	53.9	50.3	56.1	50.8	52.4
T. time (s)	175.5	245	319.5	338.8	353.9	410.5	401.4	410.9
Ending CR	0.309	0.3171	0.385	0.4144	0.422	0.427	0.4286	0.4326
Test CR	0.3073	0.3108	0.3677	0.39	0.3911	0.3935	0.3943	0.396

Summary	**CIGA2**	**CIGA4**
Initial CR	0.2667	0.2612
Generations	409.3	429.5
T. time (s)	2475.3	2655.5
Ending CR	0.4097	0.4326
Test CR	0.3803	0.3960

Notes: The experiment setting is the same as that for Table 4.2.

Appendix C

Major Routines of GAs/IGAs and Rule Sets Generated

This appendix lists the major routines of GAs/IGAs, which include main evolution loop, crossover, mutation, fitness evaluation, etc. They are written in Java. A sample rule set generated for the wine data is also listed.

```
public void newGeneration()  //main evolution procedure
   {
     generation++;
     for (int i = 0; i < popSize; i++)
      {
         kids[i] = mate();                    //generate kids
         kidVals[i] = evalValue(kids[i]);
         kidFits[i]=kidVals[i];
      }
     sortKidsByVals();
     if (survivorsPercent > 0)                //replace parents with kids according to survivorsPercent
      {  int n = (popSize * survivorsPercent) / 100;
         for(int i = n; i < popSize; i++)
            if(i > n - 1)
            {  chroms[i] = kids[i - n];
               vals[i] = kidVals[i - n];
               fits[i] = kidFits[i - n];
            }
      } else
      {
        for(int i = 0; i < popSize; i++)
         {  chroms[i] = kids[i];
            vals[i] = kidVals[i];
            fits[i] = kidFits[i];
         }
      }
      processFitness();                       // sort according to fitness
      processValue();                         // check stagnation
      bestVal[generation]=vals[0];
      bestChrom[generation] = chroms[0];
      testingFit[generation]=testing(bestChrom[generation]);
      if ((generation>=generationLimit) || bestVal[generation]>=1.0 || stagnationCounter >=
           stagnationLimit) { exitFlag=true;}        //stopping criteria
   }
```

```
private String mate()                  //generate kids
{
   oldSumFit=selHandler.getSumFitness(fits);        //select parents
   int mom = selHandler.getParent(oldSumFit, popSize, fits);
   int dad = selHandler.getParent(oldSumFit, popSize, fits);
   String kid = crsHandler.crossover(chroms[mom], chroms[dad], crossoverRate1, rossoverRate2);
   kid = mutation(kid,mutationRate1, mutationRate2);   //crossover and mutation
   return kid;
}
 public String mutation(String chrom, double rate1, double rate2)
   {
      StringBuffer sb = new StringBuffer(chrom);
      int size = chrom.length();
      int rule, gene, gNum, genepos;
      char zero=(char)48;   char one=(char)49;
      boolean permitted=false;  double rate;

      for(int i = 0; i < size; i++)
       { if ((i % ruleLen)<GaaAction.ageneNum*GaaAction.ageneSize
            || (i % ruleLen)==ruleLen-1)                    //determine the mutation point
         { permitted=flipm(rate1); rate=rate1;}
         else { permitted=flipm(rate2); rate=rate2;}

      if(permitted)
        {
           rule= i / ruleLen;
           gene=i % ruleLen;
           gNum=gene/ageneSize;
           genepos=gene % ageneSize;
           int p=ageneNum*rule+gNum;
           if (genepos==0 && gene!=ruleLen-1)       //mutate the activeness element
           { if(flipm(0.5*rate))
           if (sb.charAt(i)==zero)   sb.setCharAt(i, one);  else  sb.setCharAt(i, zero);
            }
         if (genepos==1)                    //mutate the minimum value
           {
            double n1 = agene[p].lmin + Math.random() * (agene[p].lmax - agene[p].lmin);
            char kar = agene[p].encodeLValue(n1);
            sb.setCharAt(i, kar);
            }
         if (genepos==2)                    //mutate the maximum value
