2008

An Adaptive Multiobjective Evolutionary Approach To Optimize Artmap Neural Networks

Assem Kaylani
University of Central Florida

An Adaptive Multiobjective Evolutionary Approach to Optimize ARTMAP Neural Networks

by

Assem Kaylani
B.S. University of Jordan, 1998
M.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2008

Major Professors:

Michael Georgiopoulos
Mansooreh Mollaghasemi

Abstract

This dissertation deals with the evolutionary optimization of ART neural network architectures. ART (adaptive resonance theory) was introduced by a Grossberg in 1976. In the last 20 years (1987-2007) a number of ART neural network architectures were introduced into the literature (Fuzzy ARTMAP (1992), Gaussian ARTMAP (1996 and 1997) and Ellipsoidal ARTMAP (2001)). In this dissertation, we focus on the evolutionary optimization of ART neural network architectures with the intent of optimizing the size and the generalization performance of the ART neural network. A number of researchers have focused on the evolutionary optimization of neural networks, but no research has been performed on the evolutionary optimization of ART neural networks, prior to 2006, when Daraiseh has used evolutionary techniques for the optimization of ART structures. This dissertation extends in many ways and expands in different directions the evolution of ART architectures, such as: (a) uses a multi-objective optimization of ART structures, thus providing to the user multiple solutions (ART networks) with varying degrees of merit, instead of a single solution (b) uses GA parameters that are adaptively determined throughout the ART evolution, (c) identifies a proper size of the validation set used to calculate the fitness function needed for ART's evolution, thus speeding up the evolutionary process, (d) produces experimental results that demonstrate the evolved ART's effectiveness (good accuracy and small size) and efficiency (speed) compared with

other competitive ART structures, as well as other classifiers (CART (Classification and Regression Trees) and SVM (Support Vector Machines)). The overall methodology to evolve ART using a multi-objective approach, the chromosome representation of an ART neural network, the genetic operators used in ART's evolution, and the automatic adaptation of some of the GA parameters in ART's evolution could also be applied in the evolution of other exemplar based neural network classifiers such as the probabilistic neural network and the radial basis function neural network.

To my parents

Acknowledgments

I would like to take the opportunity to acknowledge a number of individuals for providing me with their support during my doctoral work. First of all, I would like to express my deepest gratitude to my advisors, Dr. Michael Georgiopoulos and Dr. Mansooreh Mollaghasemi, for their guidance on my dissertation, for their mentorship, and most importantly, for the friendship they have extended over the years.

I am grateful for having an exceptional doctoral committee, and I wish to thank Dr. Avelino Gonzalez and Dr. Ronald DeMara for their time spent reviewing my work and providing valuable feedback.

I would also like to thank my colleagues at Productivity Apex for their support, understanding, and encouragement that they continuously and tirelessly provided during my doctoral work. I am extremely appreciative for their everlasting friendship.

Finally, I would like to extend my sincerest thanks and gratitude to my family and friends for their encouragement and love over the years. Most of all, to my parents, Buthaina and Abdallah, for their unconditional love and inspiration, to my sister, Maysoon, and my brothers, Haydar, Hussam, Hazem and Khaldoon for their unwavering support and encouragement. For all, I extend my love and sincerest gratitude.

This dissertation acknowledges the support by the National Science Foundation grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0203446.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

The Adaptive Resonance Theory (ART) was developed by Grossberg [Gro76]. Some of the ART architectures that have appeared in the literature include Fuzzy ARTMAP (FAM) [CGM92], Ellipsoidal ARTMAP (EAM) [Ana01], and Gaussian ARTMAP (GAM) [Wil96]. All of these ART architectures possess a number of desirable properties, such as they can solve arbitrarily complex classification problems, they converge quickly to a solution (within a few presentations of the list of input/output patterns belonging to the training set), they have the ability to recognize novelty in the input patterns presented to them, they can operate in an on-line fashion (new input patterns can be learned by the ART system without retraining with the old input/output patterns), and they produce answers that can be explained with relative ease.

One of the limitations of these ART architectures that has been repeatedly reported in the literature is the category proliferation problem. This refers to the problem where ART, in the process of solving a classification problem, creates unnecessarily large architectures. This problem is more amplified when the data in the classification problem are noisy, and/or significantly overlapping. Another limitation of these ART architectures is the dependence of their performance on the parameters chosen in the training phase (e.g., vigilance parameter, choice parameter, order of training pattern presentation). Good choices for these parameters are problem dependent, thus requiring experimenta-

tion with various parameter choices (an expensive proposition) in order to obtain the best possibly performing ART networks.

To alleviate these problems, genetic Fuzzy ARTMAP architectures were introduced in [Al 06]. These architectures use a genetic algorithm (GA) (see [Gol89]) to evolve simultaneously the weights, as well as the topology of FAM, EAM or GAM neural networks. It starts with a population of trained ART networks, whose number of nodes in the hidden layer and the values of the interconnection weights converging to these nodes are fully determined (at the beginning of the evolution) by ART's training rules. To this initial population of ART networks, GA operators are applied to modify these trained ART architectures (i.e., number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures.

The following points summarize the main contributions of this dissertation (also illustrated in Figure 1.1):

1. Identify the redundant GA operators used in the algorithm proposed in [Al 06] and propose well-justified GA operators and fitness function for the evolution of ART networks.

2. Develop a scheme for self-adaptation in the genetic algorithm to avoid a grid GA parameter search (a costly proposition), while at the same time improving the GA's convergence speed.

3. Incorporate a theoretically justified evaluation relaxation scheme to improve the GA's convergence speed.

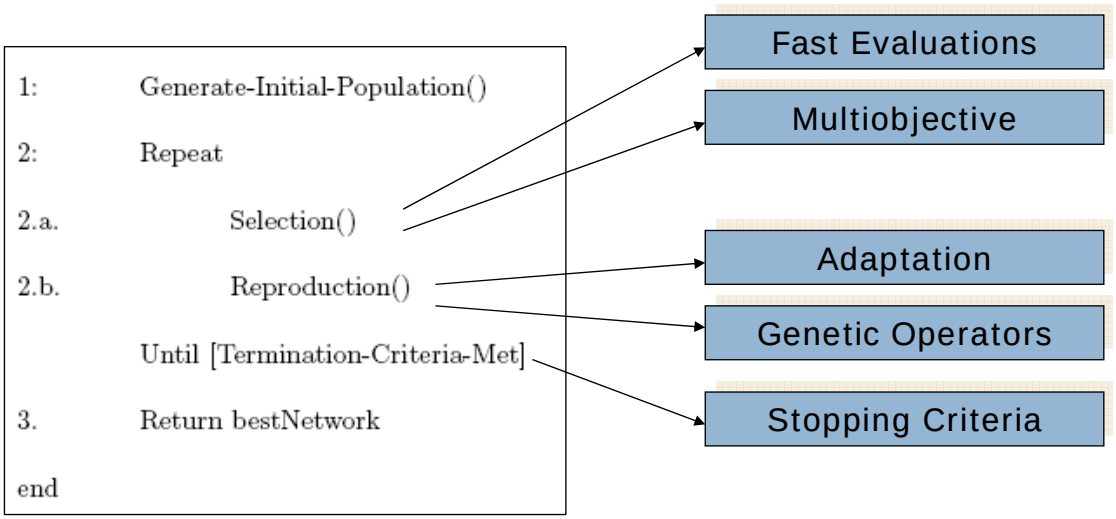


Figure 1.1: Research overview

The areas of research that are addressed in the dissertation to develop the Genetic Algorithm used to optimize ART neural networks. The objectives are: (a) achieve better generalization, (b) smaller size network, (c) reduce overall training time, and (d) eliminate the need to set algorithm parameters that are problem-dependent.

4. Propose a multiobjective evolutionary algorithm to evolve ART architectures, thus providing multiple good ART solutions at the end of the evolutionary process.

The contributions listed above were performed incrementally, and as it will be shown, each was verified and justified properly. The first contribution, which is described in Chapter 3 of the dissertation, resulted in a more careful choosing and justification of the specific genetic operators used in the evolution of the ART architectures introduced in [Al 06]. Although the evolved ART architectures presented in this chapter contributes only a slight improvement in quality of the networks produced when compared to the ones contained in [Al 06], it significantly simplifies it and provides a more justifiable design. In addition, the proposed improvement resulted in significant reduction in training time. For example, on a sample dataset (g4c-25), the training time was on average reduced

from 77 seconds to 12 seconds (or 84% reduction) without any loss in the solution's quality. The fitness function proposed in this chapter allows the user to easily control the importance of accuracy and complexity in the network produced; a capability not available in the architecture introduced in [Al 06]. The aforementioned improvements resulted in allowing us to apply these evolved ART architectures to a wider (and more complicated) range of problems. The evolved ART architecture of Chapter 3 serves as the baseline evolved ART architecture against which additional contributions in this dissertation, outlined in Chapters 4 through 6, are judged against.

The second contribution in this dissertation is described in detail in Chapter 4. This chapter proposes an adaptive genetic algorithm to evolve the ART architectures, by using a metric referred to as the confidence factor. This metric is used to adaptively and efficiently drive the progress ART's evolution. The results presented at the end of this chapter show the benefits of the suggested approach compared to the approach of Chapter 3. For example, for a sample dataset (g4c-25) the total run time for evolving ART was reduced from 12 seconds to 5 seconds (or 58%). Another major advantage of the proposed architecture is the elimination for the need of pre-specifying the GA parameters used in the evolution of ART. The proposed adaptive mechanism allows these parameters to be chosen properly for every dataset over the process of evolution.

The third contribution described in detail in Chapter 5 of the dissertation introduces another refinement, with focus on improving the convergence speed. This refinement is focused on reducing the size of the validation set needed to calculate the fitness of every GA solution. The refinement is applied to the adaptive approach of Chapter 4 of

the dissertation. It is shown experimentally that the proposed refinement has an added value in terms of significantly improving the GA convergence speed without degrading the quality of the solutions attained by the GA. For example for a sample dataset (g4c-25) the training time was reduced from 5 seconds to 2.3 seconds (or 55%). It is worth mentioning that the total improvements, presented in Chapters 3 to 5, resulted in reduction of training time from 77 to 2.3 seconds (or 97%) for the g4c-25 dataset (similar improvements have been observed for other datasets, as well). Furthermore, it is worth emphasizing that this refinement can be applied to the neural evolution of other NN architectures, other than the ART architectures, which is the focus of this dissertation.

The final contribution of this dissertation, described in Chapter 6, proposes a multiobjective approach to evolve ART architectures (the two objectives of interest are network generalization performance and network size). As shown in the chapter, this approach allows the GA to return multiple solutions (ART networks), each one of which has varying degrees of merit (i.e., generalization performance or network size). This is advantageous in practical applications because it provides the users with the flexibility to choose the ART network that best solves their application problem. Furthermore, it is shown in this chapter, that the multiobjective evolutionary approach is necessary to find networks (solutions) that cannot be found using the single objective approach of the previous chapters and the one introduced in [Al 06]. For example, on a sample dataset (Pendigits), the best accuracy achieved by the architecture introduced in [Al 06] was 91%, where as shown it will be shown, the proposed architecture is able to achieve over 98% accuracy, while reducing the training time from 1140 seconds to 72 seconds.

The final product of this dissertation is compared to other state-of-the ART classifiers; CART (Classification and Regression Trees) and SVM (Support Vector Machines). It is shown that the proposed approach compares very favorably in terms of generalization performance, classifier complexity and training time. Furthermore, the proposed approach produces several alternative networks of varying accuracy and size at no additional cost.

The next chapter (Chapter 2) provides a comprehensive literature review of genetic algorithms. It also provides an overview of their applications to evolve neural networks. As mentioned earlier, Chapters 3 through 6 describe the different contributions proposed in this dissertation. Finally, Chapter 7 provides a detailed description of the final product of this dissertation and compares it with two other classifiers: CART and SVM. It also provides direction for future research.

CHAPTER 2
GENETIC ALGORITHMS AND GENETIC NEURAL NETWORKS

Evolutionary algorithms (EAs) are population-based search algorithms that use mechanisms inspired by natural evolution. EAs evaluate candidate solutions to a given problem in an iterative fashion. In each iteration a population of these solutions is evaluated and then evolutionary operators are repeatedly applied to the population with the objective of evolving this population to some desired optima. 'Survival of the fittest' concept ensures that the overall search direction will eventually lead to a good solution. Individuals in the population exchange features using biologically-inspired operators such as selection, mutation and cross-breeding.

Compared to many other optimization algorithms, EA's are known for their resistance to getting trapped in local optima. They do not depend on gradient information and thus are quite suitable for problems where such information is unavailable or very costly to obtain or estimate. More importantly, genetic algorithms can handle problems with complicated and noisy solution surfaces and problems where the solution space is large and not well understood.

Evolutionary algorithms include Genetic Algorithms (GAs), Evolutionary Programming (EP) and Evolutionary Strategies (ES). GAs evolve a population of individuals each representing a possible solution to a given problem. Each individual solution is

evaluated using a fitness function. The highly fit individuals are given more opportunity to survive and breed. Less fit individuals slowly die out and are replaced by individuals that result from cross-breeding, mainly, highly fit individuals. The offspring individuals share features taken from each parent. The offspring individuals have higher portion of the characteristics possessed by the good parents in the previous generation. This way, over many generations the good characteristics are mixed and exchanged and the most promising areas of the solution space are explored. Evolutionary strategies on the other hand use real-vectors as coding representation, and primarily mutation and selection as search operators. Mutation typically relies on Gaussian perturbation where the step size or mutation strength, represented by the standard deviation of the Gaussian distribution, is often governed by self-adaptation.

Genetic algorithm are known to be robust search and optimization techniques because of their ability to locate the global optimum in a *multimodal* landscape. A number of such multimodal problems exist and the focus in this work is on optimization of neural network structure and weights. Compared to gradient decent-based methods, GAs are capable of finding the global optima.

In this effort the focus is on the use of a GA to evolve ARTMAP architectures. Also an ES-inspired mutation operator is used to bring about random change in the weights. This chapter introduces Genetic Algorithms, highlighting the basic genetic operators and concepts to be used later in the development of Genetic ARTMAP Architectures. Comprehensive description of genetic algorithms can be found in [Gol89] and [Mit96].

Also, this chapter presents a brief review of the literature concerning the application of GAs to various types of neural network architectures.

2.1 Introduction

John Holland [Hol75] is known to have developed genetic algorithms (GA) during the 1960s and 1970s. Genetic algorithms uses an analogy inspired from biological evolution. A solution is represented as an individual. A set of candidate solutions are represented by a population of individuals.

2.2 The Basic Genetic Algorithm

Genetic algorithm are a class of population-based Evolutionary search algorithms. GA's are typically used to solve optimization problems. Genetic algorithms start with a population of candidate solutions, or individuals, and evolve this population by repeatedly applying genetic operators to this population. After each evolutionary step a new generation of individuals is produced. The genetic operators are designed in such a way to lead the search to the desired optimal solution. The search does not typically rely on local criteria (as opposed to other optimization algorithms, such as gradient-based algorithms). Therefore, the GA is allowed to visit any solution in the solution space, making GA a global optimization algorithm. A pseudo-code for the basic GA is shown below.

```
Generate-Initial-Population();
repeat
    Selection();
    Reproduction();
until max number of generations reached ;
return Best Solution;
```

Figure 2.1: Pseudo-code of a basic Genetic Algorithm

2.3 Initialization

The genetic evolution process starts by creating an initial population of solutions. The solutions are often initialized randomly, however, it is known that starting with good initial solutions is beneficial for the performance of the GA. Once solutions are created, they are converted to chromosomes using some encoding scheme.

2.3.1 Encoding Schemes

There are several different types of encoding of a chromosome, where the choice of encoding often depends on the structure and the precision requirements of the optimization problem. The most commonly used chromosome encoding is binary and real number encoding.

2.3.2 Population Size

The population size is one of the most important issues to consider when creating the initial population. Its importance is due to the tradeoff between the time to find the

solution and accuracy of the solution. A small population runs the risk of seriously under-covering the solution space and increases the chance of premature convergence to a local optimum, whereas a larger population has a better population diversity and is therefore less prone to premature convergence [Mit96]. However, a large population allows the exploration of fewer generations per unit of computational effort and, if the available computational effort is limited, it may preclude convergence at all.

2.4 Selection

This operator selects individuals in the current generation to be used for constructing the next generation. This operator in GAs is analogous to the process of natural selection in biology. Fitter individuals are more capable of survival and breeding. In GAs, selection allows the search to move towards better solutions as long as the fitness is measured in terms of the objective function of the problem at hand. Therefore, the first step in selection is evaluation of fitness.

After evaluating all the chromosomes, the selection operation determines the ones that will be selected for the reproduction of the next generation of chromosomes. The set of selected individuals is usually referred to as the mating pool. In general, the purpose of the selection operation is to emphasize fit individuals in the population by giving them more chance to breed than less fit individuals. Therefore, selection preserves characteristics of fit individuals to be used to construct new offspring, and also removes bad individuals so that the overall population fitness improves over successive generations.

Selection should be appropriately balanced in favoring more fit individuals over less fit individuals. This balance is referred to as the *selection pressure*. The convergence rate of a GA is highly dependent on the selection pressure; a strong selection pressure highly favors fit individuals and results in faster convergence. However, if the selection pressure is too strong, few highly fit individuals will take over the population, reducing the *diversity* needed for exploring different regions of the solution space and therefore might be resulting in premature convergence of the GA. On the other hand, if the selection pressure is too weak, the GA will require more iterations before finding the optimal solution. A balance of the two is required in order to preserve population diversity and help avoid premature convergence. The following subsections describe the most common selection schemes used in GAs, with a brief analysis of the selection pressure imposed by the selection scheme.

2.4.1 Fitness proportionate selection

This scheme assigns probability of selection for every individual that is proportional to its fitness, as shown in Equation (2.1). Therefore, more fit individuals are more likely to be selected for reproduction (see Figure 2.2). With fitness proportionate selection there is a chance that some higher fitness solutions will be eliminated and some weaker solutions may survive the selection process. This is an advantage, as it helps preserve the diversity of the population and avoid the GA from performing hill-climbing search that is likely to end up at a local optimum.

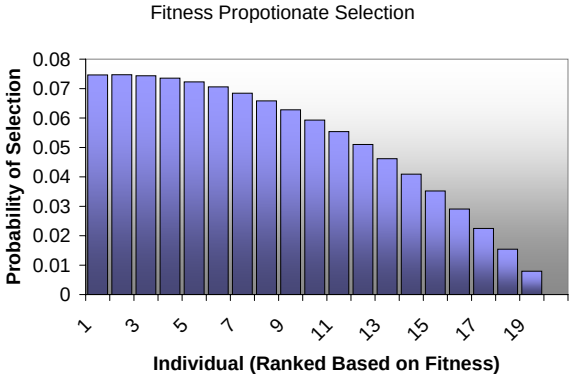


Figure 2.2: Selective probability distribution of fitness proportionate selection

$$p_{sel}(i) = \frac{fit(i)}{\sum_{i=1}^{L}(fit(i))} \tag{2.1}$$

The most common implementation of the fitness proportionate selection scheme, known as roulette-wheel selection, uses the analogy to a roulette wheel in which each candidate solution represents a slice on the wheel. The size of the slices are proportional to the individual's fitness. For each individual in the mating pool, the wheel is spun and individual under the wheel marker is selected. This method has the disadvantage of, for small population size, the actual number of times an individual is selected is far from the expected value (which is proportional to its fitness). To help avoid this problem, a modified scheme referred to as Stochastic Universal Sampling is used where, the wheel is spun once and an individual is selected if it is under one of the equally spaced markers. The number of markers is set as the the number of parents.

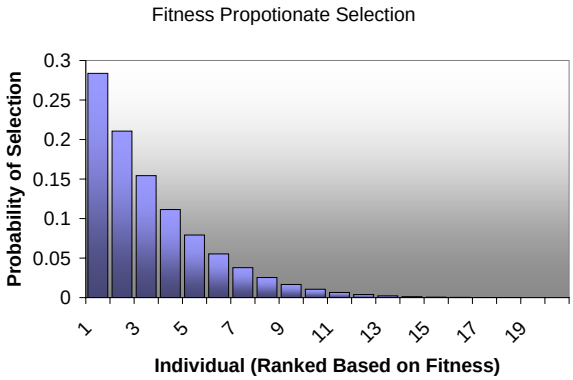


Figure 2.3: Selective probability distribution of fitness proportionate selection for high-variance fitness values

One problem that is often associated the proportionate selection scheme is that, when the variance of the population fitness values is large, as it is usually the case at the beginning of the search, the highly fit individuals tend to dominate the population (see Figure 2.3). This quickly reduces diversity and may hinder exploring new regions in the solution space, therefore leading to premature convergence. Another problem with this selection scheme is the stagnation of the fitness values. This could happen towards the end of the run when all chromosomes tend to have relatively high but similar fitness values, so there is a relatively small difference between the selection probabilities.

Other variation of fitness proportionate selection include sigma scaling in which the selection pressure is proportional to the scaled fitness values. The scaling aims at keeping the selection pressure relatively constant to avoid the problems mentioned above. Another variation is the Boltzmann selection where a simulated annealing-inspired tem-

perature schedule is used to apply weak selection pressure at the beginning of the search and stronger selection pressure as evolution progresses.

2.4.2 Rank Selection

The rank selection scheme is an ordinal-based selection scheme in which the selection is based on the relative rank of the fitness of the individual rather than its actual fitness value. Ordinal schemes alleviate the problems mentioned above with fitness proportionate selection where the highly fit individuals and their successor may take over the population when the fitness variance is high. Also, this scheme maintains the selection pressure when the fitness variance is low.