           {
            double n2 = agene[p].rmin + Math.random() * (agene[p].rmax - agene[p].rmin);
            char kar = agene[p].encodeRValue(n2);
            sb.setCharAt(i, kar);
            }
         if (gene==ruleLen-1)            //mutate class category
           {int n3 = cgene[rule].min + (int) Math.round (Math.random() * (cgene[rule].max -
              cgene[rule].min));
            char kar = cgene[rule].encodeValue(n3);
            sb.setCharAt(i, kar);
           }
          }
        }
      String st = sb.toString();
      return st;
   }
```

```
public String crossover(String chrom1, String chrom2, double crossoverRate1, double
        crossoverRate2)
   {
      String s = "";
      int pos = (int)Math.floor(Math.random() * (double)chrom1.length());
      if ((pos % IGaaAction.ruleLen)<GaaAction.ageneNum*GaaAction.ageneSize
        || (pos % IGaaAction.ruleLen)==IGaaAction.ruleLen-1)     //determine the crossover point
      if(flip(crossoverRate1))
         {
            String s1 = chrom1.substring(0, pos);
            String s2 = chrom2.substring(pos);
            s = s1.concat(s2);
         } else  s = chrom1;

      else
      if(flip(crossoverRate2))
         {
            String s1 = chrom1.substring(0, pos);
            String s2 = chrom2.substring(pos);
            s = s1.concat(s2);
         } else  s = chrom1;
      return s;
   }


   public double evalValue(String chrom)       //compute chromosome fitness
      { double fit=0D;
        int totalCorrect=0;
        boolean correct=false;
        boolean valid=true;

        char[] ruleAnswer=new char[ruleNumber];
        boolean[] ruleValid=new boolean[ruleNumber];
        double actualAnswer;
        char zero=(char)48;
        char one=(char)49;
        char two=(char)50;
        char three=(char)51;

        int testingDataNum=tData.recordNumber/2;

        for (int i=0; i<testingDataNum; i++)
         {
           for (int j=0; j<ruleNumber;j++)
          {
             ruleValid[j]=false;
             valid=false;
             int p1=ageneSize*ageneNum;
             int p2=cgeneSize*cgeneNum;
             int p=(ageneSize*ageneNum+cgeneSize*cgeneNum)*j;

             int m=j*ageneNum;
             for (int g=0; g<ageneNum; g++)          //evaluate rule set with training data
               { if (chrom.charAt(p+g*ageneSize)==one)
```

```
                {    if (agene[m+g].decodeLValue(chrom.charAt(p+g*ageneSize+1))<=
                          agene[m+g].decodeRValue(chrom.charAt(p+g*ageneSize+2))+0.001)
                     if ((agene[m+g].decodeLValue(chrom.charAt(p+g*ageneSize+1))<=
                          tData.data[i][g+numSN]+0.001 &&
                          agene[m+g].decodeRValue(chrom.charAt(p+g*ageneSize+2))+0.001>=
                          tData.data[i][g+numSN]))
                     valid=true;
                     else {valid=false; break;}
                }
               }
            if (valid==true)
              ruleValid[j]=true;
              else ruleValid[j]=false;
            ruleAnswer[j]=chrom.charAt(p+ruleLen-1);          //obtain ruleValid and ruleAnswer
          }
           correct=false;
           actualAnswer=tData.data[i][ageneNum];

           int max=0, OneNum=0, TwoNum=0, ThreeNum=0;     //voting mechanism
           char finalAnswer=(char)52;
           for (int w=0; w<ruleNumber; w++)
            if (ruleValid[w]==true)
              { if (ruleAnswer[w]==one) OneNum++;
                if (ruleAnswer[w]==two) TwoNum++;
                if (ruleAnswer[w]==three) ThreeNum++;
              }
           max=Math.max(OneNum, TwoNum);
           max=Math.max(max,ThreeNum);

           if (max==OneNum && max!=TwoNum && max!=ThreeNum) finalAnswer=one;
           if (max!=OneNum && max==TwoNum && max!=ThreeNum) finalAnswer=two;
           if (max!=OneNum && max!=TwoNum && max==ThreeNum) finalAnswer=three;

           if (finalAnswer==String.valueOf(actualAnswer).charAt(0))
             correct=true; else correct=false;          //determine whether the instance is correctly classified
           if (correct) totalCorrect++;
       }
          fit=(0.0+totalCorrect)/testingDataNum;       //compute fitness
          return fit;
       }

public String encodingChrom(int popNum,GaaAction gaa, HGaaAction hgaa, int type)
                                        //generate initial population with IS1 and IS2
 {   String st = "";
     if (type==1)              //IS1
     {
     int popNo=popNum;
     String gaBestChrom=gaa.getCurrentBestChrom();
     StringBuffer sbga = new StringBuffer(chromLen).append(gaBestChrom);
     int startAgeneNum=gaa.getAgeneNum();
     for (int j=0; j<ruleNumber; j++)
     {
     StringBuffer sb = new StringBuffer(chromLen);
     for(int i = startAgeneNum; i < ageneNum; i++)
                              //integrate old chromosome with randomly created elements
        {
           int m=j*ageneNum+i;
```

```
            int n0;
            if (Math.random()<activeRate)
                    n0=0; else n0=1;
            char kar=(char) (n0+48);
            sb.append(kar);
           double n1 = agene[m].lmin + Math.random() * (agene[m].lmax - agene[m].lmin);
            kar = agene[m].encodeLValue(n1);
            sb.append(kar);
           double n2 = agene[m].rmin + Math.random() * (agene[m].rmax - agene[m].rmin);
            kar = agene[m].encodeRValue(n2);
            sb.append(kar);
          }
     sbga.insert(startAgeneNum*ageneSize+j*ruleLen,sb.toString());
    }
    st=sbga.toString();
    }

  if (type==2)                //IS2
   {
    int popNo=popNum;
    String gaBestChrom=gaa.getCurrentBestChrom();
    String hgaBestChrom=hgaa.getCurrentChrom(popNo);
    StringBuffer sbga = new StringBuffer(chromLen).append(gaBestChrom);
    for (int j=0; j<ruleNumber;j++)            //integrate the old and new chromosomes
  {
    char curClass=gaBestChrom.charAt((j+1)*GaaAction.ruleLen-1);
    int canNum=0;
    StringBuffer canPartChrom=new StringBuffer(gaBestChrom.length());
    for (int i=0; i<HGaaAction.ruleNumber; i++)
      { char hcurClass=hgaBestChrom.charAt((i+1)*HGaaAction.ruleLen-1);
        if (hcurClass==curClass)
        {
         canPartChrom.append(hgaBestChrom.substring(i*HGaaAction.ruleLen,
             (i+1)*HGaaAction.ruleLen-1));
         canNum++;
        }
      }
    if (canNum!=0)          //integration
    { int choseNum=(int)Math.floor(Math.random()*(double)canNum);
       String sb1=canPartChrom.substring(choseNum*(HGaaAction.ruleLen-1),
            (choseNum+1)*(HGaaAction.ruleLen-1));
       int startAgeneNum=gaa.getAgeneNum();
       sbga.insert(startAgeneNum*ageneSize+j*ruleLen,sb1);
   }
    else
    { StringBuffer sb = new StringBuffer(chromLen);
      int startAgeneNum=gaa.getAgeneNum();
      for(int i = startAgeneNum; i < ageneNum; i++)
        {
           int m=j*ageneNum+i;
           int n0;
           if (Math.random()<activeRate)
                    n0=0; else n0=1;
           char kar=(char) (n0+48);
           sb.append(kar);
           double n1 = agene[m].lmin + Math.random() * (agene[m].lmax - agene[m].lmin);
           kar = agene[m].encodeLValue(n1);
```

```
            sb.append(kar);
            double n2 = agene[m].rmin + Math.random() * (agene[m].rmax - agene[m].rmin);
            kar = agene[m].encodeRValue(n2);
            sb.append(kar);
          }
        sbga.insert(startAgeneNum*ageneSize+j*ruleLen,sb.toString());
       }
     }
      st=sbga.toString();
    }
    return st;
  }
```