One way to implement rank selection is using a linear ranking function. Linear ranking selects each individual in the population with a probability linearly proportional to the rank of the individual.

$$p_{sel}(i) = \frac{max - (max - min)(\frac{i-1}{\lambda-1})}{\lambda} \tag{2.2}$$

where λ is the population size, max is the number of desired copies for the best chromosome and min is the number of desired copies of the worst chromosome, respectively [Bak85]. Figure 2.4 illustrate the relative probability of selection of individuals in the population.

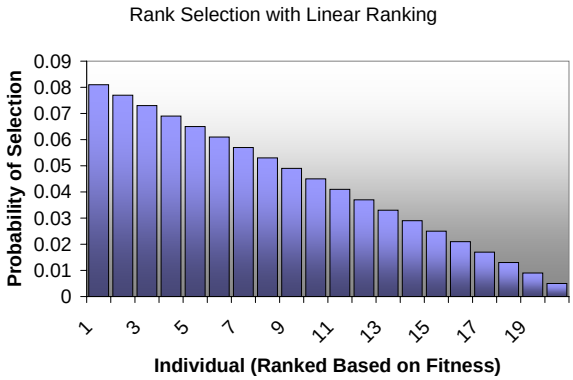


Figure 2.4: Rank selection by means of a linear ranking function

2.4.3 Tournament Selection

Tournament selection runs a "tournament" among a few randomly chosen individuals in the population. The individuals are then ranked and one individual is selected based on a given probability distribution. That is, with probability p_1 choose the best individual in the tournament, with probability p_2 choose the second best, and so on. A variation to this method, referred to as deterministic tournament selection, always selects the best individual in any tournament. The number of individuals in the tournament, referred to as the tournament size, can be used to control the selection pressure. A tournament size of 1 is equivalent to random selection where no selection pressure is applied and all individual have equal chance to breed. Tournament selection is ordinal-based selection scheme and has several benefits such as that it is efficient to code and allows the selection pressure to be easily adjusted. However, the selection pressure become very strong quickly as the tournament size increases. This is illustrated in figures 2.5 and 2.6.

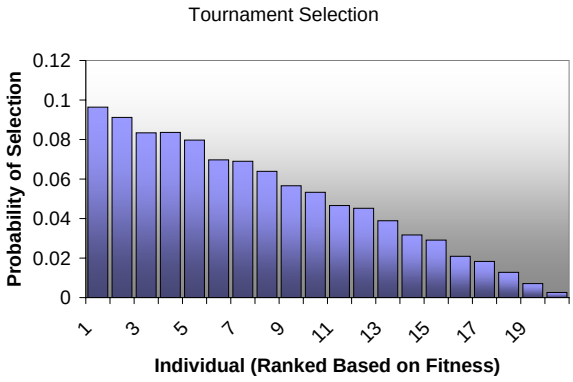


Figure 2.5: Tournament selection using tournament size = 2

The selection probability of i-th individual, ranked in ascending order of fitness, can be calculated as follows:

$$p_{sel}(i) = \frac{i^q - (i-1)^q}{\lambda^q} \tag{2.3}$$

where λ is the population size and q is the tournament size.

2.4.4 Elitism

Elitism is a mechanism that can be used in conjunction with any of the previously mentioned selection methods. Elitism refers to carrying over good performing chromosomes from the old generation to new generation without change. This prevents the possibility of losing good chromosomes from one generation to the next. Elitism guarantees monotonic improvement of the search and that the search will eventually return the best solution

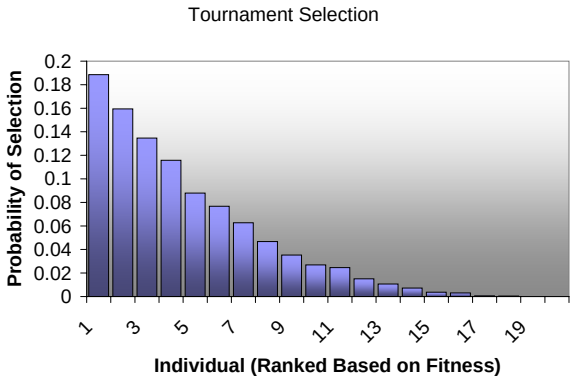


Figure 2.6: Tournament selection using tournament size = 4

found. Research has shown that elitism significantly improves the performance of the GA (see [Mit96]). Using elitism has also been shown to guarantee a global convergence under some assumptions since the best chromosome in the population is monotonically improved. The assumption is that any chromosome must be reachable from any other chromosome by means of mutation and recombination.

2.4.5 Selection Intensity Analysis

A number of authors tried to analyze the convergence properties in terms of selection scheme used (see [GD91]). In [MS93] the authors define the selection intensity as a quantitative measure to the selection pressure. Selection intensity is then used to account for the convergence properties of genetic algorithms. In [TG94] convergence models were developed for proportionate selection, binary tournament selection, truncation selection and elitist recombination. In [MG95] the author extend the results to tournament sizes

larger than two. Selection intensity and the convergence models proposed are summarized in the following paragraphs.

The selection schemes discussed previously, result in a probability of selection for each chromosome. The probability of selection determines the expected number of copies a given chromosome will have in the mating pool. For example if the algorithm is selecting N chromosomes to be used as parents for the next generation of chromosomes, then the expected number of of copies for chromosome i is given by Np_{sel}. For example, for proportionate selection, the expected number of copies is given by equation (2.4):

$$E(x, t+1) = \frac{fit(x)}{\mu(P, t)} \tag{2.4}$$

where $\mu(P, t)$ is the average fitness of solutions in the population $P(t)$ at generation t. The selection operator in genetic algorithms is the mechanism that mimics natural selection in biology as it favors better solutions by giving them more chance to breed. Therefore, better solutions should have larger expected value of the number of copies to be used to produce the next generation. The selection pressure refers to the degree to which better solutions are favored. Selection pressure is a significant factor that determines the convergence rate of the genetic algorithm. If the selection pressure is too low the convergence rate will be slow and the GA will unnecessarily take longer to find the optimal solutions. If the selection pressure is too high there is an increased chance that the GA prematurely converging to a suboptimal solution.

To quantify selection pressure, in [MS93] the authors define the *selection intensity*, I, of a genetic algorithm as the expected value of the average fitness of a population after

selection is performed on a population whose fitness is distributed according to the unit normal distribution $N(0,1)$. Therefore, if the population at generation t is distributed according to the normal distribution $N(\mu_t, \sigma_t)$, then after selection, the expected mean fitness can be determined by equation (2.5):

$$\mu_{t+1} = \mu_t + I\sigma_t \tag{2.5}$$

This model assumes that the fitness is normally distributed. According to [MG96], the normality assumption is reasonable as recombination and mutation operators have a normalizing effect on the population. In [MS93] the author derive the selection intensity for proportionate selection as σ_t/μ_t, where μ_t is the mean and σ_t is the standard deviation of the population at generation t. The selection intensity of linear ranking is given by [BT95] as $(max - 1)\frac{1}{\sqrt{\pi}}$ where max denotes the number of desired copies of the best individual.

In [MG95] the authors present the results of analysis for estimating the selection intensity for tournament selection using order statistics. The selection intensity of tournament selection of tournament size q is the expected value of the maximal order statistic for a sample of size q drawn from the normal distribution, denoted by $\mu_{q:q}$. (The maximal order statistic, denoted as $\mu_{i:j}$, represents the expected value of the i^{th} biggest sample out of a sample of size j drawn from the unit normal distribution). Table 2.1 lists the selection intensity for tournament sizes 2 to 5, as provided by [MG95].

Table 2.1: Selection Intensity for tournament selection.

q	$\mu_{q:q}$	value of $\mu_{q:q}$
2	$\frac{1}{\sqrt{\pi}}$	0.5642
3	$\frac{3}{2\sqrt{\pi}}$	0.8463
4	$\frac{6}{\pi\sqrt{\pi}} tan^{-1}(\sqrt{2})$	1.0294
5	$\frac{5}{4\sqrt{\pi}} + \frac{15}{2\pi\sqrt{\pi}} sin^{-1}(\frac{1}{3})$	1.163

2.5 Reproduction

A new generation is formed in every iteration of the genetic algorithm. The new generation is formed by applying *genetic operators* to the selected mating pool of the current generation. The classical genetic operators used for reproduction are crossover (also referred to as recombination), and mutation.

2.5.1 Crossover

Crossover is an operation used to explore new regions in the solution space. This is done by combining chromosomes (parents) from the mating pool to form new chromosomes (offsprings). Recombination through crossover can be severely disruptive to the candidate solutions and for that reason many researchers avoid the use of crossover altogether for some problems. For example [ASP94] suggests an ES inspired algorithm to evolve MLP NNs. But in other problems, crossover can be a very powerful operator. For example, in this research we found that crossover allows us to form ARTMAP networks that are difficult to form otherwise. Therefore crossover allowed us to reach parts of the solution space that might not be reachable otherwise.

2.5.2 Mutation

Mutation is a method of creating new offspring by modifying the parent. Binary mutation is quite simple; it is done by flipping a bit from 0 to 1, or the other way around, according to a specific probability. Floating point mutation can be accomplished in different ways. For example, one way is to add a randomly selected number from a Gaussian distribution. The mutation rate should be appropriately controlled for given problem. It is important to mention that low mutation rate results in less exploration, while high mutation rate could be disruptive. In Chapter 4, innovative ways of controlling the mutation rate are studied in details.

2.6 Termination

The termination criteria determines when to stop the iterative process of a GA and return the optimal solution. There are many termination criteria that can be used in combination. The choice depends on the problem at hand. The basic termination criterion is when a user-specified computational budget is consumed. This budget can be measured in terms of the number of iterations or CPU time. This criterion of termination does not guarantee that a global optimum is found; it only returns the best solution found for the given budget. More appropriate criteria might be to stop after convergence is achieved, where convergence can be defined in different ways. The most frequently used convergence criterion is when the solution quality has reached a plateau such that

no improvement by more than a specific amount is observed over a specific number of iterations.

2.7 Evolving Neural Networks: Literature Review

Genetic algorithms have been extensively used to evolve artificial neural networks. GA's are capable of finding the global optima rather than the local optima as is the case with gradient decent procedures. Also, GA's are not as sensitive to the initialization of weights. Moreover the fitness function can include a measure of complexity in addition to a measure of error.

The literature is rich with articles proposing applying evolutionary optimization algorithms to train neural networks [Fer05, FW02, MW02, PHU05, WC96]. The majority of the focus, though, was on MLP neural networks [Yao99]. However, a number of authors proposed using evolutionary optimization algorithms with other neural network models such as RBF neural networks as in [FW02, WC96]. In [Yao99] a comprehensive literature review was conducted to summarize the prior efforts that aimed at combining evolutionary optimization algorithms with neural networks. The author divides these efforts into three main classes as follows:

1. Evolution of connection weights.

2. Evolution of architectures.

3. Evolution of learning rules.

In evolving connection weights, researchers suggest methods that replace the typically-used gradient decent procedure (see for example [SDJ98]). Some authors such as [MW02] have suggested the use of hybrid procedure combining a GA and gradient decent. The GA is better at finding the global minimum approximate location, while the gradient decent procedure is more efficient at fine-tuning the search.

Evolving neural network architectures involves the task of determining the number of hidden nodes and layers, interconnections between nodes, and activation functions (see for example [MH93]). This task is normally accomplished by relying on expert opinion or tedious trial-and-error experimentation. Genetic Algorithms can be used to find the optimal number of hidden layers, nodes in each layer, connections between these nodes and the activation function to use.

Evolving learning rules include choosing the learning algorithm and its parameters using GA's (see for example [Han92]). This is appealing since it is hard to design optimal learning rules that will work with different types of architectures when there is no prior information about these architectures. The simpler form of evolving training rules is evolving algorithmic parameters such as the learning rate, while more complex tasks would include evolving weight update rules.

Some authors like [Fer05] used a GA to evolve both the architecture and the learning rules, while using back propagation algorithm (BP) for weight learning. Other authors like [PHU05] used a GA for evolving the architecture and the weights. This approach has the advantage of eliminating the dependence on the weight initialization problem associated with back propagation.

Furthermore, GAs may also be used to select the features that are input to the neural network. Since the pioneering work by Siedlecki and Sklansky [SS89], genetic algorithms have been used for many selection problems using neural networks [T 95, YH98], and other classifiers, such as decision trees [BDH96], k-nearest neighbors [J 91, PGC93], and Naïve Bayes classifiers [ILE99, Can02].

Application of GA's requires the design of encoding schemes to represent the aspects being evolved as chromosomes. A great deal of attention has been directed to this issue because of the many problems that appeared to be related to flaws or deficiencies in the encoding schemes. One of the problems is referred to as the permutation problem or many-to-one problem (see [PHU05]). This refers to the existence of multiple representations that translate to the same network, which makes evolution process inefficient and reduces population diversity and may lead to premature convergence.

When evolving connection weights, the encoding schemes can be either binary or real-valued. In binary representation, the researcher has to decide on the ranges and precision of the real values being represented. A tradeoff between representation precision and the length of chromosome often has to be made. For real number representation the researcher has to decide on how to implement the mutation operator. Most researchers use Gaussian mutation with real number representation.

For evolving architectures, [Yao99] identifies two encoding schemes: direct encoding and indirect encoding. The direct encoding specifies the full connectivity of the network using a binary connectivity matrix. This might result in very long chromosomes. Also this approach is vulnerable to the permutation problem as the connectivity matrix may

represent two functionally equivalent networks. The indirect encoding on the other hand only stores parameters such as the number of hidden layers and the number of nodes in each hidden layer. This results in much shorter chromosome than the direct encoding scheme. The problem with such representation, however, is that it is capable of searching only a subset of the feasible architecture space. As will be shown later, in evolving ART architectures, a 2-level representation scheme is used. The first level contains the components (hidden units) of the network and level 2 contains the weights represented using real-valued encoding.

Evolution of weights often uses Gaussian perturbation in the mutation operator to bring about changes in the weights [Fog93]. When relying on mutation to bring about change in the network structure and/or weights, some researchers needed to incorporate an adaptive mechanism to control the severity of mutations [ASP94, YL98]. This was done by adopting a Simulated Annealing-based strategy, in which mutations are allowed to be aggressive at the beginning of the evolutionary process while allowing only milder mutations as the process progresses towards the optimum. In [ASP94] the author defines a network temperature that is inversely proportional its fitness. The temperature parameter is used to control severity of mutation for both weight and structure. This way, networks with a high temperature are mutated severely, and those with a low temperature are mutated only slightly allowing a broad search initially, and a narrower search as a network approaches a solution. For weight mutation, the temperature is used to determine the standard deviation of the Gaussian distribution used to bring about change in

the weights. For structural mutation, the temperature is used to scale the range of the number of modifications to be made to the structure.

Evolution of architecture often uses mutation of the architectural structure that includes addition and deletion of connections and nodes. In [ASP94], the author recommends minimally changing the behavior of the network when devising structural mutation. For example when adding a new node, its weights are initialized to zero to keep the network functionally unchanged. In this work, structural additions are avoided as the probability of adding a viable component is very low.

Many authors [ASP94, PHU05, Yao99] point out to problems associated with using crossover when evolving structure. The crossover has a destructive effect as it may combine parents that result in nonviable offspring solutions. The probability of producing offspring solutions with worse fitness than the parents is relatively high when crossover is used. This significantly reduces the effectiveness of the EA. To combat this problem, some authors [ASP94, PHU05] eliminate the use of crossover and rely only on mutation. In [ASP94] the author recommends EA techniques that rely solely on mutation as the reproductive operator for searching over architecture space when there are no solid rules to guide recombination by crossover.

The 2-level encoding scheme adopted in this work to evolve ART architecture has the advantage of being able to apply viable crossover operations at level 1 of the representation. On the other hand, adaptive mutations are applied at both levels, as will be shown later. Mutations at level 1 cause structural modifications (referred to as Pruning), while mutations applied at level 2 cause weight modifications.

The work that utilizes GAs and ART neural network architectures is rather limited. In [Al 06] application of a genetic algorithm to ART architectures was introduced for the first time. In [LLL03], a GA algorithm was employed to appropriately weigh attributes of input patterns before they were fed into the input layer of Fuzzy ARTMAP. The results reported reveal that this attribute weighting was beneficial because it produced a trained ART architecture of improved generalization. In [BV97], a Fuzzy ART neural network was employed as a GA fitness function evaluator. This work aims at developing an evolutionary algorithm to evolve ART architectures, that will overcome the shortcomings and improve upon the work that was proposed in [Al 06]. In Chapter 3 the genetically engineered ART architectures proposed in [Al 06] are studied carefully and a number of improvements are proposed.

CHAPTER 3
IMPROVING THE GENETIC ARTMAP ARCHITECTURES

As mentioned earlier the focus of this effort is the evolution of ART architectures with the objective of improving the generalization performance and alleviating the category proliferation problem in ART. With the same objectives, researchers introduced a number of ART variations. For example, in [AG02] the authors introduce a new family of ART networks that rely on the concept of supervision. In [Al 06] the authors use a genetic algorithm to evolve ART architectures. In this chapter improved, compared to [Al 06], evolved ART architectures are introduced. Through extensive experimentation it will be shown that these evolved ARTMAP architectures exhibit good generalization and are of small size, while consuming reasonable computational effort to produce an optimal or a sub-optimal network. The product introduced in this chapter will serve as a baseline for further refinements to the evolutionary ART architectures that will be introduced and analyzed in future chapters of this dissertation.

In Section 3.1 the ART architectures are reviewed, to a level that is necessary to understand the evolved ART architectures. Section 3.1 is not intended to be a complete reference for the ART architecture, but a refresher for readers with prior ART knowledge. This section also provides a number of references for readers who do not have that prior knowledge in ART. In Section 3.2 we describe the evolved ART architectures introduced in [Al 06] and highlight a number of improvements that are proposed. In Section 3.3 a

sound technique is introduced and used to find good default parameter values for the evolved ART algorithms. Section 3.4 compares the resulting architectures to other ART architectures that were introduced into the literature to address the category proliferation problem. The comparison is also extended to other non-ART classifiers to show that architectures introduced in this chapter are competitive beyond the ART family.

3.1 ART Preliminaries

Grossberg [Gro76] introduced the foundation of ART in 1976. Later, based on that work, ART1 was developed to perform clustering (self-organizing) of binary patterns [CG87b]. ART1 was then extended to ART2 to handle real-valued patterns [CG87a]. In 1991 Carpenter and Grossberg introduced ARTMAP [CGR91a], which was capable of performing classification of binary patterns. They then simplified ART2 architecture and introduced an improved version called Fuzzy ART [CGR91b]. Furthermore, in 1992 Carpenter and Grossberg extended ARTMAP to Fuzzy ARTMAP, which is capable of classifying real-valued input patterns [CGM92].

Since the introduction of Fuzzy ARTMAP, other ART architectures have been introduced into the literature and the focus of this effort is on Fuzzy ARTMAP and two other ART architectures: Ellipsoidal ARTMAP (see [Ana01]) and Gaussian ARTMAP (see [Wil96]). The objective in this effort is to illustrate how genetically engineered ART architectures can be designed from a population of Fuzzy ARTMAPs, or Ellipsoidal ARTMAPs, or Gaussian ARTMAPs. It is assumed that the reader is familiar with all

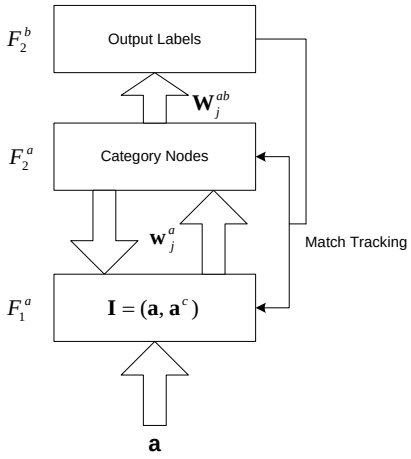


Figure 3.1: The block diagram of a FAM Architecture

these ART architectures. This section only describes the specifics of ART architectures that are needed to understand the genetically engineered ART structures. For simplicity all these ART architectures are referred to as ART and their specific name is used (FAM, or EAM or GAM) only when there is a need to discriminate one from the other.

3.1.1 Operation of ART

The block diagram of an ART architecture is shown in Figure 3.1 (for FAM) and Figure 3.2 (for EAM and GAM). Notice that this block diagram is simpler than the block diagram of FAM, reported in Carpenter and Grossberg in 1992, and it has been adopted by various researchers in the field (e.g., [Kas93]), because it completely describe the functionality the ART architecture when dealing with classification problems. The ART architecture, depicted in Figures 3.1 (FAM) and 3.2 (EAM or GAM), has three major layers. The input layer F_1^a where the input patterns (designated by $\mathbf{I}$) are presented,

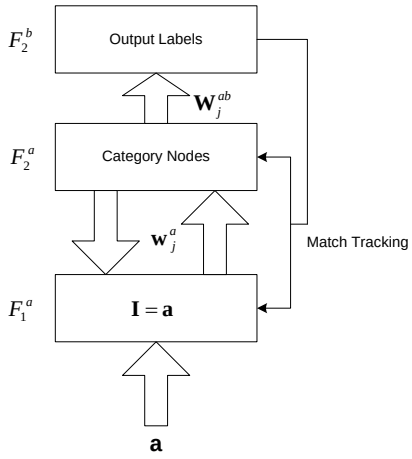


Figure 3.2: The block diagram of an EAM or GAM Architecture

the category representation layer F_2^a, where compressed representations of these input patterns (designated as $\mathbf{w}_j^a$), are formed, and the output layer F_2^b that holds the labels of the categories formed in the category representation layer. An additional layer, not shown in Figures 3.1 and 3.2, and designated by F_0^a , is a pre-processing layer and its functionality is to pre-process the input patterns, prior to their presentation to ART. The first level of ART pre-processing takes the input vector and normalizes it so that each one of its components lies in the interval [0, 1], and that is the only level of pre-processing needed for EAM and GAM. The second level of pre-processing (needed only for FAM) takes the normalized input vector, referred to as $\mathbf{a}$ and complementary encodes it, by appending to it another vector, referred to as $\mathbf{a}^c$. The complement of vector $\mathbf{a}$ is defined as

$$\mathbf{a}^c = (1 - a(1), 1 - a(2), ..., 1 - a(M_a)) \tag{3.1}$$

where

$$\mathbf{a} = (a(1), a(2), ..., a(M_a)) \tag{3.2}$$

and M_a, in the above equations, stands for the dimensionality of the input pattern of the pattern classification task under consideration. It is worth mentioning that the complementary encoding of the input patterns is necessary for the successful operation of Fuzzy ARTMAP (for an explanation see [GHH94]), however it is not needed by either EAM or GAM. Therefore, it is assumed that the inputs to FAM are normalized and complementary encoded, while the inputs to EAM and GAM are simply normalized (see Figure 3.1 for FAM, and Figure 3.2 for EAM and GAM). Note that normalization of inputs prior to their presentation to a neural network is common practice in the neural network literature.

ART can operate in two distinct phases: the training phase and the performance (test) phase. The training phase of ART can be described as follows: Given a set of inputs and associated label pairs, $\mathbf{I}_1, label(\mathbf{I}_1)$, $\mathbf{I}_2, label(\mathbf{I}_2)$, ..., $\mathbf{I}_{PT}, label(\mathbf{I}_{PT})$ (called the training set), it is desired to train ART to map every input pattern of the training set to its corresponding label. To achieve the aforementioned goal the training set is presented to the ART architecture repeatedly. That is, present $\mathbf{I}_1$ to F_1^a, $label(\mathbf{I}_1)$ to F_2^b, then $\mathbf{I}_2$ to F_1^a, $label(\mathbf{I}_2)$ to F_2^b, and finally, $\mathbf{I}_{PT}$ to F_1^a, $label(\mathbf{I}_{PT})$to F_2^b. The training set is presented to the ART network as many times as it is necessary for ART to correctly classify these input patterns. The task is considered accomplished (i.e., learning is complete) when

the weights in ART do not change during a training set presentation, or after a specific number of list presentations is reached.

```
repeat
    foreach p ∈ Training Set do
        Present-Pattern(p);
    end
until [number of epochs reached] or [no learning occurred] ;
```

Figure 3.3: Pseudo-code of ART Training

The learning process forms category regions whose dimensions and locations are determined by the training patterns presented to the learning algorithm. The effect is that patterns get encoded by these category regions. The selection of a category by a pattern is based on the Category Choice Function (CCF). The CCF acts as a measure of similarity of a pattern to a category. Patterns try to encode categories with highest CCF. When a category encodes a pattern, the category minimally expands to include that pattern in the input space.

Another mechanism is defined to track novelty in the input patterns. This is implemented by defining a Category Match Function (CMF). The CMF ensures that patterns are sufficiently close to the categories they are about to encode. If no category was found to be close enough, a new category is created. The CMF is compared against a user-defined parameter called vigilance parameter. Small values of the vigilance parameter lead to a broader generalization and higher pattern compression. Small values allow categories to encode patterns that are far away from each other, hence creating categories that are of relatively larger size. Vigilance parameter values close to 0 create categories that might fill the entire input space. Larger values, on the other hand, tend to make

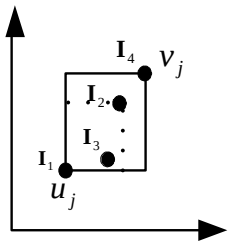
it harder for a pattern to be encoded by an existing category, and therefore, categories tend to be smaller. The vigilance parameter calibrates the minimum confidence used for deciding if a category should encode a given pattern. When a pattern is about to encode a category its label is checked. If it does not match that of the category, the confidence level is raised to trigger a mechanism called match tracking. This ensures that, in the future, this pattern will be matched with a category with correct label (see Figure 3.3 and Figure 3.4 for the Pseudo Code of the training phase in ART).

Algorithm: Present-Pattern(p)
input: training pattern p

Calculate CMF for each category for pattern p;
Define the set $S(p)$ of categories whose CMF exceeds the vigilance;
start:
if *S(p) is empty* **then**
    Add new node to encoding p;
else
    Calculate CCF for every category in $S(p)$;
    Find the category J that has the maximum CCF;
    if *label of J matches the label of p* **then**
        Category J learns pattern p by minimally expanding;
    else
        Set vigilance = CMF(J) + a small number;
        Remove J from $S(p)$;
        Go to start;
    end
end
Reset Vigilance to baseline value

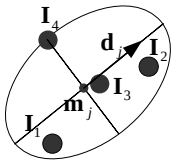
Figure 3.4: Pseudo-code of Present-Pattern(p)

The performance phase of ART works as follows: Given a set of input patterns (referred to as the test set), it is desired to find the ART output (label) produced when each one of the aforementioned test patterns is presented at the ART input layer. In



This hyperbox has encoded patterns $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \mathbf{I}_4$. In the figure, the $\mathbf{a}$ portion of these input patterns is depicted, as well as the lower end-point $\mathbf{u}_j$ and the upper endpoint $\mathbf{v}_j$ of this hyperbox.

Figure 3.5: A hyperbox category representation in FAM.



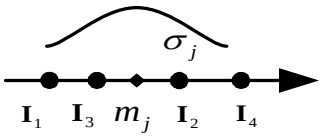
This ellipsoid has encoded patterns $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \mathbf{I}_4$. In the figure, the center point $\mathbf{m}_j$ and the direction vector $\mathbf{d}_j$ are shown, while the radius of the major axis, and the ratio of lengths of minor to major axis are easily deduced from the figure.

Figure 3.6: An ellipsoidal category representation in EAM

order to achieve this goal, the test set is presented to the trained ART network and the network's output is observed.

3.1.2 Geometric Interpretation of ART Categories

The weights (templates) in ART create compressed representations of the input patterns presented to the ART network during its training phase. These compressed representations have a geometrical interpretation. In particular, every node (category) in the category representation layer of Fuzzy ARTMAP (FAM) has template weights that com-

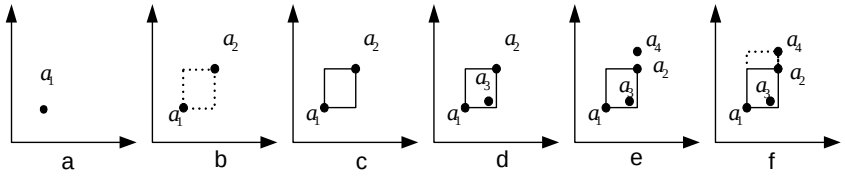


This Gaussian distribution has encoded patterns $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3, \mathbf{I}_4$. In the figure, the center point $\mathbf{m}_j$ and the standard deviation vector σ_j of the Gaussian curve are shown, while the number of points n_j that this Gaussian curve represents is easily deduced as being equal to 4.

Figure 3.7: A Gaussian curve category representation in GAM.

pletely define the lower and upper endpoints of a hyperbox. This hyperbox includes within its boundaries all the input patterns that chose this category as their representative category in FAM's training phase and were subsequently encoded by this category. Figure 3.5 shows the hyperbox of a category in a FAM architecture (2-D example), with lower endpoint $\mathbf{u}_j$, and upper endpoint $\mathbf{v}_j$, and the input patterns (the $\mathbf{I}$'s that it has encoded).

Also, every node (category) in the category representation layer of Ellipsoidal ARTMAP (EAM) has template weights that completely define an ellipsoid through its center, direction of major axis, length of the major axis, and ratio of lengths of minor axes to major axis in the ellipsoid. This ellipsoid includes within its boundaries all the input patterns that chose this category as their representative category in EAM's training phase and were subsequently encoded by this category. Figure 3.6 shows the ellipsoid of a category in a EAM architecture (2-D example), with center $\mathbf{m}_j$, direction of the major axis $\mathbf{d}_j$, length of the major axis, represented by its radius r_j (implied from the figure), ratio



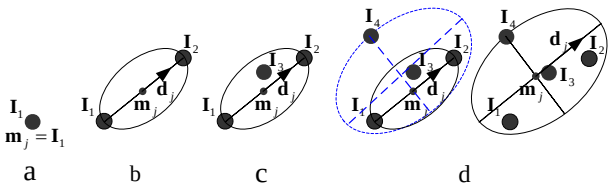
(a) A category with 0 size; (b) Introducing a new pattern $\mathbf{I}_2$, represented by $\mathbf{a}_2$; (c) The category expands to include $\mathbf{a}_2$; (d) Since new pattern $\mathbf{I}_3$, represented by $\mathbf{a}_3$ is inside the category, it doesn't change its size; (e) New Pattern $\mathbf{I}_4$, represented by $\mathbf{a}_4$ is presented; (f) Since $\mathbf{a}_4$ is outside the category, the category is expanded to include $\mathbf{a}_4$, within its boundaries

Figure 3.8: FAM learning (2-D Example)

of the lengths of minor axes to major axis μ (implied from the figure), and the input patterns $\mathbf{I}$'s that it has encoded.

Finally, every node (category) in the category representation layer of Gaussian ARTMAP has template weights that define the mean vector, the standard deviation vector of a multi-dimensional Gaussian distribution, and the number of points that are associated with this Gaussian distribution. The mean vector of this Gaussian distribution and the standard deviation vector of this Gaussian distribution are defined in terms of the means and the standard deviations (across every dimension) of the points that chose this node (category) as their representative category, while the number of the points associated with this Gaussian distribution are the number of points that chose this category as their representative category. Figure 3.7 shows the Gaussian distribution of a category in a GAM architecture (1-D example), with mean $\mathbf{m}_j$, standard deviation σ_j, number of points n_j (in this example $n_j = 4$), and the input patterns (i.e., $\mathbf{I}$'s) that it has encoded.

In essence, at the beginning of training, every category of FAM starts as a trivial hyperbox (equal to a point) and subsequently it expands to incorporate within its



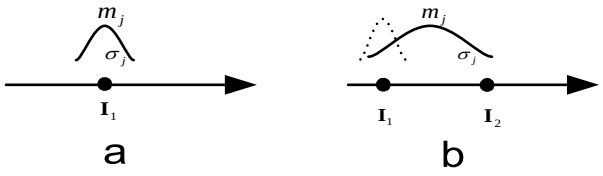
(a) A category with 0 size; (b) Introducing a new pattern $\mathbf{I}_2$; the category expands to include $\mathbf{I}_2$; (c) Introducing a new pattern $\mathbf{I}_3$; since the category includes $\mathbf{I}_3$, it does not change its size; (d) Pattern $\mathbf{I}_4$ is presented; since this pattern is outside the category, the category is expanded to include $\mathbf{I}_4$ within its boundaries.

Figure 3.9: EAM learning (2-D Example)

boundaries all the input patterns that in the training phase choose this hyperbox as their representative hyperbox, and are encoded by it (see Figure 3.8, where the category expansion of FAM is shown for an example dataset). The size of hyperbox is measured as the sum of the lengths of its sides.

Similarly, at the beginning of training, every EAM category starts as a trivial ellipsoid (equal to a point) and subsequently it expands to incorporate within its boundaries all the input patterns that in the training phase chose this ellipsoid as their representative ellipsoid, and are encoded by it (see Figure 3.9, where the category expansion of EAM is shown for an example dataset). The size of the ellipsoid is measured as the length of the major axis.

Finally, at the beginning of training, every category of GAM starts as a collection of Gaussian distributions in every dimension, with mean equal to the input pattern that was first encoded by this category, and a small standard deviation vector (part *a* of Figure 3.10); as training progresses in GAM this GAM category is modified to incorporate the information of the additional input patterns that are encoded by it (see part *b* of Figure



(a). A category with 0 size ; (b) Introducing a new pattern $\mathbf{I}_2$; the category characteristics (mean, standard deviation, of the Gaussian curve, as well as number of points encoded by the Gaussian curve) change to include the new knowledge that the new input pattern brings.

Figure 3.10: GAM learning

3.10 for an illustration of how the GAM category is modified for an example dataset). At any point in time the mean vector of this Gaussian distribution, corresponding to a category, is equal to the mean vector of all the input patterns encoded by the category, and the variance vector of the Gaussian distribution is equal to the variance vector corresponding to the input patterns that were encoded by the category. The ability of the category to encode new input patterns depends on the Mahalanobis distance of an input pattern from the mean/variance vectors of the Gaussian distribution corresponding to the category.

The performance of ART networks is measured in terms of the number of categories created in its training phase (small number of categories is good), and how well it generalizes on unseen data (high generalization accuracy is good). The performance of ART architecture depends on the choice of the vigilance parameter. It has been a known fact that performance in ART is also affected by the order according to which training data are presented to an ART architecture.

3.1.3 ART Category Proliferation Problem

One of the limitations of these ART architectures that has been repeatedly reported in the literature is the category proliferation problem. This refers to the creation of a relatively large number of categories to represent the training data. Categories are the hidden nodes (or units) in an ART neural network. Categories in ART are formed in order to compress the input data prior to mapping these compressed data to their respective labels. The creation of a large number of categories means poor compression of the training data. Quite often the category proliferation problem, observed in ART, is connected with the issue of overtraining. Over-training happens when ART is trying to learn the training data perfectly at the expense of degraded generalization performance (i.e., classification accuracy on unseen data) and also at the expense of creating many categories to represent the training data (leading to the category proliferation problem). Also, it has been related to several limitations of ART, such as the representative inefficiency of the categories or the excessive triggering of the match tracking mechanism due to existence of noise.

Since the early 1990's a number of ART modifications and improvements where published in the literature. These modifications tried to improve the ART's learning properties, improve speed, and address the category proliferation problem. In particular, a variety of ART innovations were proposed to combat the category proliferation problem in ART. In [MH95] PROBART was introduced. The authors suggest that the match tracking mechanism is the major cause of over-learning. PROBART eliminates the match tracking mechanism and instead stores probability information in the map field. In 1996, Gaussian ARTMAP (GAM) appeared in the literature [Wil96]. In this paper, the au-

thors attribute the category proliferation problem to two causes: sensitivity to noise and inefficiency of FAM categories. GAM operates in a similar fashion as FAM but it relies on a Gaussian-based measure of similarity in its operation, therefore eliminating the reliance on the inefficient category shapes of FAM and at the same time reducing ART's sensitivity to noise. Micro-ARTMAP [GDC] suppresses the match tracking mechanism and uses probabilistic map to encourage creation of large categories, and hence reduce their number. Safe Micro-ARTMAP [GDC01], introduced later, adds a mechanism to limit the growth of a category in response to a single pattern. Semi-supervised learning of ART was introduced [ABG03] in 2003, where it allows, with a certain tolerance, categories to encode patterns that are not mapped to the same label. This reduces the sensitivity to noise and hence the number of ART categories. Three semi-supervised architectures where introduced in [ABG03]: ssFAM, ssGAM and ssEAM. In [KGA01] the authors suggest the use of cross-validation to prevent over-training and therefore avoid the creation of unnecessary categories. In addition, the work by [CMN98], [Wil97] and [PGD] is worth mentioning, where the ART structure is changed from a winner-take-all to a distributed version and slow learning is employed with the intent of creating fewer ART categories and reducing the effects of noisy patterns. Finally, in order to address the inefficiency of FAM categories in representing data, Ellipsoid ARTMAP (EAM) was introduced in 2001 [Ana01]. EAM was similar to FAM except that it relied on category regions defined as hyper-ellipsoids rather than FAM's original hyper-rectangles.

3.1.4 GAM and High Dimensionality Problems

The category match function function (CMF) in Gaussian ARTMAP (GAM) can be expressed as in Equation (3.3):

$$\rho(I^r|w_j) = e^{[-\frac{1}{2}\sum_{i=1}^{M_a}[\frac{I_i^r-\mu_{ij}}{\sigma_{ij}}]^2]} \tag{3.3}$$

where M_a is the dimensionality of the data, μ_j is the mean vector for node j and σ_j is the standard deviation vector for node j.

From Equation (3.3) one can easily notice that when the dimensionality, M_a, of the data is large, the sum in the exponent can become quite large. This will cause $\rho(I^r|w_j)$ to be very small. When comparing $\rho(I^r|w_j)$ to the baseline vigilance during training, the vigilance test will almost always fail for values of the baseline vigilance other than 0. This will cause the creation of a new category for every pattern in the training data. Therefore, for datasets of large dimensionality, one should always set the vigilance parameter to 0; any other value would result in very poor performance. As it will be described in a later section, evolving GAM architectures relies on varying the vigilance parameter to train a diverse initial population of GAM networks. To be able to work with any problem regardless of dimensionality, in this work a slightly modified equation is used for the category match function (CMF). This modification does not change the basic operation of the algorithm; it only makes the range of vigilance parameter more equivalent for problems of low and high dimensionality. The modified CMF is expressed in Equation (3.3).

$$\rho(I^r|w_j)' = e^{[-\frac{1}{M_a}\sum_{i=1}^{M_a}[\frac{I_i^r-\mu_{ij}}{\sigma_{ij}}]^2]} \tag{3.4}$$

With some manipulations it is easy to show that this is equivalent to choosing a baseline vigilance that is transformed depending on the dimensionality, as shown in the following equation,

$$\bar{\rho}_a' = [\bar{\rho}_a]^{\frac{2}{M_a}} \tag{3.5}$$

It is worth mentioning that the modification above reduces the floating point computational errors because the exponent in the definition of the CMF can get too small for high dimensional problems.

3.2 The Improved Genetic ARTMAP Architectures

The genetic ART architectures were first introduced in [Al 06]. Three new architectures were introduced: Genetic Fuzzy ARTMAP (GFAM), Genetic Ellipsoidal ARTMAP (GEAM) and Genetic Gaussian ARTMAP (GGAM). In this work we will refer to these architectures collectively as GART. In this work several necessary improvements to these architectures were introduced. This section describes these architectures, highlighting the modifications that were made as a part of this effort. In the next section (section 3.3), a robust technique for finding the optimal algorithm parameters to be used in conjunction with these algorithms is introduced in order to get the best possible performance. In

section 3.4 a comparison of these improved architectures and other ART architectures is presented.

More details about the original implementation of these architectures can be found in [Al 06]. The pseudo-code for the improved genetic ART (GART) architectures is shown in Figure 3.11.

$P(0) \leftarrow$ Generate-Initial-Population();
for $t \leftarrow 1$ **to** Gen_{max} **do**
    Evaluation();
    if *stopping criteria met* **then** exit for;
    $P'(t) \leftarrow$ Selection($P(t)$);
    $P(t) \leftarrow$ Reproduction($P'(t)$);
end
return *Best Network in* $P(t)$;

Figure 3.11: Pseudo-code of the GART Algorithm

In this work we introduced the following improvements to the algorithms introduced in [Al 06].

- In the evaluation step, a fitness function is evaluated for each individual. The fitness function introduced in [Al 06] is as follows:

$$Fit(p) = \frac{(Cat_{max} - N_a(p))PCC^2(p)}{\frac{100}{Cat_{min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon} \tag{3.6}$$

  This fitness function was found to be unnecessarily complex. More importantly, this fitness function does not allow the user to tradeoff the two objectives optimized in this genetic algorithm: accuracy (PCC) and complexity (N_a). In this work the following fitness function is adopted instead.

$$Fit(p) = PCC(p) - \alpha(N_a(p) - Cat_{min}) \tag{3.7}$$

This fitness function allow the user to specify their preference of accuracy versus complexity. The parameter α can be controlled by the user. The simplicity of this function allow the user to predict the outcome of a certain setting of the parameter α. For example, a value of 0.5 indicates that one percentage of better correct classification of a network, or two categories less of a network, increase the fitness function by the same amount (i.e., by an amount of 1).

- Eliminate the genetic operator Cat_{add} because it was found ineffective. This operator adds a category to a network randomly. It is obvious that adding a category with random size, random location and random label has a very slim chance of producing a positive outcome. This observation was verified experimentally. This was done by running experiments on several datasets for several replications, and then observing statistically significant differences between the old and the new results. The Cat_{add} operator was found ineffective in improving the performance of the algorithm and therefore it was eliminated. This also eliminated one of the algorithm parameters that the user has to set appropriately.

- Introduced a performance dependent stopping criteria. In the original implementation [Al 06], the algorithm was run for 500 generations for every problem. In this work a stopping criteria was introduced in which the genetic process is stopped if no significant improvement is observed for a number of successive generations. This

resulted in significantly faster algorithms, especially for problems that are easier to solve and thus require a small fraction of the originally proposed 500 generations.

In the remainder of this section, the operation of the improved algorithms, GFAM, GEAM and GGAM and collectively referred to as GART is described. It is assumed that for every classification problem there is a training set, a validation test, and a test set. The training set is used for the training of GFAM, GEAM, and GGAM architectures under consideration. The validation set is used to optimize the produced GFAM, GEAM or GGAM network in ways that will become apparent in the following text. Finally, the test set is used to assess the performance of the optimized GFAM, GEAM, or GGAM network created.

GFAM, GEAM, and GGAM are evolved FAM, EAM, GAM networks, respectively, that are produced by applying, repeatedly, genetic operators on an initial population of trained FAM, EAM, or GAM networks. GFAM, GEAM, GGAM use deterministic tournament selection, as well as genetic operators, including crossover and mutation. In addition, GFAM, GEAM and GGAM use a special operator, Cat_{del}; this special operator is needed so that categories could be destroyed in the ART architectures, thus leading us, through evolution, to smaller ART structures.