Sample rule set for the wine data:

No.0 IF (13.30 <= X1 <= 14.83) AND (1.21 <= X3 <= 3.29) AND (66.00 <= X5 <= 149.00) AND (1.62 <= X6 <= 3.86) AND (1.39 <= X7 <= 4.89) AND (0.10 <= X8 <= 0.48) AND (0.78 <= X11 <= 1.52) AND (880.00 <= X13 <= 1510.00) THEN Class=1

No.1 IF (20.80 <= X4 <= 20.82) AND (91.00 <= X5 <= 100.00) AND (0.23 <= X8 <= 0.34) AND (2.45 <= X10 <= 7.19) AND (1.03 <= X11 <= 1.27) AND (2.06 <= X12 <= 3.20) THEN Class=1

No.2 IF (11.37 <= X1 <= 14.40) AND (0.99 <= X9 <= 2.31) AND (2.82 <= X10 <= 12.60) AND (0.67 <= X11 <= 1.15) AND (651.00 <= X13 <= 1564.00) THEN Class=1

No.3 IF (14.53 <= X1 <= 15.36) AND (1.08 <= X2 <= 3.65) AND (0.14 <= X8 <= 0.70) AND (1.96 <= X9 <= 2.59) THEN Class=1

No.4 IF (3.57 <= X2 <= 5.35) AND (2.22 <= X3 <= 3.15) AND (1.07 <= X6 <= 1.16) AND (0.66 <= X7 <= 2.93) AND (0.39 <= X8 <= 0.76) AND (0.32 <= X11 <= 1.18) AND (465.00 <= X13 <= 1340.00) THEN Class=1

No.5 IF (13.50 <= X4 <= 23.50) AND (0.13 <= X8 <= 0.67) AND (0.77 <= X11 <= 1.62) AND (3.67 <= X12 <= 3.94) THEN Class=1

No.6 IF (1.21 <= X3 <= 2.33) AND (0.31 <= X7 <= 1.55) AND (0.80 <= X8 <= 0.75) AND (2.72 <= X9 <= 3.56) AND (4.46 <= X10 <= 4.52) AND (2.45 <= X12 <= 3.33) AND (550.00 <= X13 <= 1190.00) THEN Class=1

No.7 IF (13.61 <= X1 <= 15.44) AND (1.45 <= X7 <= 4.21) AND (4.68 <= X10 <= 11.94) THEN Class=1

No.8 IF (11.14 <= X1 <= 12.80) AND (66.00 <= X5 <= 139.00) AND (2.51 <= X6 <= 3.52) AND (2.15 <= X7 <= 4.78) AND (0.24 <= X8 <= 0.66) AND (6.36 <= X10 <= 11.49) AND (1.40 <= X11 <= 1.69) AND (1480.00 <= X13 <= 1640.00) THEN Class=1

No.9 IF (1.08 <= X2 <= 4.78) AND (0.22 <= X8 <= 0.34) AND (1.33 <= X9 <= 3.04) AND (822.00 <= X13 <= 1490.00) THEN Class=1

No.10 IF (1.09 <= X6 <= 2.24) AND (1.02 <= X11 <= 1.64) AND (1.63 <= X12 <= 2.52) AND (518.00 <= X13 <= 829.00) THEN Class=2

No.11 IF (10.38 <= X1 <= 15.51) AND (1.48 <= X7 <= 2.15) AND (0.23 <= X8 <= 0.57) AND (0.76 <= X11 <= 1.79) AND (1.51 <= X12 <= 3.25) AND (289.00 <= X13 <= 939.00) THEN Class=2

No.12 IF (0.65 <= X2 <= 1.98) AND (1.41 <= X6 <= 3.79) AND (1.40 <= X12 <= 3.33) AND (432.00 <= X13 <= 544.00) THEN Class=2

No.13 IF (12.05 <= X1 <= 13.91) AND (65.00 <= X5 <= 118.00) AND (1.83 <= X6 <= 3.80) AND (1.20 <= X9 <= 3.49) AND (1.11 <= X11 <= 1.70) AND (356.00 <= X13 <= 598.00) THEN Class=2

No.14 IF (0.73 <= X2 <= 4.72) AND (3.07 <= X3 <= 3.97) AND (2.39 <= X6 <= 3.28) AND (0.44 <= X8 <= 0.71) AND (1.98 <= X12 <= 2.65) AND (803.00 <= X13 <= 871.00) THEN Class=2