3.2.1 Initialization

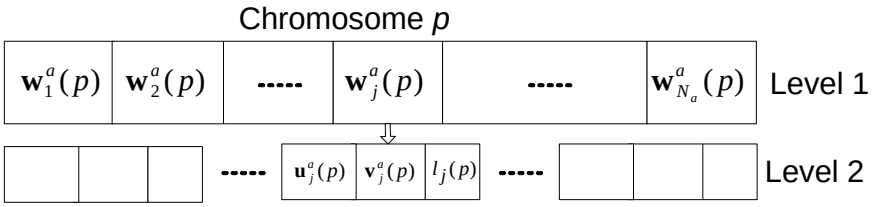
The algorithm starts by training Pop_{size} ARTMAP networks (FAM, EAM or GAM), each one of them trained with a different value of the baseline vigilance parameter, and

order of training pattern presentation (it has been a known fact that performance in ART is affected by the specific value of its baseline vigilance parameter, as well as the order of the training pattern presentation to the ART architecture). In this work the population size is fixed at $Pop_{size} = 20$. The choice parameter in a FAM network was chosen to be equal to 0.1. The choice parameter in an EAM network was chosen to be equal to 0.01. The ratio of the length of the minor axes to major axes in EAM was chosen equal to 1. The initial value of the standard deviation γ in a GAM network is chosen to be equal to 0.6. Also, the values of the base vigilance parameter for the networks in the initial population were equally distributed between a lower bound of $\bar{\rho}_a^{min} = 0.1$ and upper bound of $\bar{\rho}_a^{max} = 0.95$ for GFAM and GEAM, $\bar{\rho}_a^{min} = 0.1$ and $\bar{\rho}_a^{max} = 0.45$ for GGAM.

Once the Pop_{size} networks are trained, they need to be converted to chromosomes so that they can be manipulated by the genetic operators. GFAM, GEAM and GGAM use a real number representation to encode the networks. Each chromosome consists of two levels, level 1 containing all the categories of the network, and level 2 containing the template parameters needed to represent every category in level 1, as well as the label of every category in level 1. The chromosome encoding is explained in more detail in Figure 3.12 for GFAM, in Figure 3.13 for GEAM and in Figure 3.14 for GGAM.

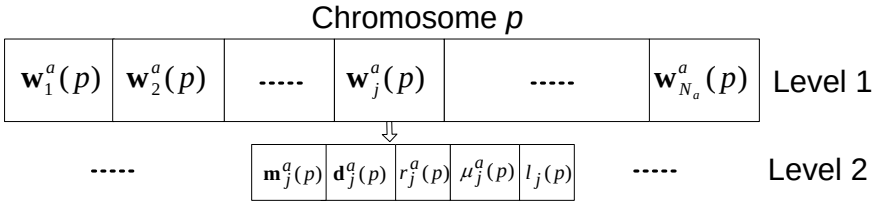
3.2.2 Evaluation

A weighed sum approach is used to define a fitness function that combines the two objectives of the optimization problem; $PCC(p)$ which designates the percentage of correct



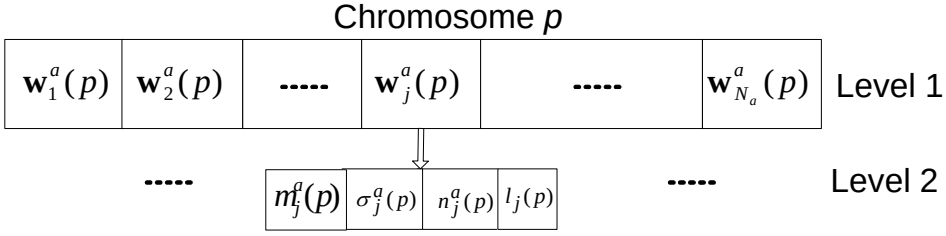
At level 2, the category's weight $\mathbf{w}_j^a$ contains the information about the lower end-point, $\mathbf{u}_j^a$, and the upper end-point, $\mathbf{v}_j^a$, of the hyperbox corresponding to the category, as well as the label l_j of the category.

Figure 3.12: GFAM chromosome structure



At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center, $\mathbf{m}_j^a$, the direction vector of the major axis, $\mathbf{d}_j^a$, the radius (half length) of the major axis, r_j, and the ratio of the lengths of the minor axes over the length of the major axis, μ_j, of the ellipsoid corresponding to this category, as well as the label l_j of the category.

Figure 3.13: GEAM chromosome structure



At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center of the Gaussian curve, $\mathbf{m}_j^a$, the standard deviation vector of the Gaussian curve, σ_j^a, and the number of points represented by the Gaussian curve, n_j, as well as the label l_j of the category.

Figure 3.14: GGAM chromosome structure

classification, exhibited by the p-th network, on the validation set, and $N_a(p)$ which designates the number of categories of the p-th network. The fitness function $Fit(p)$ of the p-th network is defined as follows:

$$Fit(p) = PCC(p) - \alpha(N_a(p) - Cat_{min}) \tag{3.8}$$

Obviously, this fitness function increases as $PCC(p)$ increases or as $N_a(p)$ decreases. The value of Cat_{min} is chosen to be equal to the number of classes of the classification problem at hand. It is evident from the fitness equation that, for $\alpha = 0.5$, one percentage of better correct classification of a network, or two categories less of a network, increase the fitness function by the same amount (i.e., by an amount of 1). This is one of the simplest ways of defining a fitness function that depends on two measures (generalization of the network and size of the network) and has been extensively adopted in the classification literature (e.g., see [BFO84]).

3.2.3 Selection

Initialize a temporary population P', where the parent chromosomes used to create the next generation are selected. The parents are chosen using deterministic tournament selection, as follows: Randomly select two groups of two chromosomes each from the current generation, and use as a parent, from each group, the chromosome with the best fitness value in the group. If it happens that from both groups the same chromosome

is chosen then we choose from one of the groups the chromosome with the second best fitness value.

The algorithm implements elitism as follows: it finds the best NC_{best} chromosomes (i.e., the chromosomes having the NC_{best} highest fitness values) from the current generation and copies them to the next generation without change. The value of NC_{best} was chosen to be 3 in this work.

3.2.4 Reproduction

Once the selection step determines the parents, reproduction operators are used to create individuals for the next generation. As expected, the reproduction operators are problem specific. In this section the reproduction operators used in evolving ARTMAP networks are described.

The two well-known operators for reproduction in GAs are crossover and mutation. In addition to crossover, two mutation-based operators are used. The first is referred to as the *Mutation operator*, and it performs Gaussian mutations on chromosomes based on a user-specified probability $Pr(Mut)$. The second operator, referred to as the Cat_{del}, deletes a category from a network based on a user-specified probability $Pr(Cat_{del})$. The mutation operator applies Gaussian perturbations of the weights at level 2 of the chromosome string (see Figures 3.12, 3.13, and 3.14). On the other hand, the Cat_{del} operator applies structural mutation at level 1 of the chromosome string.

3.2.4.1 Cat_{del}

The operator Cat_{del} deletes one of the categories of every chromosome in P' with probability $Pr(Cat_{del})$. If a chromosome is chosen to have one of its categories deleted then this category is picked randomly from the collection of the chromosome's categories.

3.2.4.2 Mutation

Every chromosome in P' gets mutated using Gaussian random number of mean of 0 and standard deviation of 0.01. The mutation is applied as described below:

- In *GFAM*, for each category, either its u or v endpoint is selected randomly (with 50% probability) and then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

- In *GEAM*, for each category, every component of the ellipsoidal center m gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Furthermore, the mutated category's axis ratio μ or radius r is selected with 50% probability. We add a small number, to the axis ratio or the radius. The small number is drawn from a Gaussian distribution. However if μ, or r, due to mutation, become larger

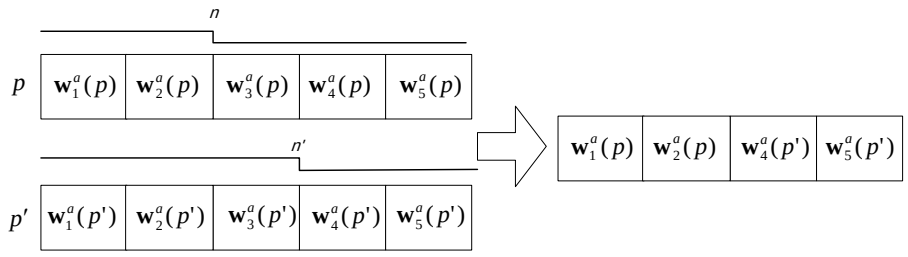
than 1, they are set back to the value of 1, while if they become smaller than zero we set their value to 0.0001.

- In *GGAM*, for each category, either its mean vector m, or standard deviation vector σ is selected randomly (50% probability). Then every component of this selected vector is mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

Notice that mutation is applied at level 2 of the chromosome structure. The label of the chromosome is not mutated because our initial GA population consists of trained networks, and consequently there is a lot of confidence in the labels of the categories that these trained networks have discovered through the ART training process.

3.2.4.3 Cross-Over Operation

The remaining $Pop_{size} - NC_{best}$ chromosomes are created by crossing over pairs of parents. For each parent, p, p', a random cross-over point is chosen, designated as n, n', respectively. Then, all the categories with index greater than n' in the chromosome p' and all the categories with index less than or equal to index n in the chromosome with index p are moved into an empty chromosome within the new generation. Notice that crossover is done at level 1 of the chromosome. This operation is pictorially illustrated in Figure 3.15.



In crossover the weight vectors of chromosome p, with index smaller than or equal to index n, and the weight vectors of chromosome p' with index larger than n', are combined (concatenated) to produce a new chromosome.

Figure 3.15: GFAM, GEAM, GGAM Crossover implementation

3.2.5 Stopping Criteria

If a stopping criterion is not met, go to the next iteration of the genetic evolution. Otherwise, terminate and return the best network. There is a need for an automated stopping criterion so that the evolution does not proceed for unnecessarily many generations. Ideally, the evolution should be allowed to proceed for as long as it is necessary, and it should terminate when network performance improvements are not attainable any more. In practice though, there is a tradeoff between network performance improvements and computational effort expended to achieve these improvements. It might be beneficial to use multiple stopping criteria to terminate the evolution of ART networks. One obvious stopping criterion is to set a threshold for the maximum number of generations, Gen_{max}, that the evolution is allowed to continue. The advantage of having this stopping criterion is that it ensures that the algorithms will always terminate and would not get trapped in an infinite loop if the other stopping criteria are never triggered. The user can always set the maximum number of generations to a large number to allow the algorithm to

terminate using other, more appropriate, stopping criteria. Another popular stopping criterion is to stop when no more improvement in fitness is observed. To ensure the lack of improvement is not due to the stochasticity of the search, the evolution is terminated only when no significant network performance improvements are observed for a number of consecutive evolutions. This number of consecutive evolutions can be chosen to be a percentage of the maximum number of generations Gen_{max}. In this work, Gen_{max} was chosen to be 500, and furthermore the evolution was stopped if 50 generations (10% of Gen_{max}) elapsed without an appreciable network fitness improvement. Appreciable network fitness improvement is an improvement larger than 0.01.

3.3 Selection of the GA Parameters

In this section we describe a method for selection of the algorithm parameters for GFAM, GEAM and GGAM. The performance of these algorithms rely on good choice of these parameters. The objective in this section is to provide good default values for these parameters based on a sound experimental exercise. This section is therefore devoted to the selection of good values for two parameters: probability of deleting an ART category, $Pr(Cat_{del})$, and probability of mutating an ART category, $Pr(Mut)$. As it is evident from our prior discussion there are a few other GA parameters that one has to carefully choose, such as Pop_{size}, Gen_{max}, and NC_{best}; we did not perform exhaustive experimentation to decide on the values of these parameters, but limited experimentation with these parameters showed that reasonable choices for these parameters were: $Pop_{size} = 20$,

$Gen_{max} = 500$, and $NC_{best} = 3$. The experiments in this section were conducted on 19 datasets. Detailed information about these datasets can be found in Appendix B.

The proposed approach to select good values for the GA parameters $Pr(Cat_{del})$, and $Pr(Mut)$ consisted of a number of steps delineated below:

- **Select GA Step 1:** Four different values for the $Pr(Cat_{del})$ were selected to experiment with. These were: 0.05, 0.1, 0.2, and 0.4. Also, four different values for the $Pr(Mut)$ parameter were selected to experiment with. These were: 0.0, 0.1, 0.2, and 0.4. This resulted in 16 combinations of parameter settings for $Pr(Cat_{del})$, and $Pr(Mut)$.

- **Select GA Step 2:** For each one of the 16 settings of the $Pr(Cat_{del})$, and $Pr(Mut)$ parameters, and for each of the 19 datasets, the GA optimization of FAMs, EAMs, and GAMs was applied 10 different times (using a different initial seed for the GA optimization). Consequently, for each database, and each parameter setting, and each of the genetically engineered ART algorithms 10 PCC and 10 N_a numbers were obtained.

- **Select GA Step 3:** For each genetically engineered ART algorithm (i.e., GFAM, GEAM, or GGAM), and each dataset, the best-performing (with respect to validation PCC of the 10 experiments) parameter setting was chosen. Then, ANOVA statistical tests were applied to choose other parameter settings that did not significantly differ (statistically) from the best performing parameter setting. These parameter settings were marked as good settings for this database and the associated GART (GFAM or GEAM or GGAM) algorithm.

- **Select GA Step 4:** After Step 3 was performed, for all databases and all genetically engineered ART algorithms we counted the number of databases for each GART algorithm for which a particular parameter setting was deemed as "good" from the Select GA Step 3. Based on these counts we recommended the best parameter setting for each GART algorithm, and a range of acceptable parameter settings.

Table 3.1 summarizes the results for GFAM. Similar tables have been produced for GEAM and GGAM but are omitted because they present similar information. In Table 3.1 an entry of "1" for a database indicates that the corresponding parameter setting performed well (with respect to the average PCC on the validation set). An underscored "1" entry indicates that the corresponding parameter setting performed the best for this database (with respect to the PCC on the validation set). In Table 3.1 the "1" entries corresponding to the Number of Categories criterion (actually average number of categories criterion) are omitted to preserve the table's clarity. However an entry of "1" for the PCC resulted also in an entry of "1" for the Number of Categories (not shown in Table 3.1). In Table 3.1, designated with an asterisk are the parameter setting, that performed best for this database (with respect to the average Number of Categories criterion). A careful observation of the results shown in Table 3.1 indicate that any value of $Pr(Cat_{del})$ in the interval $[0.2, 0.4]$, and any value of the $Pr(Mut)$ in the interval $[0.05, 0.4]$ gives good results. Also, the results from Table 3.1 indicate that the best performing parameter setting for GFAM is $Pr(Cat_{del}) = 0.1$, and $Pr(Mut) = 0.4$, since for this parameter setting we observe the highest number of good performances (19), and

best performances (7) of the associated GFAM (the count of the best performances consider the best observed performances with respect to the average PCC on the validation set or the average number of categories) . Finally, we can also deduce from the results of Table 3.1 that a probability of mutation equal to 0 is not recommended, since it always (for all databases) results in a GFAM network that is not performing well.

From similar tables produced for GEAM and GGAM (omitted) one can draw similar conclusions. In particular, a careful observation of the GEAM results indicate that any value of $Pr(Cat_{del})$ in the interval $[0.2, 0.4]$, and any value of the $Pr(Mut)$ in the interval $[0.05, 0.4]$ gives good results for GEAM. Also, the best performing parameter setting for GEAM is $Pr(Cat_{del}) = 0.2$, and $Pr(Mut) = 0.4$, since for this parameter setting we observe the highest number of good performances (19), and best performances (6) of the associated GEAM. Finally, a probability of mutation equal to 0 is not recommended for GEAM, since it always (for all databases) results in a GEAM that is not performing well. Additionally, a careful observation of the GGAM results indicate that any value of $Pr(Cat_{del})$ in the interval $[0.2, 0.4]$, and any value of the $Pr(Mut)$ in the interval $[0.05, 0.4]$ gives good results for GGAM. Also, the best performing parameter setting for GGAM is $Pr(Cat_{del}) = 0.4$, and $Pr(Mut) = 0.1$, since for this parameter setting we observe the highest number of good performances (19), and best performances (4) of the associated GGAM. Finally, a probability of mutation equal to 0 is not recommended for GGAM, since it always (for all databases) results in a GGAM that is not performing well.

Table 3.1: Goodness of Parameter Settings for GART

$Pr(Cat_{del})$, $Pr(Mut)$	g2c-05	g2c-15	g2c-25	g2c-40	g4c-05	g4c-15	g4c-25	g4c-40	g6c-05	g6c-15	g6c-25	g6c-40	4cirinsq	cirInSq	cins-70-30	2cins-50-30-20	Iris	abalone	pageblocks	sum	best
0.05	1																			1	
0.0																					
0.05	1	1	1	1	1	1	1	1	1	1	1		1		1	1	1	1	1	17	4
0.1																					
0.05	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	
0.2																			*		1
0.05	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	2
0.4																					
0.1																				0	
0.0																					
0.1	1	1	1	1	1	1	1		1	1	1		1		1	1	1	1	1	16	
0.1																					
0.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	1
0.2																					
0.1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	
0.4																	*		*		7
0.2	1																			1	
0.0																					
0.2	1	1	1	1	1	1	1	1	1	1	1	1	1		1	1	1		1	17	1
0.1																					
0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	
0.2																					1
0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	1
0.4																					
0.4																				0	
0.0																					
0.4	1	1	1	1	1	1	1	1	1	1	1		1	1		1	1	1	1	17	
0.1																					1
0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	1
0.2																					
0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	19	
0.4													*								2

3.4 Comparison with other ART architectures

The main purpose of the proposed algorithms, GFAM, GEAM, GGAM (collectively GART) is to solve the category proliferation problem while preserving the good performance that can be obtained by ART architectures. Therefore, to measure the merit of these architectures, in this section they are compared to other ART architectures that have previously appeared in the literature and addressed the category proliferation problem.

The ART architectures that were chosen to compare GFAM, GEAM, GGAM with are: ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP. The first three are based on the principle of semi-supervision, introduced [AGV02], and [VHG01]. Semi-supervision is a term attributed to learning in an ART architecture (FAM, EAM or GAM), where categories in ART are allowed to encode patterns of different labels provided that the percentage of patterns that belong to the plurality label exceed a certain threshold. Safe micro-ARTMAP is a Fuzzy ARTMAP that allows categories in Fuzzy ARTMAP to encode patterns that are mapped to different labels. In safe micro-ARTMAP (see [GDC01]) though the mixture of labels allowed in a category, or in all of the categories is controlled by the entropy of the category or categories.

As mentioned in an earlier section, in every classification problem there is a training set, a validation test, and a test set. The training set is used for the training of GART (ART) architectures under consideration. The validation set in the GART case is used to guide the evolution of the trained ART networks from generation 1 to generation Gen_{max}. The validation set in the other ART networks' case is used to choose optimal

values for the ART network parameters (e.g., vigilance, choice parameter, order of pattern presentation, etc); optimal values of ART network parameters are the ones that give the highest value of the already defined fitness function. The test set is used to assess the performance of the optimized GART (ART) network. The percentages of different class data-points in the training, validation and test set are the same as the ones found in the original dataset.

For each of the ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP networks, a number of experiments were performed with different settings of their network parameter values on 19 datasets (for more details about the datasets, please see Appendix B). For each one of these network parameter settings the resulting network's fitness function was calculated (the same fitness function was used as the one utilized for the GART networks (see equation (3.8))). For the training of ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP the same training set was used as the one used for the GART networks, and for the validation of the performance of each of the ssFAM, ssEAM, ssGAM, and Safe micro-ARTMAP networks the same validation set was used as the one used for the GART networks. The parameter setting of the ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP network that maximized the fitness function was chosen as the best parameter setting for the specific database; the number of categories created by the "best" parameter setting network, and its corresponding percentage of correct classification on the test set are reported in Table 3.2.

In particular, the parameter settings that we experimented with ssFAM were: baseline vigilance values ranging from 0 to 0.9 with step size of 0.1, choice parameter values of

0.001 and 0.01, maximum allowable mixture threshold values ranging from 0 to 1 with step size of 0.1, and 100 different orders of pattern presentations of the training data (resulting in 22,000 different parameter settings). Furthermore, the settings for ssEAM were: baseline vigilance values ranging from 0 to 0.9 with step size of 0.1, choice parameter values of 0.001 and 0.01, maximum allowable mixture threshold values ranging from 0 to 1 with step size of 0.1, minimum axes to maximum axis ratio values ranging from 0.1 to 1 with step size of 0.1, and 100 different orders of pattern presentations of the training data (resulting in 220,000 different parameter settings). Also, the settings for ssGAM were: baseline vigilance values ranging from 0 to 0.9 with step size of 0.1, initial standard deviation parameter ranging from 0.1 to 1 with step size of 0.1, maximum allowable mixture threshold values ranging from 0 to 1 with step size of 0.1, and 100 different orders of pattern presentations of the training data (resulting in 110,000 different parameter settings). Finally, the settings for safe micro-ARTMAP were: baseline vigilance values ranging from 0 to 0.4 with step size of 0.2, baseline vigilance parameter values of 0.001 and 0.01, 5 values for the maximum "all-categories" entropy threshold, 6 different ratios of the values of the "categories" entropy threshold to the "all-categories" entropy threshold, three values of the maximum allowable expansion of a category, and 100 different orders of pattern presentations of the training data (resulting in 90,000 different parameter settings).

The best parameter setting, identified in the previous sub-section, for GFAM, GEAM, and GGAM was used for each of the 19 databases. Ten (10) experiments per database were conducted for 10 different initial seeds of the GA optimization process. The network

that produced the maximum value of the fitness function, was deemed as "best" The number of categories of the "best" GFAM, GEAM and GGAM for each database and its corresponding performance (PCC) on the test set are reported in Table 3.2.

Some of the conclusions that can be deduced from the comparative results, depicted in Table 3.2, are emphasized below:

- **Observation 1 (Overall Performance of GART networks):** GFAM, GEAM and GGAM attain good performance on all the datasets, and quite often, optimal performance (e.g., see performance of all the networks in the Gaussian databases, and performance of GGAM on the structures-within-structure problems, and on the real databases). The best performing network from the class of GART networks (GFAM, GEAM, and GGAM) is GGAM.

- **Observation 2 (Comparative Performance of GART networks, with respect to each other)**. GGAM and GEAM outperform the performance of GFAM on all the structures, within structure problems. For all the other problems the differences between GEAM, and GGAM versus GFAM are not statistically significant.

- **Observation 3 (Comparative Performance of GART networks compared with ssFAM):** ssFAM performs as well as the GART networks for the 2-class Gaussian datasets. For all the other datasets at least one (if not all) the GART networks perform better (achieving higher PCC with fewer ART categories). The largest difference in PCC observed is almost 12% (for the 4 Circle in the Square problem), while the largest ratio of number of ssFAM versus GART categories is for the modified IRIS problem (ratio of 4).

Table 3.2: Comparison of GART with ssART

Best results obtained from GFAM, GEAM and GGAM compared to best results obtained from Safe μARTMAP, ssFAM, ssEAM and ssGAM

Database	GFAM		GEAM		GGAM		Safe μAM		ssFAM		ssEAM		ssGAM	
Name	PCC	Size	PCC	Size	PCC	Size	PCC	Size	PCC	Size	PCC	Size	PCC	Size
G2c-05	95.4	2	95.3	2	95.3	2	95.2	2	94.7	2	94.6	2	94.6	2
G2c-15	85.3	2	85.2	2	85.2	2	85.0	2	85.5	2	85.2	2	85.5	2
G2c-25	75.2	2	75.2	2	75.2	2	74.9	2	75.0	2	75.1	2	75.0	2
G2c-40	62.0	2	61.8	2	61.7	2	61.4	3	59.5	2	59.5	2	59.5	3
G4c-05	95.1	4	95.0	4	95.0	4	95.0	4	94.5	5	94.9	4	95.5	4
G4c-15	84.7	4	84.6	4	84.7	4	83.2	4	82.7	4	82.0	4	83.4	6
G4c-25	75.0	4	75.1	4	75.3	4	74.5	4	70.3	9	72.9	4	72.3	21
G4c-40	59.9	4	59.8	4	75.3	4	58.9	4	57.8	7	54.7	7	59.5	14
G6c-05	94.8	6	94.7	6	94.8	6	92.3	6	87.2	8	93.4	6	94.6	8
G6c-15	84.8	6	85.1	6	85.2	6	80.9	6	80.5	6	82.0	7	83.4	9
G6c-25	74.3	6	74.1	6	74.4	6	67.9	6	70.2	15	71.4	7	71.2	13
G6c-40	60.1	6	59.9	6	60.0	6	54.0	6	55.1	17	49.3	7	55.1	13
4Ci/Sq	95.0	9	99.1	7	98.9	6	95.4	8	87.2	18	94.6	5	93.4	12
1Ci/Sq	97.7	7	99.6	3	99.8	2	94.7	8	92.9	8	97.0	8	91.0	8
30:70	97.9	6	99.9	2	99.9	2	96.8	8	93.2	8	97.1	8	92.3	8
20:30:50	97.5	5	98.1	3	99.5	3	97.2	6	90.2	12	97.0	3	95.6	9
MOD-IRIS	95.3	2	95.3	2	94.9	2	94.9	2	94.7	8	94.7	2	94.7	2
ABALONE	61.8	3	62.2	3	62.6	3	58.1	3	60.0	6	58.8	3	56.3	2
PAGE	96.7	5	95.0	5	96.2	5	92.9	5	87.9	3	93.8	2	94.3	5

- **Observation 4 (Comparative Performance of GART networks compared with ssEAM):** ssEAM performs as well as the GART networks for the 2-class Gaussian datasets. For all the other datasets at least one (if not all) the GART networks perform better (achieving higher PCC with fewer ART categories). The largest difference in PCC observed is more than 10% (for the 6 class Gaussian problem with 40% overlap), while the largest ratio of number of ssEAM versus GART categories is for Circle in the Square problem (ratio of 4).

- **Observation 5 (Comparative Performance of GART networks compared with ssGAM):** ssGAM performs as well as the GART networks for the 2-class Gaussian datasets. For all the other datasets at least one (if not all) the GART networks perform better (achieving higher PCC with fewer ART categories). The largest difference in PCC observed is more than 8% (for the 1 Circle in the Square problem), while the largest ratio of number of ssGAM versus GART categories is for the four Gaussian dataset with 25% overlap problem (ratio larger than 5).

- **Observation 6 (Comparative Performance of GART networks compared with safe micro-ARTMAP):** Safe micro-ARTMAP performs as well as the GART networks for the 2-class, and 4-class Gaussian datasets. For all the other datasets at least one (if not all) the GART networks perform better (achieving higher PCC with fewer ART categories). The largest difference in PCC observed is more than 6% (for the 6 class Gaussian dataset with 25% overlap), while the largest ratio of number of safe micro-ARTMAP versus GART categories is for the Circle in the Square problem (ratio of 4).

What is also worth pointing out is that the better performance of the GART network is attained with reduced computations as compared with the computations needed by the alternate methods (ssFAM, ssEAM, ssGAM, safe micro-ARTMAP). Specifically, the performance attained by ssFAM, ssEAM, ssGAM and the safe micro-ARTMAP required training these networks for a large number of network parameter settings (at least 22,000 experiments) and then choosing the network that achieved the highest value for the fitness function that we introduced earlier (through cross-validation). In GFAM, GEAM and GGAM cases we trained only a small number of these networks ($Pop_{size} = 20$ of them), compared to the large number of networks trained in the ssFAM, ssEAM, ssGAM or micro-ARTMAP cases (at least 22,000). Furthermore, in GFAM, GEAM and GGAM cases we evolved the trained networks $Gen_{max} = 500$ times, each evolution requiring cross-validating $Pop_{size} = 20$ networks. Hence, the total number of networks cross-validated in the ssFAM, ssEAM, ssGAM and micro-ARTMAP cases were at least 22,000, while in the GFAM, GEAM and GGAM networks were 10,000; furthermore the networks cross-validated in the ssFAM, ssEAM, ssGAM, and micro-ARTMAP cases have higher number of category nodes than the ones cross-validated in the GFAM, GEAM and GGAM cases. As a result, we can argue that the improved performance (smaller number of nodes and better generalization) of GFAM, GEAM, and GGAM, compared with ssFAM, ssEAM, ssGAM, and micro-ARTMAP, is achieved with reduced computational effort. Of course, one can claim that such an extensive experimentation with these networks might not be needed, especially if one is familiar with the functionality of these networks and chooses to experiment only with a limited set of network parameter values. However, the practitioner in the field might lack the expertise to carefully choose the network

parameters to experiment with, and consequently might need to experiment extensively to come up with a good network. In this case, GART has an advantage over the other ART network approaches because it has already provided a list of default parameter settings for the evolution of trained ART classifiers, and as a result the experimentation with a separate validation set is not needed.

The comparison of GART, and ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP provided above, is fair because it used the same databases and datasets per database for training, validation and testing of these architectures, as well as the same criterion for finding the best of these ART architectures (the criterion was to maximize the fitness function, defined in Equation (3.8)).

In all the experiments above, simulated or real datasets were used that predominantly had input patterns of dimensionality 2. Furthermore, for the GART results reported in Table 3.2, these datasets were used to identify good (default) GA parameter values. It is therefore worth reporting GART's performance on datasets that have input patterns of higher than 2 dimensionality, and for which the good GA parameter used are the values identified in the previous section using the original 19 datasets. The results for GFAM (PCC on the test sets, and number of categories created) are depicted in Table 3.3 (in particular, Table 3.3 shows the GFAM results for two α parameter values of Equation (3.8); a value of 0.5 (this is the value used for the Table 3.2 results) and a value of 0.1. Note that a smaller parameter value allows for higher size GART networks that end up exhibiting higher accuracy (PCC) on unseen data.

Table 3.3: Accuracy and size results achieved by GFAM on 8 UCI databases. The results are recorded for 2 settings of the fitness function parameter: 0.5 and 0.1.

Database Name	GFAM (0.5)		GFAM (0.1)	
	PCC	***Size***	***PCC***	***Size***
OPTDIGITS	88.09	13	91.21	22
PENDIGITS	90.25	15	94.35	28
SAT	83.35	7	84.6	8
SEG	94.13	12	95.14	15
WAV	81.55	3	83	4
SHUTTLE	99.55	5	99.55	5
PIMA	77.59	2	76.72	3

In order for the reader to be able to evaluate how good the GFAM results are the reader is refers to the work by Lim, Loh and Shih, [LLS00], where they compared the accuracy and size of a 33 classifiers belonging to the tree, statistical and neural types classifiers. Three of the datasets that Lim, Loh and Shih have experimented with are the Satellite, the Segmentation and the Waveform datasets that GFAM has been tested on (see Table 3.3). The GFAM results on the Satellite dataset are: 83.35% (84.6%) classification accuracy, needing 7 (8) categories (see Table 3.3). The accuracy results reported on the Satellite dataset by Lim, Loh, and Shih are: Minimum classification accuracy of 60% and maximum classification accuracy of 90%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 8, while the median tree size was 63. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 63 and 216. The GFAM results on the Segmentation dataset are: 94.13% (95.14%) classification accuracy, needing 12 (15) categories. The accuracy results reported on the Segmentation dataset by Lim, Loh, and Shih are: Minimum classification accuracy of 48% and maximum classification accuracy of 98% (achieved by the nearest neighbor classifier, which performs no data compression). Furthermore the

tree type classifiers (22 of them) created a minimum tree size of 6, while the median tree size was 39. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 69 and 42. The GFAM results on the Waveform dataset are: 81.55% (83%) classification accuracy, needing 3 (4) categories. The accuracy results reported on the Waveform dataset by Lim, Loh, and Shih are: Minimum classification accuracy of 52% and maximum classification accuracy of 85%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 3, while the median tree size was 16. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 14 and 54.

3.5 Summary

In this work a number of improvements are proposed to the genetic ART architectures introduced in [Al 06]. These improvements resulted in a more efficient, effective and more elegant approach. Furthermore, in this chapter a sound technique for finding the default algorithm parameters was employed to provide the users of GART with default values that are guaranteed to work well with most datasets. Finally, a comparison was carried to prove the merit of the GART architectures compared to other ART architectures that appeared in the literature and tried to solve the category proliferation problem.

GART networks were found to be superior to a number of other ART networks (ssFAM, ssEAM, ssGAM, safe micro-ARTMAP) that have been introduced into the literature to address the category proliferation problem in ART. More specifically, GFAM,

GEAM, and GGAM gave a **better generalization performance** (in almost all problems tested) and a **smaller than, or equal, size** network (in all problems tested), compared to these other ART networks, **requiring reduced computational effort** to achieve these advantages. More specifically, in some instances the difference in classification performance of GFAM, GEAM, and GGAM with these other ART networks quite significant (as high as 12%). Also, in some instances the ratio of the number of categories created by these other ART networks, compared to the categories created by GFAM, GEAM or GGAM was large (as high as 5).

In this chapter the effect of the improvements introduced over the GA implementation provided in [Al 06] were not fully quantified since the the two approaches are similar in many ways. However, limited experimentation show that similar network quality was achieved for some datasets. However the proposed approach reduces the overall training time significantly. For example for the G4C-25 dataset, the training time was reduced from 77 seconds to 12 seconds (or 84% reduction). For harder datasets, due to the fitness function defined in [Al 06], it is not possible to achieve competitive generalization. For example on the Pendigits dataset, the best accuracy achieved by the architecture introduced in [Al 06] was 91%, where as shown in the results in Table 3.3, by properly controlling the parameter in the fitness function, the proposed approach was able to achieve 94% accuracy. In future chapters the product of the improved genetic ART architectures will be used as a baseline for quantifying further refinements of the genetic evolution of ART architectures.

CHAPTER 4
SELF ADAPTATION IN GENETIC ALGORITHMS

In the previous chapter, genetic ARTMAP (GART) was described. In this chapter an adaptive genetic algorithm is applied to Fuzzy ARTMAP (FAM), ellipsoidal ARTMAP (EAM), and Gaussian ARTMAP (GAM). One of the major advantages of the proposed genetic algorithm is that it adapts the GA parameters automatically, and in a way that takes into consideration the intricacies of the classification problem under consideration. The resulting genetically engineered ART architectures are referred to as AG-FAM, AG-EAM and AG-GAM or collectively as AG-ART (adaptive genetically engineered ART). The performance (in terms of accuracy, size, and computational cost) of the AG-ART architectures is compared with GART, and other ART architectures that have appeared in the literature and attempted to solve the category proliferation problem (ssFAM, ssEAM, ssGAM). The results demonstrate that AG-ART architectures exhibit better performance than their other ART counterparts (ssFAM ssEAM and ssGAM) and better performance than GART. Also AG-ART's performance is compared to other related results published in the classification literature to demonstrate that AG-ART architectures exhibit competitive generalization performance and, quite often, produce smaller size classifiers in solving the same classification problems. AG-ART's performance gains is shown to be achieved within a reasonable computational budget.

4.1 Introduction

In this chapter an improved GA for the evolution of ART architectures is proposed. The proposed GA relies on adaptive GA parameter control mechanisms. While GART required the user to choose values for the probability of an ART category deletion, and for the probability of a category mutation, AG-ART finds good values for these parameters through an adaptation mechanism that takes into consideration the specificities of the problem under consideration. Hence, not only AG-ART is more elegant (requires minimal user intervention) than GART, but as experiments illustrate it is faster than GART, because the GA parameters are more wisely chosen through appropriate adaptation mechanisms.

4.2 Adaptation in Genetic Algorithms: Literature Review

When applying a GA to solve an optimization problem, it is not only needed to choose the algorithm, representation, and operators for the problem, but what is also needed is to choose parameter values and operator probabilities for the GA so that it will not only find the solution, but also find the solution efficiently. In many cases, researchers choose these parameter values and operator probabilities based on experience or experimentation on a specific problem. The problem of finding good GA parameter values and good operator probability values has been addressed before in the literature. A number of researchers suggested good parameter values as a result of extensive experimentation on a range of optimization problems. For examples, [Jon75] proposed to set $p_m = 0.001$ and $p_c = 0.6$,

where p_m and p_c stand for the probability of mutation and crossover, respectively. In [MS93] the authors propose $p_m = \frac{1}{\beta}$ and in [Bac92] $p_m = \frac{1.75}{n*\beta^{\frac{1}{2}}}$, where n represents the population size and β represents the bit-length of a chromosome.

Finding good GA parameter values for a certain optimization problem is a time consuming process. It is prone to human error which may lead to suboptimal results. Moreover, what might be initially considered as a good parameter setting could, in the progress of evolution, prove not to be as good, since the search may move to different regions of the solution space. As a result, an emphasis was placed on designing GAs where the parameters automatically adapt to the problem at hand.

The GA parameters that affect its performance include environmental parameters such as population size and the objective (fitness) function. The adaptation can be applied to global parameters such as mutation rate, mutation strength or crossover rate that are affecting all individuals in the population, or applied to local parameters where the parameter value is customized for each individual. Also, some existing research (for example, see [FAF95]) proposes the customization of the parameter setting at the component level (part of the individual).

Adaption provides the opportunity to customize the evolutionary algorithm to the problem and to modify the configuration and the strategy parameters used while the algorithm is running. This enables the GA to not only incorporate domain information and multiple reproduction operators more easily, but can allow the algorithm itself to select those values and operators which might give better results. Also these values can be modified during the run of the GA to suit the specific situation during that part of the

run. When the information about which of the operators available are most suitable to a particular problem is not easily determined, adaptation can be used to provide feedback or to determine when they should be used.

The majority of the research though focuses on adapting the mutation rate or mutation strength. For real-valued representations, the term mutation strength, sometimes called mutation step size, refers to the magnitude of change in each mutated variable. This is different in binary representation schemes, where the term mutation rate is used to express how probable it is for a certain binary variable to be changed (inverted).

The GA adaptation approaches can be distinguished in three groups, in order of increasing complexity: *Deterministic*, *Adaptive*, and *Self-Adaptive* (see [HME97]). Each of these approaches is described below.

- Deterministic: Deterministic adaptation refers to the dynamic adjustment of a GA parameter using a deterministic rule, and without feedback from the quality of the solution achieved by the evolutionary process. This rule can be based on a schedule (similar to simulated annealing) or number of generations (see [Fog89]). The objective in this approach is to alter the GA parameters in such a way that results in a wide-spread search at the beginning of the optimization, and increasingly localized search at later stages. An example of this approach can be found in [LTD02], where the mutation step sizes are discounted by a constant factor each time an offspring is produced.

- Adaptive: This approach uses some form of feedback from the GA that is used to determine the direction and/or magnitude of the change to the GA parameter.

In [ASP94], the standard deviation of the Gaussian mutation was varied based on a temperature parameter. The temperature parameter was defined as $T(\eta) = 1 - \frac{f(\eta)}{f_{max}}$, were f_{max} is the maximum fitness for a given task. Thus, the temperature of a solution is determined by how close the solution is to being an optimal solution for the task under consideration. Solutions with a high temperature are mutated severely, and those with a low temperature are mutated only slightly. This allows a coarse-grained search initially, and a progressively finer-grained search as the GA approaches a solution for the assigned task.

In [SP94] the authors suggest the use of a feedback signal that is defined by the difference $fit_{max}(P,t) - \mu(P,t)$, where $fit_{max}(P,t)$ is the maximum fitness and $\mu(P,t)$ is the average fitness of solutions in population $P(t)$ at generation t. This difference is used as an indication of closeness to convergence. This difference is likely to be less for a population that has converged to an optimal solution than that for a random population, scattered in the solution space. The authors define the adaptive rates of crossover and mutation for all chromosomes to be inversely proportional to this difference. To make this mechanism less disruptive for good solutions, the mechanism is adjusted to have low values of parameters for high fitness values and high values of parameters for low fitness values, as follows: $p_m(x) = k_1 \frac{fit_{max}(P,t) - fit(x,t)}{fit_{max}(P,t) - \mu(P,t)}$, and $p_c = k_2 \frac{fit_{max}(P,t) - fit(x,t)}{fit_{max}(P,t) - \mu(P,t)}$. Therefore, in this case, the parameters are controlled at the individual level.

- Self-Adaptive: In this approach, the GA parameters undergo evolution. The GA parameters are encoded in the chromosomes and evolved as part of the solution.

The encoded parameters will lead to better fitness for individuals with better parameters values, and since these individuals are more likely to survive and reproduce, this mechanism will propagate these better parameter values.

Self adaptation differs from the adaptive approach in that in the adaptive approach, the feedback from the GA is used deterministically to change the GA parameters. Instead, self-adaptation allows the GA to determine the best GA parameters at any time in the evolutionary process. In other words, in the adaptive approach, it is usually assumed that the GA should reduce the scope of the search as it approaches the optimum solution. In the self-adaption, this assumption is not made, and therefore the parameters are allowed to be changed in the best way that will lead to better solutions.

4.3 Adaptive Approach to Evolving ART Architectures

Adaptation was applied to the genetic ART architectures introduced in Chapter 3. In the implementation of genetic optimization of ARTMAPs, referred to as AG-ART, an *adaptive* approach was chosen, where a feedback signal is used to determine the operator probability at the *component level*.

The pseudo-code for the genetic ART architectures is listed in Figure 3.11 and repeated below in Figure 4.1. The proposed adaptation is applied to the Reproduction step. Therefore, in this section, the Reproduction step is described in details.

```
P(0) ← Generate-Initial-Population();
for t ← 1 to Gen_max do
    Evaluation();
    if stopping criteria met then exit for;
    P'(t) ← Selection(P(t));
    P(t) ← Reproduction(P'(t));
end
return Best Network in P(t);
```

Figure 4.1: Pseudo-code of the AG-ART Algorithm

To avoid the need for finding proper values for the mutation and pruning probabilities, or setting default values that might result in suboptimal operation, an adaptation mechanism was employed to automatically adjust, based on performance, the invocation of reproduction operators. This performance based adaptation is implemented at the gene (category) level. More specifically, adaptive, performance based, parameters are computed for each component in the individual. The performance feedback relies on a metric defined for each category, referred to as the *confidence factor*, CF.

The confidence factor is a metric that measures the performance at the category level. Since our objective is to find a network with good generalization and small size, the performance of a category is defined in terms of accuracy and frequency of selection of the category. We favor accurate and frequently selected (therefore larger) categories. The assumption here is that if categories are frequently selected, the network size would likely to be smaller. The confidence criteria used, were based on similar confidence criteria designed by other researchers in the field (see, [CT95], [TRL06]), whose objective was to prune under-performing ART categories. The confidence factor is defined, for every category j of the $p-th$ ART network, that is mapped to label k, as follows:

$$CF_j^k(p) = 0.5A_j^k(p) + 0.5S_j^k(p) \tag{4.1}$$

where $A_j^k(p)$ is a measure of accuracy of classification achieved by category j, in the $p-th$ network, that is mapped to label k. Furthermore, $S_j^k(p)$ is a measure of probability of selection of category j in the $p-th$ network, that is mapped to label k.