No.15 IF (10.61 <= X1 <= 13.02) AND (2.04 <= X6 <= 3.80) AND (0.49 <= X9 <= 2.94) AND (1.14 <= X11 <= 1.50) THEN Class=2

No.16 IF (1.08 <= X3 <= 3.75) AND (77.00 <= X5 <= 124.00) AND (0.99 <= X6 <= 3.34) AND (0.97 <= X7 <= 2.49) AND (0.43 <= X8 <= 0.78) AND (1218.00 <= X13 <= 948.00) THEN Class=2

No.17 IF (3.21 <= X2 <= 3.61) AND (3.07 <= X3 <= 3.52) AND (15.20 <= X4 <= 17.50) AND (112.00 <= X5 <= 153.00) AND (3.65 <= X9 <= 0.77) AND (9.82 <= X10 <= 10.55) THEN Class=2

No.18 IF (12.81 <= X1 <= 12.84) AND (1.33 <= X2 <= 5.78) AND (2.10 <= X3 <= 2.15) AND (0.93 <= X6 <= 1.44) AND (2.38 <= X7 <= 2.83) AND (0.72 <= X8 <= 0.73) THEN Class=2

No.19 IF (10.71 <= X1 <= 12.62) AND (1.02 <= X3 <= 2.78) AND (0.68 <= X7 <= 4.93) AND (0.86 <= X11 <= 1.64) AND (379.00 <= X13 <= 1021.00) THEN Class=2

No.20 IF (1.95 <= X3 <= 3.07) AND (20.30 <= X4 <= 30.70) AND (0.40 <= X8 <= 0.70) AND (0.86 <= X9 <= 2.81) AND (7.82 <= X10 <= 10.67) AND (320.00 <= X13 <= 1029.00) THEN Class=3

No.21 IF (11.07 <= X1 <= 14.86) AND (2.10 <= X2 <= 5.03) AND (1.65 <= X3 <= 3.88) AND (24.30 <= X4 <= 29.70) AND (1.13 <= X9 <= 1.84) AND (1.81 <= X12 <= 2.43) AND (745.00 <= X13 <= 1048.00) THEN Class=3

No.22 IF (14.00 <= X4 <= 15.70) AND (0.15 <= X8 <= 0.65) AND (1.36 <= X9 <= 2.65) AND (1.76 <= X10 <= 4.74) AND (0.60 <= X11 <= 1.50) AND (1.22 <= X12 <= 2.32) THEN Class=3

No.23 IF (4.68 <= X2 <= 5.54) AND (70.00 <= X5 <= 140.00) AND (0.40 <= X8 <= 0.79) AND (2.76 <= X10 <= 10.24) AND (0.66 <= X11 <= 1.15) AND (363.00 <= X13 <= 1241.00) THEN Class=3

No.24 IF (12.41 <= X1 <= 15.84) AND (24.20 <= X4 <= 29.50) AND (94.00 <= X5 <= 104.00) AND (2.22 <= X7 <= 3.62) AND (0.11 <= X8 <= 0.57) AND (0.30 <= X9 <= 1.24) AND (9.75 <= X10 <= 12.82) AND (468.00 <= X13 <= 1187.00) THEN Class=3

No.25 IF (11.99 <= X1 <= 14.06) AND (1.70 <= X3 <= 3.72) AND (0.37 <= X11 <= 0.89) THEN Class=3

No.26 IF (13.32 <= X1 <= 15.53) AND (1.05 <= X3 <= 2.28) AND (0.98 <= X7 <= 2.40) AND (3.36 <= X12 <= 3.43) THEN Class=3

No.27 IF (14.09 <= X1 <= 14.51) AND (2.59 <= X2 <= 4.65) AND (70.00 <= X5 <= 115.00) AND (2.77 <= X6 <= 2.88) AND (3.14 <= X7 <= 4.27) AND (0.52 <= X8 <= 0.57) AND (4.71 <= X10 <= 8.45) AND (1.39 <= X11 <= 1.56) AND (1.45 <= X12 <= 2.83) THEN Class=3

No.28 IF (10.03 <= X1 <= 11.77) AND (1.42 <= X2 <= 4.27) AND (2.94 <= X3 <= 3.70) AND (12.90 <= X4 <= 28.10) AND (143.00 <= X5 <= 152.00) AND (1.80 <= X6 <= 2.28) AND (2.70 <= X7 <= 4.84) AND (3.49 <= X10 <= 9.23) AND (0.66 <= X11 <= 1.15) AND (1.38 <= X12 <= 3.65) AND (1208.00 <= X13 <= 1348.00) THEN Class=3