The accuracy measure, $A_j^k(p)$, is defined as follows: the probability of correct classification for category j divided by the maximum probability of correct classification for any category in the same network ($p-th$ network) that predicts the same class label, k. This measure assumes higher values for categories that are performing relatively well. In particular, if the number of validation samples that selected this category, and were correctly classified by it, is denoted by $P_j^k(p)$, and the number of validation samples that selected this category is denoted by $C_j^k(p)$, then,

$$A_j^k(p) = \frac{P_j^k(p)/C_j^k(p)}{\max_j(P_j^k(p)/C_j^k(p))} \tag{4.2}$$

We also define $S_j^k(p)$ as the probability of selection by the validation patterns of a category, j, of the $p-th$ network, that is mapped to label k. The probability of selection of category j, of the $p-th$ network, that is mapped to label k, is the number of validation patterns that selected this category, $C_j^k(p)$, divided by the maximum number of patterns $C_{j_{max}}^k(p)$ that selected any category j that predicts the same classification label, k, for the $p-th$ network:

$$S_j^k(p) = C_j^k(p)/C_{j_{max}}^k(p) \tag{4.3}$$

This measure achieves higher values for categories that were selected more often using the validation patterns. The scaling ensures that $A_j^k(p) \in [0,1]$, $S_j^k(p) \in [0,1]$ and therefore $CF_j(p) \in [0,1]$. In addition, in every network, at least one category has $A_j(p) = 1$, and at least one category (but not necessarily the same) has $S_j(p) = 1$. Therefore, in every generation the confidence factor is calculated for every category based on the performance on the validation set.

4.3.1 Prune

To be able to create smaller networks using the evolutionary search, a genetic operator, *Prune*, is introduced, that deletes categories from a network using some appropriate selection criteria. Pruning is prohibited if it violates the *class inclusion criterion*. The class inclusion criterion dictates that in every network there is at least one category for each class label present in the data. It is obvious that the criterion used for selection of categories to be pruned affects the efficiency of the genetic search. One selection criterion is to randomly prune a category using a user-specified probability. However, this criterion does not exploit the knowledge we have about the performance of a category on the validation set after every generation. It might be beneficial to take this information into consideration when deciding on which categories to be deleted. However, complete reliance on this knowledge would result in a hill-climbing search that would probably end

up at local optima. To avoid this situation a probabilistic approach is used that increases the chance of deletion of under-performing categories. This way, the search is directed towards better solutions, but not limited from exploring other regions of the solution space. Consequently, in AG-ART, with probability of $1 - CF_j^k(p)$, categories are delete from every chromosome in the temporary population, $P'(t)$.

This operator replaces the Cat_{del} operator described in Chapter 3. While Cat_{del} deletes one category from a network based on probability $Pr(Cat_{del})$, the $Prune$ operator probabilistically deletes multiple categories from a network, giving more likelihood to deletion of categories with low CF value. It can be noticed that the $Prune$ operator reduces the size of networks quickly by eliminating under-performing categories. Also, the $Prune$ operator does not rely on a user-specified probability that is set constant through out the search; but rather, the CF values are updated in every generation and used to dynamically guide the pruning operator.

4.3.2 Mutation

Every chromosome gets mutated as described in Chapter 3, Section 3.2. As described earlier, mutation is implemented by using a small Gaussianly-distributed random quantity to bring about change in the individuals. However, instead of applying the $Mutation$ with fixed standard deviation for the Gaussian distribution, here the standard deviation is controlled for each category based on its performance. In other words, the Gaussian distribution has a mean of 0 and a standard deviation that is equal to a *severity factor*,

(SF), that is calculated for each category based on its performance. A performance based severity of mutation should be used to impose higher probability of mutation to those categories that do not perform well. The following expression is used to control the severity of mutation:

$$SF_j^k(p) = 0.05(1 - CF_j^k(p)) \tag{4.4}$$

The ART category values are then mutated by random numbers, chosen from the following distribution:

$$Normal(0, SF_j^k(p)) \tag{4.5}$$

Therefore, again the $Mutation$ is adapted at the component (category) level. The user-specified parameter, $Pr(Mut)$, is eliminated.

4.4 Evaluation of The Approach

In this section several experiments are performed to assess the performance of the AG-ART architectures and to compare their performance with other ART and non-ART based classifiers. The AG-FAM architectures are compared with the GFAM architecture described in Chapter 3. Furthermore, the AG-ART collection are compared to other ART-based classifiers that have addressed the ART category proliferation problem; this comparison is thorough because we have coded and experimented with these other ART-

based classifiers on the same datasets used to assess AG-ART's performance. Finally, the AG-ART performance is compared to the performance attained by other non-ART based classifiers. In this section, 11 datasets are used in the comparison. These datasets are described in detail in Appendix B.

4.4.1 Comparing AG-FAM with GFAM

This section compares the AG-ART to GART, described in Chapter 3. The purpose of this comparison is to assess the value of the adaptive approach to choose the GA parameters, proposed in this chapter, to the static approach to choose these parameters, utilized in GART.

AG-ART introduced an adaptively defined, confidence factor, according to which ART categories are pruned, and an adaptively defined severity factor, according to which ART categories are mutated. In GFAM the probability of deleting an ART category was chosen after expensive experimentation and evaluation of the appropriateness of candidate probability values on a limited collection of classification problems; then these probability values were used for all other classification problems. Hence, the GART approach was not only computationally costly, but it was also not dataset-dependent, and it did not change throughout the evolutionary process. All these issues have now been addressed by the AG-ART approach, using the adaptively defined confidence factor, that is dataset-dependent and performance-based varying (relying on the performance of a category at each generation). Furthermore, in GFAM the severity of mutation was fixed,

and the only user defined parameter was the probability of mutation. This parameter was chosen in GFAM after expensive experimentation and evaluation of candidate probability values on a limited collection of classification problems; then these probabilities were used for all other classification problems. Hence, the GART approach for choosing the mutation probability was also computationally costly, and not dataset-dependent, and did not change throughout the evolutionary process. All these issues have now been addressed by the AG-ART approach, by using the adaptively defined severity of mutation factor, that is dataset-dependent and performance-based varying (relying on the performance of a category at each generation).

In summary, AG-ART's approach of choosing the category prune probabilities and the severity of mutation is much more elegant, sensible and cost-effective, compared to the approach used in GART. The important difference is that AG-ART, due to the adaptively chosen GA parameters, converges to this solution faster. Hence, the AG-ART approach is not only more elegant and more cost-effective in defining good values for the GA parameters, but even after the evolution starts AG-ART is more efficient than GART (converges to the final solution faster).

In the following the performance of the AG-FAM is experimentally compared to that of GFAM.

To demonstrate the point about the efficiency of the AG-FAM approach compared to the GFAM approach, Table 4.1 compares the average run times of GFAM, and AG-FAM discussed in this chapter. The total run time of AG-FAM and GFAM are defined as the total times, needed over a number of runs (different initial seeds), by the evolutionary

Table 4.1: Total run time for AG-FAM vs. GFAM, in seconds

Dataset	10 runs GFAM	10 runs AG-FAM	Gain
1Ci/Sq	387.578	150.78	61.10%
g4c-25	131.077	100.69	23.18%
g6c-15	231.64	171.14	26.12%
glass	4.439	2.81	36.63%
MOD-IRIS	32.157	26.61	17.25%
page	134.172	92.23	31.26%
pendigits	2684.204	2039.75	24.01%
pima	3.813	1.58	58.59%
sat	1033.171	679.17	34.26%
seg	151.234	79.86	47.19%
wav	288.516	238.77	17.24%

process in AG-FAM and GFAM to converge to a solution, respectively; in this case (see Table 4.1) AG-FAM and GFAM were run for 10 different initial seeds. Both approaches were able to find solutions of similar quality. However, the AG-FAM approach was able to reduce the evolutionary computation time up to 60% in some cases. Figure 4.2 illustrates how parameter adaptation allows the genetic algorithm to find better solutions more quickly. In review, with adaptation, the genetic algorithm is able to find same quality solutions using a smaller number of generations. The reduction in run time in AG-FAM is attributed to the use of the Prune operator, introduced in this chapter. The Prune operator is more efficient than the fixed probability of pruning used in GFAM. The Prune operator in AG-FAM contributes in finding smaller FAM networks faster in its evolutionary process than the corresponding operator in GFAM; since these smaller FAM networks are validated faster we end up with a reduced run time with AG-FAM, compared to GFAM.

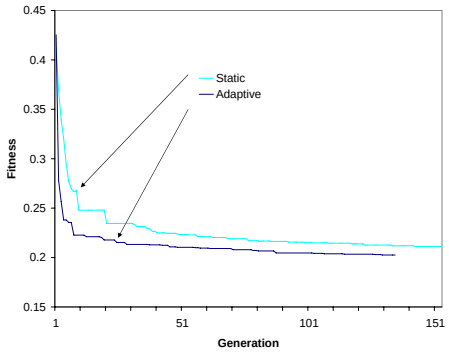


Figure 4.2: Fitness as a function of generation (Satellite dataset). The upper curve was produced using a non-adaptive approach. The lower curve was produced using adaptation. It is clear that adaptation allows faster progress of fitness towards optimum.

4.4.2 Comparison with Other ART Architectures

This section compares AG-ART's performance to that of other popular ART architectures, which have been proposed in the literature with the intent of addressing the category proliferation problem, such as ssFAM, ssEAM, and ssGAM. These approaches are based on the principle of semi-supervision, introduced by in [ABG03], and [VHG01].

The comparison is based on three measures of performance: generalization, size and computational cost. The results obtained from ssFAM, ssEAM, and ssGAM depend on the setting of the parameters of these networks. The choice of good settings for these parameters depends on the dataset at hand. Therefore to obtain good results from these networks one should experiment with a range of settings for the network parameters. Since the results obtained from AG-FAM, AG-EAM, and AG-GAM is a result of evolving (optimizing) a population of ART networks, it is appropriate to compare their performance to that obtained from the ssFAM, ssEAM and ssGAM experimentation performed to find their best parameter setting for any given database.

Since in this work we are not only focusing on generalization performance, but also on the size of the network produced, it becomes more complicated to compare and rank networks. To provide a fair comparison, we resort to a comparison approach that considers the two objectives simultaneously. Since the existence of the two, sometimes competing, objectives result in multiple good solutions rather than one "best" solution, in our comparison, we assess multiple solutions (sets of solutions) produced by the different classifiers, under consideration. In other words, for each classification algorithm, we produce a number of solutions that have attained the two objectives (good generalization and small size) at different levels of success. Then we choose the *non-dominated solutions*. A non-dominated solution is defined to be a network, where no other network from the list of found solutions dominates its performance, that is, achieves better generalization utilizing equal or smaller number of categories.

For each of the ssFAM, ssEAM, and ssGAM, and for each of the 11 databases, we performed a number of experiments with different settings of their network parameter values. In particular, the parameter settings that we experimented with ssFAM were: baseline vigilance values ranging from 0.1 to 0.9 with step size of 0.1, choice parameter values of 0.01 and 0.1, maximum allowable mixture threshold values ranging from 0 to 0.9 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 1,800 different parameter settings). Furthermore, the settings for ssEAM were: baseline vigilance values ranging from 0.1 to 0.9 with step size of 0.1, choice parameter values of 0.001 and 0.01, maximum allowable mixture threshold values ranging from 0 to 0.5 with step size of 0.1, minimum axes to maximum axis ratio values ranging

from 0.5 to 1 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 6,480 different parameter settings). Also, the settings for ssGAM were: baseline vigilance values ranging from 0 to 0.9 with step size of 0.1, initial standard deviation parameter ranging from 0.5 to 1 with step size of 0.1, maximum allowable mixture threshold values ranging from 0 to 0.9 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 6,000 different parameter settings). It should be emphasized that these parameter ranges reflect extensive experience of what are good parameter settings for these ART networks.

For the training of ssFAM, ssEAM, and ssGAM we used the same training set, and validation set as the one used for the AG-FAM, AG-EAM, and AG-GAM networks. We choose the solution networks proposed by each method based on the network size and performance of the network on the validation set. Different network solutions for the ssFAM, ssEAM and ssGAM network were produced by changing the parameter settings for these networks, as delineated in the previous paragraph. The total computation time required to obtain these network solutions for each database and each method, which is the sum of training and validation CPU times (in seconds) for all the tried settings, is reported in Table 4.3, and referred to as the *Total Run Time*.

Experiments were also conducted for AG-FAM, AG-EAM, and AG-GAM for each of the 11 databases. To obtain different solution networks, we varied the fitness parameter, α (see equation (3.8)). In particular, the different solutions for AG-FAM, AG-EAM and AG-GAM were obtained by considering the following α values: 0.01, 0.05, 0.1, 0.2 and 0.5. The total computation time needed to produce these solutions for AG-FAM, AG-

EAM, and AG-GA, is also referred to as the *Total Run Time*, and reported in 4.3, as well.

A one-to-one comparison of the results reported in Table 4.3 reveals that the *Total Run Time* of the AG-FAM, AG-EAM and AG-GAM networks is smaller, sometimes an order of magnitude smaller than the *Total Run Time* of their corresponding counterparts, ssFAM, ssEAM, and ssGAM, respectively. The *Total Run Time* results are also shown in a condensed, pictorial, fashion in Figures 4.3, 4.4 and 4.5.

To compare the generalization performance of AG-FAM and ssFAM, AG-EAM and ssEAM, and finally AG-GAM and ssGAM we use a metric that compares the network solutions obtained by the ss-network (for all different parameter settings) and the network solutions obtained by the AG-network (for the five different α values. This metric has been used before in similar situations (see [ZT99], [FES03], [FS05]). This metric is defined as follows:

$$C(A, B) = \frac{|b \in B : \exists a \in A, b \prec a|}{|B|} \tag{4.6}$$

This metric measures the fraction of members in set B that are dominated by at least one member in set A. Therefore, $C(A, B) = 1$ means all members in B are dominated by members in A. In this case the approach that produced set A is a clear winner. It is obvious that we need to consider also $C(B, A)$ in order to properly compare the two sets. Since the calculated values of $C(A, B)$ and $C(B, A)$ are dependent on the seed used to evolve the population of FAMs, EAMs and GAMs in the AG-ART approach, we produced network solutions for the five different α parameter values, by changing the seed 10 times. Con-

sequently, 10 different values of $C(AG-FAM, ssFAM)$, and $C(ssFAM, AG-FAM)$, were produced. Similarly, 10 different of $C(AG-EAM, ssEAM)$, and $C(ssEAM, AG-EAM)$, as well as of $C(AG-GAM, ssGAM)$, and $C(ssGAM, AG-GAM)$ were produced. In Table 4.2 we compare the average values (over the 10 replications) of $C(AG-FAM, ssFAM)$ versus $C(ssFAM, AG-FAM)$, and $C(AG-EAM, ssEAM)$, versus $C(ssEAM, AG-EAM)$, and $C(AG-GAM, ssGAM)$ versus $C(ssGAM, AG-GAM)$. It is obvious from the table that the average values of $C(AG-FAM, ssFAM)$ are larger than $C(ssFAM, AG-FAM)$ values, which indicates that networks produced by AG-FAM are more likely to dominate networks produced by ssFAM, and therefore, the networks produced by AG-FAM are expected to be of higher quality. The p-value column reported in the table corresponds to the t-test for the 2 sample means. The p-values in the table indicate that the difference in the means between $C(AG-FAM, ssFAM)$ and $C(ssFAM, AG-FAM)$ is statistically significant. Similar conclusions can be made by comparing the means of $C(AG-EAM, ssEAM)$ and $C(ssEAM, AG-EAM)$, as well as the means of and $C(AG-GAM, ssGAM)$ and $C(ssGAM, AG-GAM)$. The box plot shown below the $C(AG-FAM, ssFAM)$ and $C(ssFAM, AG-FAM)$ values of Table 4.2 compares visually the values of $C(AG-FAM, ssFAM)$ against the $C(ssFAM, AG-FAM)$ values for the segmentation dataset (one of the tested datasets). Furthermore, the box plot shown below the $C(AG-EAM, ssEAM)$ and $C(ssEAM, AG-EAM)$ values of Table 4.2 compares visually the values of $C(AG-EAM, ssEAM)$ against the $C(ssEAM, AG-EAM)$ values for the pendigits dataset (one of the tested datasets). Finally, the box plot shown below the $C(AG-GAM, ssGAM)$ and $C(ssGAM, AG-GAM)$ values of Table 4.2 compares visually the values of $C(AG-GAM, ssGAM)$ against the

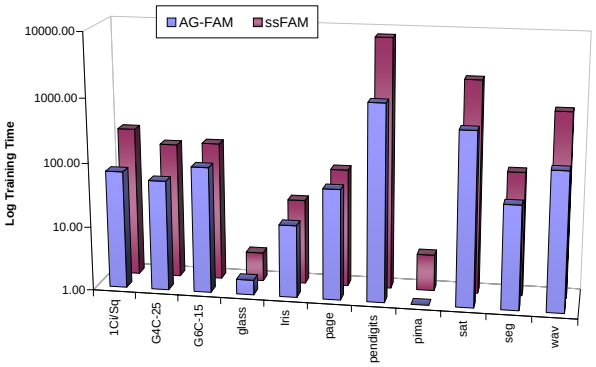


Figure 4.3: Total Run Time of AG-FAM vs. ssFAM

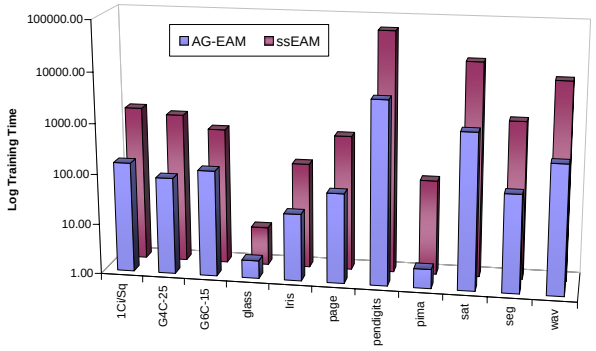


Figure 4.4: Total Run Time of AG-EAM vs. ssEAM

values of $C(ssGAM, AG-GAM)$ for the satellite dataset (one of the tested datasets). The box plots convey the same conclusions that the tabular entries in Table 4.2 convey.

Table 4.3 shows that the better performance of the AG-FAM, AG-EAM, and AG-GAM networks, is attained with reduced computations as compared with the computations needed by the alternate architectures (ssFAM, ssEAM, ssGAM). The computational advantage of genetically engineered ART networks compared to the semi-supervised ART architectures can be explained by the fact that the performance attained by ssFAM, ssEAM, and ssGAM required training these networks for a large number of network parameter settings (at least 1800 experiments) and then choosing the best networks through

Table 4.2: C-metric values. The p-value is based on t-test for the 10 values of the metric

	C(AG-FAM, ssFAM) Average	C(ssFAM, AG-FAM) Average	p-value	C(AG-EAM, ssEAM) Average	C(ssEAM, AG-EAM) Average	p-value	C(AG-GAM, ssGAM) Average	C(ssGAM AG-GAM) Average	p-value
1Ci-Sq	0.419	0.078	4.405E-06	0.892	0.000	1.004E-07	0.671	0.200	2.442E-04
G4C-25	0.900	0.050	3.888E-06	1.000	0.000	NA	1.000	0.000	NA
G6C-15	1.000	0.000	NA	1.000	0.000	NA	0.863	0.000	1.425E-13
glass	0.433	0.350	5.932E-01	0.878	0.075	1.276E-09	0.914	0.000	6.030E-09
Iris	0.533	0.300	1.647E-01	0.800	0.300	1.741E-02	0.857	0.000	NA
page	1.000	0.000	NA	0.900	0.000	2.807E-07	1.000	0.000	NA
pendigits	0.458	0.000	1.455E-09	0.422	0.175	1.216E-04	0.468	0.080	2.264E-06
pima	1.000	0.000	NA	0.860	0.100	1.187E-07	0.983	0.100	4.196E-09
sat	0.847	0.000	2.113E-09	0.722	0.000	2.346E-09	0.606	0.058	7.685E-09
seg	0.592	0.175	7.152E-05	0.572	0.243	4.131E-04	0.829	0.025	3.911E-13
wav	1.000	0.000	NA	1.000	0.000	NA	0.957	0.000	8.314E-12

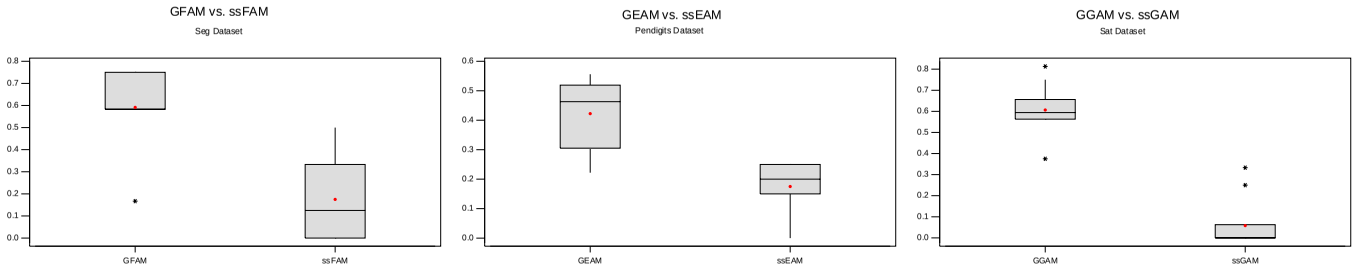


Table 4.3: Total run time for AG-FAM, AG-EAM and AG-GAM (10 replications) compared to total run time for ssFAM, ssEAM, ssGAM

Database Name	AG-FAM	ssFAM	Gain	AG-EAM	ssEAM	Gain	AG-GAM	ssGAM	Gain
1Ci/Sq	68.51	216.42	68.34%	152.13	1167.67	86.97%	142.29	1431.35	90.06%
G4C-25	52.59	130.92	59.83%	82.99	916.78	90.95%	89.32	314.49	71.60%
G6C-15	92.01	145.16	36.61%	127.15	508.22	74.98%	137.82	266.77	48.34%
glass	1.71	2.82	39.37%	2.26	5.72	60.45%	2.64	3.52	24.99%
Iris	13.48	21.30	36.73%	21.56	123.56	82.55%	22.02	109.25	79.84%
page	52.97	69.52	23.80%	60.91	484.15	87.42%	75.03	125.68	40.30%
pendigits	1142.43	7864.27	85.47%	4304.84	58865.05	92.69%	537.62	20050.39	97.32%
pima	0.95	3.68	74.21%	2.41	75.88	96.82%	1.91	32.75	94.15%
sat	508.21	2034.29	75.02%	1234.62	17162.45	92.81%	329.07	4302.16	92.35%
seg	41.93	85.55	50.99%	88.45	1331.47	93.36%	64.13	509.99	87.43%
wav	147.23	763.99	80.73%	369.38	8612.65	95.71%	83.94	1199.58	93.00%

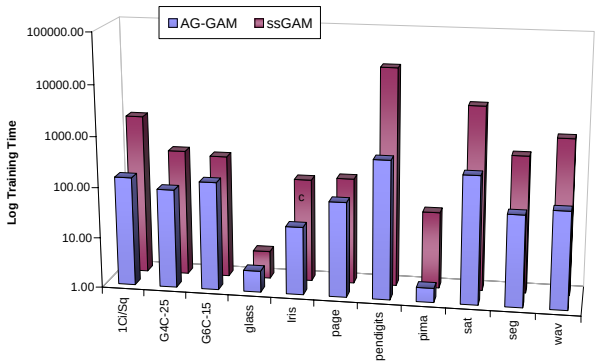


Figure 4.5: Total Run Time of AG-GAM vs. ssGAM

cross-validation. In the AG-FAM, AG-EAM and AG-GAM cases we trained only a small number of these networks ($Pop_{size} = 20$ of them). Furthermore, in AG-FAM, AG-EAM and AG-GAM cases we evolved the trained networks for at most $Gen_{max} = 500$ generations, each evolution requiring cross-validating only the $Pop_{size} = 20$ networks. Quite often, the evolutionary process converged after only a few (50) generations, because a satisfactory solution was found.

The accuracy and size advantage of AG-FAM, AG-EAM, AG-GAM compared to ssFAM, ssEAM and ssGAM can be attributed to the genetic optimization that it employs. This optimization, involving the *Prune* and *Crossover* operators, allow one to construct networks that are not attainable using the original ART training rules. Also, the operation of *Prune* and *Mutation* operators were designed to guide the genetic search to optimal solution faster, resulting in the significant computational advantages of AG-FAM, AG-EAM, and AG-GAM compared to the semi-supervised ART architectures.

Table 4.4: Performance of AG-FAM, AG-EAM and AG-GAM for 11 datasets

Dataset	AG-FAM		AG-EAM		AG-GAM	
Name	PCC	size	PCC	size	PCC	size
1Ci/Sq	98.07	31	99.70	2	99.83	2
G4C-25	74.94	4	75.14	4	75.24	4
G6C-15	84.75	6	85.01	6	84.97	6
glass	76.56	6	75.00	6	73.44	9
Iris	94.96	2	95.04	2	94.75	2
page	96.59	5	95.09	5	96.34	6
pendigits	98.20	282	98.31	331	97.83	108
pima	79.31	2	78.88	3	77.16	2
sat	88.90	310	87.85	203	88.35	118
seg	95.86	15	93.71	128	92.71	17
wav	85.90	4	87.15	4	87.50	3

4.4.3 Comparison with Other Published Results

This section compares results obtained using the proposed AG-ART architectures to literature results published by other authors using various classification algorithms. The comparison was based on well known datasets that many researchers chose to test their algorithm against. The focus is mainly on the classification accuracy as it is the case with most published work. The proposed approach is tested with a fitness function set to maintain high level of accuracy in the classifier. Table 4.4 lists the results obtained.

In [KBS97] the authors use the simple Bayes classifier to produce classification accuracy of 85.20% on satellite, 93.12% on segmentation, 78.57% on waveform, 70.11% on glass and 75.9% on pima. In [YA01], the authors use a decision tree variant to produce classification accuracy of 92% (size: 37) on segmentation, 83% (size: 65) on waveform, 60% (size 14) on glass, 75.2% (size: 3.4) on pima and 95% (size: 37) on pendigits. As it is evident from Table 4.4 our AG-ART results consistently outperform these results.

In [LLS00], the authors compared the accuracy and size of a 33 classifiers belonging to the tree, statistical and neural types classifiers. Three of the datasets that Lim, Loh and Shih have experimented with are the Satellite, the Segmentation and the Waveform datasets that has been tested in Table 4.4. The AG-FAM results on the Satellite dataset are: 88.9 classification accuracy, needing 310 categories (other AG-FAM solutions include: 83.6% with 6 categories, 84.5% with 14 categories). The AG-GAM results on the Satellite dataset are 88.35 with 118 categories. The accuracy results reported on the Satellite dataset by [LLS00] are: Minimum classification accuracy of 60% and maximum classification accuracy of 90%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 8, while the median tree size was 63. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 63 and 216. The AG-FAM results on the Segmentation dataset are: 95.86% classification accuracy, needing 15 categories. The accuracy results reported on the Segmentation dataset by [LLS00] are: Minimum classification accuracy of 48% and maximum classification accuracy of 98% (achieved by the nearest neighbor classifier, which performs no data compression). Furthermore the tree type classifiers (22 of them) created a minimum tree size of 6, while the median tree size was 39. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 69 and 42. The AG-FAM results on the Waveform dataset are: 85.9% classification accuracy, needing 4 categories for AG-FAM and 87.5% and 3 categories for AG-GAM. The accuracy results reported on the Waveform dataset by [LLS00] are: Minimum classification accuracy of 52% and maximum classification accuracy of 85%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 3, while the median tree size was 16. Finally, two

of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 14 and 54.

4.5 Discussion

In the previous section it was shown that the use of adaptation results in a the genetic algorithm that is more efficient in finding optimal solutions. The claim here is that the genetic operators used are more capable of finding solutions more quickly.

This section looks more carefully at the genetic operators and tries to quantify the effect of adaptation on the performance of these operators. For this purpose, metrics are defined to quantify the performance of these operators. The following metrics are defined:

1. **Operator Success Ratio**: This is defined as the ratio of successful applications of the operator to each individual over successive generations in isolation of other operations.

2. **Operator No-Fail Ratio**: This is defined as the ratio of applications of the genetic operator that did not cause deterioration in the fitness of the individual.

It is obvious that the higher the Operator success ratio the more efficient is the GA. However, if the value is too high, the search is equivalent to a hill-climbing search that is likely to end up at a local optima. On the other hand, a low value indicates that the GA

is not utilizing the information gained from testing solutions in every generation, and it becomes similar to a random search.

Table 4.5 shows the results of testing the two genetic operators: Mutate and Prune. The results were obtained by measuring the fitness of the solution (network) before and after each operator is applied. The success rate accounts for the percentage of instances where the application of the operator caused an improvement in fitness. The no-fail rate accounts for the percentage of instances where the application of the operator did not cause a decrease in the fitness of the individual. Therefore, higher values of the two metrics are desirable. Table 4.5 compares the performance of three different implementations of the genetic operators. The first is the adaptive mechansim proposed in this chapter. The second is the use of static probabilities as described in Chapter 3. In that chapter it was found that the optimal values for the probability of deleting a category, $Pr(Cat_{del})$, and the probability of mutating a category, $Pr(Mut)$, were 0.1 and 0.4 respectively. The third case uses the settings proposed in [Al 06] where $Pr(Cat_{del}) = 0.1$ and $Pr(Mut) = \frac{5}{numCats}$, where $numCats$ is the number of categories present in the network. It should be noted that the third cases does not use the implementation proposed in [Al 06]; it uses the implementation in Chapter 3 with only the mutation operator implemented as in [Al 06]. Also it should be noted that the experimental setup is different than that used in the previous section. For example, the algorithms are set to collect a number of metrics which adds a significant overhead to the run time. Table 4.5 also records the average size, which is the average size of the network in the population in each generation. The numbers reported in Table 4.5 are a result of running 5 replications

Table 4.5: Observed metric values for genetic operators. The reported values are averages over five replications.

	Prune Success	Mutate Success	No Fail Prune	No Fail Mutate	Avg. Size	Run Time	Num. Gens
(Mut as in [Al 06])	5.91%	2.39%	98.18%	86.25%	9.96	51.07	187.60
$P_{mut} = 0.4$, $P_{del} = 0.1$	5.38%	16.79%	97.98%	33.34%	9.42	47.77	182.00
Adaptive	31.02%	12.84%	86.06%	54.45%	6.75	31.16	162.60

using the Gaussian dataset, G4C-25 (see Appendix B for more details). From Table 4.5, one can make the following observations:

Going from the implementation proposed in [Al 06] to the one proposed in Chapter 3, it can be seen that the Prune (delete) operator performance was not significantly affected. This is expected since this operator was not changed. On the other hand, the mutation operator success ratio has significantly improved. However, the no-fail ratio has significantly decreased. The overall training time has improved slightly.

For the adaptive implementation, it is easy to observe that there is a significant improvement in the performance of the Prune operator. This has been achieved with only a small sacrifice in the no-fail ratio. The mutation operator success rate is improved over the first case, but worse than the second case. However, this improvement did not require as a large sacrifice in the no-fail ratio. The adaptive mechanism was able to significantly reduce the overall run time. The overall run time of the evolved ART architectures can be expressed as follows,

$$T_{total} = T_{training} + T_{validation} + T_{alg} \tag{4.7}$$

The training time, $T_{training}$, is time required to train the initial population (using ART training rules). The component designated as T_{alg} refers to the overall housekeeping time for the algorithm. The validation time, $T_{validation}$, is the total time required to estimate the classification error (on a validation set) of ART networks in the population, in successive generations, until convergence. The validation time, $T_{validation}$ is dependent on the number of generations until convergence, G_{conv}, population size, λ, number of validation examples, n_v, average number of categories in each ART network, $\overline{size(x)}$ (taken over all generations and all networks in the population), and the time it takes to calculate the output signal (category match function and category choice function) for a given category and a given validation pattern , t_{CMF};

$$T_{validation} = G_{conv} * \lambda * n_v * \overline{size(x)} * t_{CMF} \tag{4.8}$$

Table 4.5 shows that the adaptive mechanism was able to significantly reduce the average size, $\overline{size(x)}$. Also, the adaptation was also able to reduce the number of generation needed to reach convergence, G_{conv} by more efficiently guiding the search. Reducing $\overline{size(x)}$ and G_{conv} reduces $T_{validation}$ and therefore reduces T_{total}.

Figure 4.6 shows the progress of the fitness of the best performing network as a function of generation number. It can be easily seen that the adaptation causes faster progress towards the optimum solution.

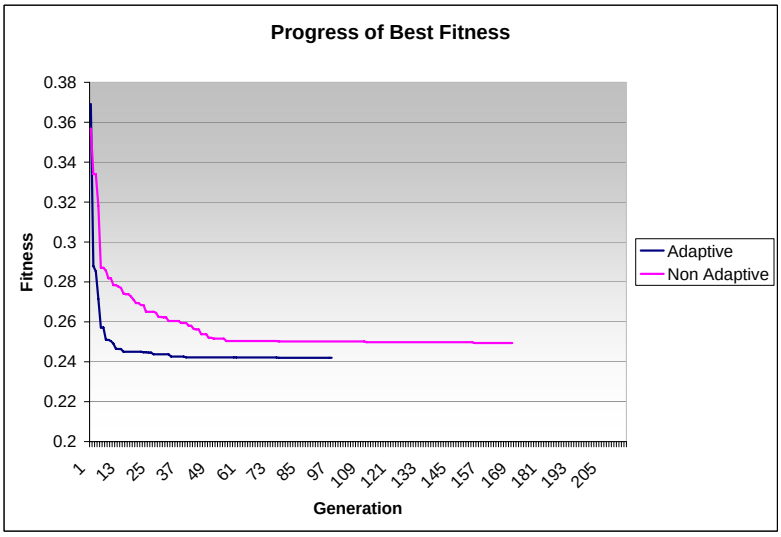


Figure 4.6: Fitness as a function of generation (Gaussian dataset (G4C-25)).

4.6 Summary

This chapter introduced an improved, compared to its predecessor GART, genetically engineered ARTMAP neural network referred to as AG-ART. Experimental results have shown that AG-ART is at least as accurate and creates as small of an architecture as GART, and it does so at reduced computational cost. While in GART the probability of deleting an ART category and the probability of mutating a category were chosen after experimentation on a limited collection of classification problems, in AG-ART an adaptive mechanism was implemented to choose these parameters based on the database at hand and the quality of the solution found. The AG-ART approach is not only more elegant and more cost-effective in defining good values for the GA parameters, but after the evolution starts AG-ART was found to be more efficient than GART.

This chapter have also presented an extensive comparison between the AG-ART architectures and semi-supervised ART architecture (ssFAM, ssEAM, ssGAM); these semi-supervised ART architectures are architectures that perform very favorably compared to other ART architectures, and quite often compared to other classification approaches. This comparison took into consideration the classification accuracy and size of the classifier at the same time, and it was fair because these semi-supervised ART architectures were coded and tested on the same datasets as the AG-ART architectures were tested. The experiments reveal clearly that the AG-ART architectures are able to produce "better quality" classifiers than the ssART architectures, at a reduced computational cost. The computational cost reduction was found, in a number of instances, to be significant (more than an order of magnitude). Furthermore, the performance of AG-ART classifiers was compared with the performance of other classifiers (non-ART based classifiers) that have appeared in the literature; this comparison showed that AG-ART classifiers are very competitive in terms of accuracy and size of a classifier that they produce.

In summary, this chapter introduced a new family of ART-based architectures, called AG-ART, using an elegant evolutionary approach. The introduced architectures are able to produce classifiers of good accuracy and small size, using a reasonable computational budget. They also have the advantage of requiring little user intervention because the algorithm parameters are automatically adapted.

CHAPTER 5
EVALUATION RELAXATION

The architectures introduced in Chapter 4 were shown to achieve competitive generalization and exceptionally small size. In addition, the genetic ART architectures described in that chapter have the advantage of alleviating the need for tweaking algorithm parameters; a well known issue with many other ART and non-ART classifiers. A major concern regarding these architectures, and any evolved neural network architecture in general, is the added overhead in terms of computational time needed to produce the finally evolved network. This chapter investigates ways of reducing this computational overhead by reducing the computations needed for the calculation of the fitness value of the evolved ART architectures. The results obtained in this chapter can be directly extended to many other evolutionary neural network architectures, beyond the genetic ART neural network architectures.

5.1 Introduction

Genetic ART starts with a population of trained ART networks, whose number of nodes in the hidden layer and the values of the interconnection weights converging to these nodes are fully determined (at the beginning of the evolution) by ART's training rules. To this initial population of networks, GA operators are applied to modify these trained

networks (i.e., number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures. The pseudo-code is listed in Figure 4.1 and as a reminder in Figure 5.1 below.

$P(0) \leftarrow$ Generate-Initial-Population();
for $t \leftarrow 1$ **to** Gen_{max} **do**
    Evaluation();
    if *stopping criteria met* **then** exit for;
    $P'(t) \leftarrow$ Selection($P(t)$);
    $P(t) \leftarrow$ Reproduction($P'(t)$);
end
return *Best Network in* $P(t)$;

Figure 5.1: Pseudo-code of the AG-ART algorithm

During the *Evaluation* step, the genetic algorithm estimates the performance (prediction error) of a solution (ART network) by measuring the classification error rate on a validation set. It was found that the majority of the CPU time spent by the genetic algorithm, is used for measuring the classification error (more than 80%) as shown in Table 5.1. Therefore, techniques that can reduce this time component have the potential of reducing the overall convergence time of the GA, when used to evolve ARTMAP NNs. Such improvement can be effective to the viability of many evolved NN architectures.

Table 5.1 shows the allocation of CPU time for 10 replications (each row is an identical replication using a different random number seed of the evolutionary process) for running the Genetic ARTMAP algorithm using a sample database. It can be observed that the majority of the time is spent during the validation of the solutions (members of the GA population).

The overall run time of the evolved ART architectures can be expressed as follows,

Table 5.1: Allocation of CPU time.

Total	Training Time		Validation Time		Algorithm Time	
Time	Seconds	%	Seconds	%	Seconds	%
19.22	2.63	13.66	16.53	86.02	0.06	0.32
25.61	2.55	9.95	23.05	89.99	0.02	0.06
119.36	2.55	2.13	116.52	97.62	0.30	0.25
30.08	2.59	8.62	27.34	90.91	0.14	0.47
27.88	2.64	9.47	25.08	89.97	0.16	0.56
21.44	2.50	11.66	18.78	87.62	0.16	0.72
20.50	2.56	12.50	17.86	87.12	0.08	0.38
58.69	2.56	4.37	55.90	95.25	0.22	0.38
111.08	2.52	2.27	108.26	97.47	0.30	0.27
22.88	2.55	11.13	20.28	88.67	0.05	0.20
26.14	2.42	9.27	23.67	90.56	0.05	0.18

$$T_{total} = T_{training} + T_{validation} + T_{alg} \tag{5.1}$$

The training time, $T_{training}$, is time required to train the initial population (using ART training rules). The component designated as T_{alg} refers to the overall housekeeping time for the algorithm. The validation time, $T_{validation}$, is the total time required to estimate the classification error (on a validation set) of ART networks in the population, in successive generations, until convergence. The validation time, $T_{validation}$ is dependent on the number of generations until convergence, G_{conv}, population size, λ, number of validation examples, n_v, average number of categories in each ART network, $\overline{size(x)}$ (taken over all generations and all networks in the population), and the time it takes to calculate the output signal (category match function and category choice function) for a given category and a given validation pattern , t_{CMF};

$$T_{validation} = G_{conv} * \lambda * n_v * \overline{size(x)} * t_{CMF} \tag{5.2}$$

This chapter investigates reducing the validation time by reducing the number of validation points, n_v. Genetic algorithms are known for their suitability in problems where the fitness evaluation is not reliable or noisy. The noise in estimating objectives (such as classification error) has the effect of causing noisy selection. This in turn might slow down the convergence or convergence to a sub-optimal solution. The noise in the estimation of classification error might be hard to reduce, since we are limited by the data availability. However, when a large amount of data is available for cross-validation, it becomes time consuming to obtain an estimate for classification error using all the data. Therefore, evaluating the classification error for a given network in a given generation using a randomly sampled subset of the validation data might be beneficial in reducing the convergence time.

Reducing the number of validation points used in estimating the classification error of the produced ARTMAP solutions leads to more noisy evaluations of the GA fitness function. However, as suggested by [FG88], in some cases, the use of fitness estimates with higher variance could actually increase the quality of solution obtained from the GA for a fixed computational budget. If time is saved by making faster, but noisier, estimates of the fitness function, then this time can be effectively used to converge to a better solution. To make a good judgement about this approach, one must understand the operation of GA, and how the variance of the estimate of the fitness function affects the progress rate of finding optimal solutions. The effect of noise in GAs has been studied by a number of researchers ([Mil97], [GDC92], [FG88], etc). Their results provide insight

to convergence properties under noisy estimation of the fitness function. In [GDC92] the authors present guidelines for choosing the population size.

In [Jin05] the author presents a comprehensive survey of fitness function approximation in evolutionary algorithms. Fitness approximation, also refereed to as evaluation relaxation, is applied when the fitness function is computationally expensive, noisy, or difficult to define and evaluate. The author identifies three levels of approximation and refers to them as problem approximation, functional approximation and evolutionary approximation. In problem approximation, which is the approach adopted in this chapter, the original fitness function is replaced by a less expensive but less accurate one such as in [FG88]. In function approximation, an expression is constructed to approximate the evaluation of the original fitness function. Evolutionary approximation are methods that are specific to evolutionary algorithms such as fitness inheritance [SDS95], where the fitness value of the offspring is estimated from that of their parents. This approach relies on surrogate functions, which are used to construct a relationship between offspring and parent fitness values. In [SLG06] the authors propose surrogate functions that automatically adapt to the problem structure where the structural form of the surrogate is inferred using a probabilistic model and the coefficients of the surrogate are estimated using a least squares method.

This chapter investigates the use of a less expensive fitness function by means of sampling. The sample size is controlled throughout the evolution in such a way to keep a certain level of confidence in the fitness value. This method significantly reduces the convergence time without sacrificing performance as will be shown in the experimental

results. The organization of the chapter is as follows: In Section 5.2 we investigate the effect of sampling the validation set and provide an estimate for the number of validation points that could be used in the evolution of ART neural networks. In section 5.3 we justify, experimentally, that this estimate leads us to computational savings for the evolutionary process needed to produce AG-FAM, AG-EAM and AG-GAM. Finally, in Section 5.4 we summarize our findings.

5.2 Sampling the Validation Set

This section investigates the possibility of using a less expensive fitness function by means of sampling the validation data. This results in a fitness function that is faster to evaluate, but is less accurate. When the evaluation of solutions is not reliable, the genetic algorithm may suffer from selection error. The result of selection error is reduction in the efficiency of the genetic algorithm and the possibility of reduction in the quality of solutions returned. Selection error happens because the estimation of the objective function is not accurate. The inaccuracy in the objective function estimation results in incorrect assignment of fitness for solutions. Therefore, the selection operator proceeds in a manner that is different than the way it would have if the objective function evaluations were accurate.