No.29 IF (1.92 <= X2 <= 2.84) AND (1.52 <= X3 <= 2.07) AND (70.00 <= X5 <= 112.00) AND (0.36 <= X8 <= 0.38) THEN Class=3

Author's Publication List

Journal Paper and Book Chapter

1. Guan, S.U. and Zhu, F.M. Class decomposition for GA-based classifier agents – a pitt approach, IEEE Trans. on Systems, Man and Cybernetics, Part B, 34 (1), 381-392, 2004.
2. Guan, S.U. and Zhu, F.M. Incremental learning of collaborative classifier agents with new class acquisition - an incremental genetic algorithm approach, International Journal of Intelligent Systems, 18 (11), 1173-1193, 2003.
3. Guan, S.U. and Zhu, F.M. and Li, P, Modular feature selection using relative importance factors, International Journal of Computational Intelligence and Applications, 4 (1), 1-19, 2004.
4. Guan, S.U. and Zhu, F.M. Agent fabrication and its implementation for agent-based electronic commerce, International Journal of Information Technology & Decision Making, 1 (3), 473-489, 2002.
5. Guan, S. U., Ngoo, C.S., and Zhu, F.M. Handy broker - an intelligent product-brokering agent for m-commerce applications with user preference tracking, Electronic Commerce Research and Applications, Elsevier Science, 1(3-4), 314-330, 2002.
6. Guan, S.U., Zhu, F.M., and Maung, M.T. A factory-based approach to support e-commerce agent fabrication, accepted, Electronic Commerce Research and Applications, Elsevier Science, 2003.
7. Guan, S.U. and Zhu, F.M. Ontology acquisition and exchange of evolutionary product-brokering agents, Journal of Research and Practice in Information Technology, 36 (1), 3-14, February 2004.
8. Guan, S.U. and Zhu, F.M. Incremental learning of classifier agents with genetic algorithms, under second review, IEEE Trans. on Evolutionary Computation, 2002.
9. Zhu, F.M., Guan, S.U., and Yang, Y. SAFER e-commerce: secure agent fabrication, evolution & roaming for e-commerce. In Internet Commerce and

Software Agents: Cases, Technologies and Opportunities, Idea Group, 190-206, 2000.

10. Ng, C.H., Guan, S.U. and Zhu, F.M., Virtual marketplace for agent-based electronic commerce, in Architectural Issues of Web-Enabled Electronic Business, 303-320, Idea Group, PA, 2002.
11. Yeo, W.C., Guan, S.U., Zhu, F.M. An architecture for authentication and authorization of mobile agents in e-commerce, in Architectural Issues of Web-Enabled Electronic Business, 348-361, Idea Group, PA, 2002.
12. Guan, S. U., Ngoo, C.S., and Zhu, F.M. Product brokering and user preference tracking, Wireless Communications and Mobile Commerce, 166-184, IRM press, 2003.

Conference Paper

13. Guan, S.U. and Zhu, F.M. Learning of classifier agents based on incremental genetic algorithms. In Proc. of 2002 IASTED International Conference on AI and Applications, Málaga, Spain, 353-357, 2002.
14. Guan, S.U. and Zhu, F.M. Evolutionary e-commerce agents with ontology exchange. In Proc. of IASTED International Conference on Internet and Multimedia Systems and Applications, 47-51, 2002.
15. Guan, S.U., Zhu, F.M., and Ko, C.C. Agent fabrication and authorization in agent-based electronic commerce. In Proc. of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce, Wollongong, Australia, 528-534, 2000. **Best conference paper award.**
16. Zhu, F.M. and Guan, S.U. Evolving software agents in e-commerce with GP operators and knowledge exchange. In Proc. of the 2001 IEEE Systems, Man, and Cybernetics Conference, Tucson, USA, 3297-3302, 2001.
17. Zhu, F.M. and Guan, S.U. Towards evolution of software agents in electronic commerce. In Proc. of the Congress on Evolutionary Computation 2001, Seoul, Korea, 1303-1308, 2001. **Presentation sponsored by IEEE Neural Network Council.**