To investigate the use of this approach to the evolution of ART architectures, a simple experiment was conducted. We used a simulated dataset (Circle in Square with probability of points in the circle is 30%). Since we know the optimal classifier, we set

Table 5.2: Comparing the number of generation and CPU time for different sizes of the validation set sample. Average taken over 10 replications.

	Avg #Gens	Avg Training Time
All	97.4	42.0174
All/2	166.2	26.4689
All/4	296.4	23.4765
All/16	326.7	7.1521

the stopping criteria of the genetic algorithm to be when the optimal solution is found. We ran 10 replications of each experiment. Each cross-validation is performed by taking a sample from the original validation set. In each experiment we varied the number of points used in the cross-validation set. Table 5.2 shows the results. It can be seen that reducing the number of validation points increases the number of generations the GA has to run in order to find the optimal solution. When all the validation points were used, the algorithm took about 42 seconds to find the optimal network. However, when the validation samples were reduced to 1/16 of the original validation set, the training time is expected to drop to about to $42/16 = 2.6$ seconds. However, the GA requires about 3 times more generations to find the optimal solution when a small validation set is used. The number of generation tripled. Therefore we adjust our estimate to $2.6 * 3 = 7.9$ seconds. The results obtained shows that the actual training time is 7.2 seconds which is approximately consistent with our prediction.

These results allow us to conclude that time can be saved by using a smaller number of validation samples and allow the GA to run for more generations. However, sampling reduces the accuracy of the estimate of the objective function. Therefore, using a very small sample size might reduce the ability of the genetic algorithm to find the optimal solution quickly due to excessive selection error. To investigate this effect, we conduct

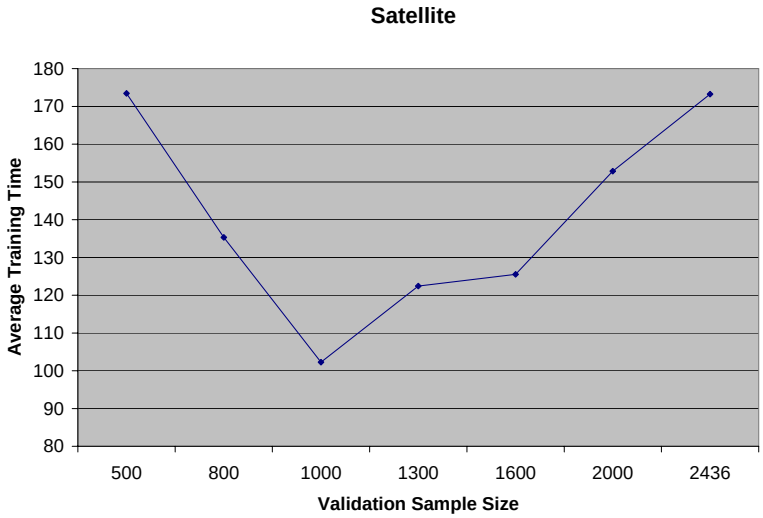


Figure 5.2: Training time vs. validation sample size for the Satellite dataset

experiments on two real datasets: Pendigits and Satellite (More details about these datasets can be found in Appendix B). We experimented with different sizes of the validation sample. We ran 10 replication of each experiment.

Figures 5.2 and 5.3 shows the results of these experiments. It should be noted that all experiments converged to similar solution quality. It can be observed that reducing the validation sample size produced overall savings in the training time up to a certain limit, after which, the training time starts to increase again. This can be explained by the fact that very small sample sizes result in estimates of the objective function that are not reliable. Although genetic algorithm can operate effectively in noisy environments, the existence of a large variance in the estimate of the objective function (the error rate of an ART network) results in reducing the efficiency of the genetic algorithm in finding optimal solutions. The variance of the estimate of the error rate on a validation set can be given by the following equation (see [MS95], page 388):

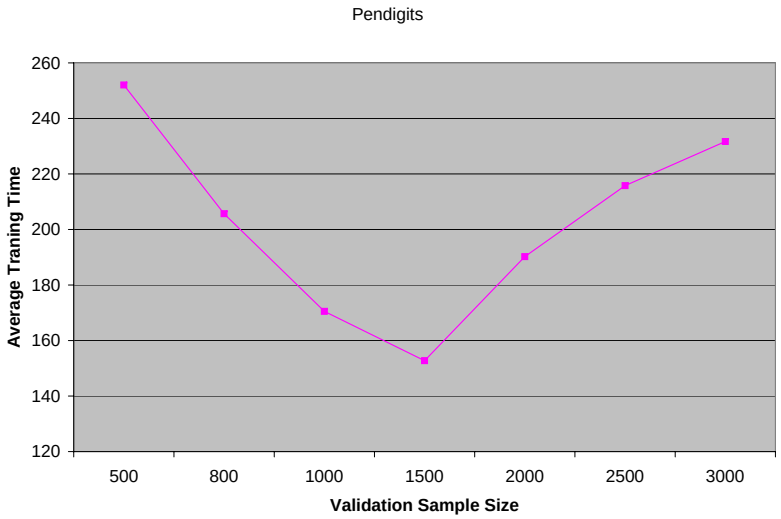


Figure 5.3: Training time vs. validation sample size for the Pendigits dataset

$$Var(\hat{p}_{err}) = \frac{p_{err}(1 - p_{err})}{n_v} \tag{5.3}$$

To avoid making unreliable estimates of the error rate, we propose to dynamically vary the validation sample based on the error rate achieved by the genetic algorithm. Our approach to determine an appropriate value for n_v is to set a desired accuracy in estimating the classification error as measured by the half-width (for example $\pm 1\%$) of the confidence interval at a desired confidence level (for example 99%). This can be estimated using equation (5.4):

$$n_v = (\frac{z_{\alpha/2}}{Halfwidth})^2 \hat{p}_{err}(1 - \hat{p}_{err}) \tag{5.4}$$

5.3 Evaluation of The Approach

In this section we present an experimental demonstration of the ideas outlined in this paper. In evolving neural networks, the error rate is the only (or the main) objective to be optimized. Without loss of generality, evolution of ART networks is taken as example. ART networks are evolved by repeatedly applying genetic operators to a population of ART networks. In every generation the selection process determines the probability of survival and breeding of an individual (network) based on its performance. Similar to a number of evolved neural network architectures, the performance is measured using a fitness function that is a linear combination of the error rate measured on the validation set and the network complexity measured in terms of the number of hidden nodes present in that network. The claim presented in this chapter is that making a faster but noisier estimation of the validation error might eventually achieve similar solution quality at reduced computational cost. The fast estimation of classification error is made by randomly sampling a subset of the available validation samples each time we need to estimate the classification error for a given network.

The experiments presented in this section compare the genetically engineered ART architectures introduced in the previous chapter, which use all the validation patterns in every evaluation, and the same genetically engineered ART architectures that sample the validation set in every evaluation, as designated by equation (5.4). The genetically engineered ART architectures introduced in the previous chapter include three architectures: Adaptive Genetic Fuzzy ARTMAP (AG-FAM), Adaptive Genetic Ellipsoidal ARTMAP (AG-EAM) and Adaptive Genetic Gaussian ARTMAP (AG-GAM). The re-

sults pertaining to AG-ART without sampling the validation patterns are referred as "AG-FAM Old", "AG-EAM Old", and "AG-GAM Old", while the results using the sampling according to equation (5.4) are referred to as "AG-FAM New", "AG-EAM New", and "AG-GAM New". We have experimented with 9 databases, and more information about these datasets can be found in Appendix B.

For each of the 9 databases, we ran the "old" and "new" architectures 10 times for 10 different initial seeds of the GA optimization process. In Table 5.3 we compare the average time (over the 10 replications) required for convergence for the "old" and "new" architectures. It can be observed from Table 5.3 that the time saved in the "new" implementation can reach up to 80% of time that the "old" took to converge, justifying the merit of the proposed technique. We also record the best performance achieved by both these implementations in Table 5.4. In this table it can be seen that both implementations were able to converge to similar quality of solutions. Therefore, the savings in time did *not* require a sacrifice in the quality of the solutions achieved.

5.4 Summary

This chapter introduces a technique that can be used to significantly improve the efficiency of many evolved neural network architectures. The technique proposed capitalizes on the ability of genetic algorithm to operate effectively in noisy environments. It was shown experimentally that relying on faster, but noisier, estimation of classification error during the evolution of neural networks might be beneficial to the overall computational

Table 5.3: Comparing the Training Time of genetic ART with and without sampling of the cross-validation set.($\alpha = 0.01$, take best of 10 reps)

	AG-FAM			AG-EAM			AG-GAM		
	Old	New	% Time Saved	Old	New	% Time Saved	Old	New	% Time Saved
1Ci/Sq	68.51	22.12	67.72%	152.13	68.15	55.20%	142.29	28.31	80.10%
G4C-25	52.59	23.33	55.63%	82.99	47.21	43.11%	89.32	32.49	63.62%
G6C-15	92.01	27.49	70.12%	127.15	43.44	65.83%	137.82	34.67	74.84%
Iris	13.48	3.47	74.22%	21.56	7.51	65.18%	22.02	4.65	78.89%
page	52.97	13.36	74.77%	60.91	36.23	40.52%	75.03	20.78	72.30%
pendigits	1142.43	874.87	23.42%	4304.84	3923.03	8.87%	537.62	311.02	42.15%
sat	508.21	236.86	53.39%	1234.62	1037.92	15.93%	329.07	192.67	41.45%
seg	41.93	29.82	28.89%	88.45	81.31	8.07%	64.13	63.32	1.26%
wav	147.23	112.85	23.35%	369.38	338.54	8.35%	83.94	56.67	32.48%

Table 5.4: Comparing the performance of genetic ART with and without sampling of the cross-validation set

	Old						New					
	AG-FAM		AG-EAM		AG-GAM		AG-FAM		AG-EAM		AG-GAM	
	PCC	size	PCC	size	PCC	size	PCC	size	PCC	size	PCC	size
1Ci/Sq	98.07	31	99.70	2	99.83	2	97.67	31	99.27	2	98.93	2
G4C-25	74.94	4	75.14	4	75.24	4	74.98	4	74.96	4	75.22	4
G6C-15	84.75	6	85.01	6	84.97	6	84.85	6	85.09	6	85.11	6
Iris	94.96	2	95.04	2	94.75	2	95.08	2	95.04	2	94.94	2
page	96.59	5	95.09	5	96.34	6	96.56	5	95.30	5	95.56	5
pendigits	98.20	282	98.31	331	97.83	108	97.86	276	96.97	175	97.86	129
sat	88.90	310	87.85	203	88.35	118	88.40	184	87.30	173	88.15	127
seg	94.86	22	93.71	128	92.71	17	95.86	35	94.43	129	91.57	13
wav	85.90	4	87.15	4	87.50	3	84.05	4	86.60	8	87.65	4

cost of evolving neural network architectures. Some of the time saved by making fast evaluations of the classification error of the evolved neural networks is used to allow the evolutionary process to reach the desired level of solution quality faster, despite the fact that more generations might be needed to achieve this goal.

The merit of the proposed technique was illustrated using genetic ART neural network architectures. It was shown, using the technique proposed in this chapter, that significant amount of computational time (as much as 80% in some cases) can be saved if the GA relies on noisy calculations of the classification error but allow the GA process to evolve over a higher number of generations. Also, the results demonstrated that this improvement in efficiency did not affect the quality (accuracy and size) of the classifier network produced. The claim in this chapter though is that these beneficial results would extend to other classification problems (beyond the ones we experimented with in this chapter) and to other neural network architectures (beyond the ART neural networks considered in this chapter).

CHAPTER 6
MULTIOBJECTIVE OPTIMIZATION OF ARTMAP ARCHITECTURE

This chapter presents, the evolution of ART Neural Network architectures (classifiers) using a multiobjective optimization approach. In particular, the use of a multiobjective evolutionary approach is proposed to evolve simultaneously the weights, as well as the topology of three ART architectures; Fuzzy ARTMAP (FAM), Ellipsoidal ARTMAP (EAM) and Gaussian ARTMAP (GAM). The resulting architectures are referred to as MO-GFAM, MO-GEAM, or MO-GGAM, and collectively as MO-GART. The major advantage of MO-GART is that it produces a number of solutions for the classification problem at hand that have different levels of merit (accuracy on unseen data (generalization) and size (number of categories created)). MO-GART is shown to be more elegant (does not require user intervention to define the network parameters), more effective (of better accuracy and smaller size), and more efficient (faster to produce the solution networks) than other ART neural network architectures that have appeared in the literature.

6.1 Introduction

AG-ART, described in Chapter 4, starts with a population of trained ART networks, whose number of nodes in the hidden layer and the values of the interconnection weights converging to these nodes are fully determined (at the beginning of the evolution) by ART's training rules. To this initial population of ART networks, GA operators are applied to modify these trained ART architectures (i.e., number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures. The optimization problem set up in AG-ART has two objectives: maximize classification accuracy on a validation set, and minimize network complexity (size of the network), measured in terms of the number of hidden nodes (categories). In AG-ART, these two objectives were combined using a weighted sum fitness function. A problem with this approach is that the user has to a-priori specify their preference of accuracy and complexity, by choosing the weights in this fitness function. However, choosing good weights for the fitness function is a data dependent problem. To overcome this, the user should run the algorithm for different settings of the weights in the fitness function; an expensive proposition. Furthermore, the weighted sum approach might *not* be able to reproduce all possible solutions that might be of interest to the user (more details later in the chapter). Since genetic algorithms are population-based approaches, they are suitable in finding multiple solutions if an appropriate multiobjective evolutionary algorithm (MOEA) is used.

The organization of the chapter is as follows: In the next section a review of the different MOEA's introduced in the literature is presented. In Section 6.4 the proposed

MO-GART algorithm is discussed. In Section 6.5 the performance of MO-GART is evaluated and compared with AG-ART and three other ART architectures: ssFAM, ssEAM and ssGAM (see [ABG03]). Finally, Section 6.6, summarizes this chapter's findings.

6.2 Multiobjective Evolutionary Algorithms

Many real world problems involve simultaneous optimization of conflicting objectives. This is the basic challenge of multiobjective optimization research. Evolutionary algorithms have been used extensively to solve multiobjective optimization problems, resulting in a body of knowledge known as multiobjective evolutionary algorithms (MOEA). This discipline resulted from the marriage of two disciplines: evolutionary computation and multi-criteria decision making. A number of authors have published surveys of MOEA, such as [Coe00], [Coe06], and the reader can find more details about MOEA's there.

With conflicting multiple objectives, there is no single optimal solution, but rather, there are a set of good solutions with varying degrees of merit. It is often desirable to find these good solutions as they provide alternative solutions to the problem at hand. As the desired solution may not be clear before hand, the availability of what is achievable allows the decision maker to choose appropriate solution to the problem after he/she gets a chance to review the list of available solutions and their respective merits. Also, when design constraints are changed and one solution becomes infeasible these other optimal solutions provide handy alternatives. For example, when one design may be too expensive

to implement, other lower cost designs become attractive alternatives. Evolutionary algorithms (EAs) are suitable for solving multiobjective optimization problems because EAs are population based search algorithm, and as such they can find, in a single run, multiple good solutions on the surface defined by the multiple objectives that are to be optimized.

Formally, the the multiobjective optimization problem can be stated as follows:

Optimize the vector function $f(x)$ of L objectives, by finding solution x^*, where:

$$f(x^*) = [f_1(x^*), f_2(x^*), ..., f_L(x^*)]^T \tag{6.1}$$

and x^* is a vector representing the input decision variables for the problem, and:

$$x^* \in F \tag{6.2}$$

where F is the feasible region in the solution space. Therefore, we want to find a solution in the feasible space that results in values of the vector function f that are acceptable to the user. The set of functions $f_1(x), f_2(x), ...$ are usually of conflicting nature. In other words, it is very rare to find a single solution that optimizes all the functions simultaneously. Therefore, several solutions may exit that optimize one or more objectives, or provide a unique tradeoff between the objectives. The minimum set of such optimal solutions is called the set of nondominated solutions, or the Pareto-optimal set. A solution is considered Pareto-optimal if there exists no other feasible solution which would decrease

some objective without causing a simultaneous increase in at least one other objective (assuming that we are trying to minimize the objectives).

Formally, a solution $x^* \in F$ is said to be nondominated if there exist no other solution $x \in F$ such that,

$$\forall i : f_i(x) \leq f_i(x^*), i = 1, 2, ...L, and, \exists i : f_i(x) < f_i(x^*) \tag{6.3}$$

A Pareto-optimal solution is the solution that is not dominated by any other solution in the search space. The entire set of such optimal tradeoff solutions is often referred to as the Pareto front.

The main focus in MOEA research is to minimize the distance of the generated solutions to the true Pareto set and to maximize the diversity of the discovered Pareto set. A good Pareto set may be obtained by appropriate guiding of the search process through careful design of selection operator and fitness assignment strategies. Special care is also taken to prevent non-dominated solutions from being eliminated in the evolutionary process.

Multiobjective optimization using Genetic Algorithms follows the same general procedure (as single objective optimization), that is listed below:

Generate-Initial-Population();
repeat
    Selection();
    Reproduction();
until *max number of generations reached* ;
return *Pareto-Optimal Solutions*;

Figure 6.1: Pseudo-code of a basic multiobjective Genetic Algorithm

The main challenge concerning multiobjective optimization using GA's is the implementation of the selection operator; that is, how to evaluate and compare solutions in the presence of multiple objectives.

6.3 Selection in the Presence of Multiple Objectives

The selection operator (in single and multiobjective optimization problems) determines the solutions that will be selected for the reproduction of the next generation. The selection operation emphasizes fit individuals in the population by giving them a higher chance to breed. The selection scheme should emphasize the characteristics of good solutions in order for the evolutionary process to produce better solutions in successive generations. In the presence of multiple objectives, the determination of "better solutions" is not as straight forward as it is in the single objective case. In review, the objective of the selection operation in a multi-objective problem is to lead the evolutionary process to a set of optimal solutions, rather than one optimal solution as in the case of single objective problems. This section outlines some of the selection schemes devised by different researchers in the field.

6.3.1 Objective Aggregation

One of the simplest approaches for dealing with a multiobjective problem is to convert the problem into a single objective problem. This is done by implementing a mechanism

that combines the multiple objectives into a single objective. These approaches try to converge to a specific point on the Pareto front. Therefore the combining mechanism determines the relative importance of the objectives. In these methods, to generate the entire Pareto front, the user must perform multiple runs and vary the conditions of the combining mechanism. The simplest method for combining the objectives is the weighted sum approach. This can be expressed as follows:

$$fit(x) = \sum_{i=1}^{L} w_i f_i(x) \tag{6.4}$$

having,

$$\sum_{i=1}^{L} w_i = 1 \tag{6.5}$$

where L denotes the number of objectives and $f_i(x)$ denotes the i-th objective function. This is the approach that was adopted in AG-ART. It follows immediately that the solution that optimizes $fit(x)$ is a Pareto optimal point, since if not, then there must exist a feasible x which improves on at least one of the objectives without compromising the others and hence produces a smaller value of the weighted sum. It is necessary to scale or normalize the objectives before using equation (6.4) to avoid having one objective dominate the others. This requires knowledge of the range of each objective, which is knowledge that might not be available for many real world applications.

The difficulty with this approach is determining the appropriate weights. In this case, any optimal point obtained will be a function of the coefficients used to combine

the objectives. Most researchers generate the Pareto front by varying the weights. This approach is very simple and easy to implement. However, in addition to being computationally costly, it has two serious drawbacks. The first drawback is that this scheme is not able to generate the non-convex regions of the Pareto front for any combination of the weights. This has been pointed out by a number of researchers, such as [Coe00] and [DD97]. The second drawback is that the solutions, selected by evenly varying the weights, are not guaranteed to be evenly distributed on the Pareto front. This becomes more important when the fitness function is used to determine the selection probability, in which case the probability of selection will vary across the Pareto front solutions based on the shape of the Pareto front. This drawback results in a poor coverage of the Pareto solutions found.

To show why this happens, a graphical illustration is used for a two-objective problem. This problem can be stated as follows:

$$minfit(x) = \alpha f_1(x) + (1 - \alpha) f_2(x) \tag{6.6}$$

where, $\alpha \in [0, 1.0]$, $f_1(x) \in [0, 1.0]$ and $f_2(x) \in [0, 1.0]$. Figure 6.2 illustrate a convex Pareto front for this problem.

Equation (6.6) can be reorganized as follows to describe an equation for a line that defines a set of solutions for a given value of α. The line has a slope that depends on the value of α. To choose a solution, this line shifts parallel to itself until it touches the Pareto front, while minimizing the value of $fit(x)$. As can be seen in Figure 6.3, varying

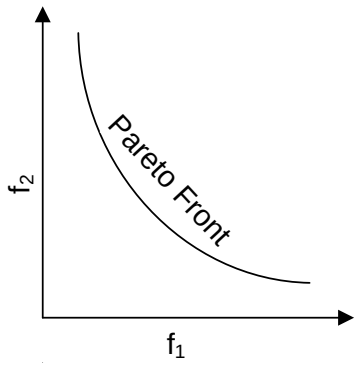


Figure 6.2: Convex Pareto front for a two-objective problem

the value of α causes the fitness function to choose different Pareto solutions. By varying α we are able to produce all solutions of this convex Pareto front.

$$f_2(x) = \frac{fit(x)}{1-\alpha} - \frac{\alpha}{1-\alpha} f_1(x) \tag{6.7}$$

In other words, the solution chosen is where the line is tangent to the Pareto front. The slope of the line is dependent of α. The slope of the point chosen at the Pareto front is therefore dependent on the value of α.

$$\frac{df_2(x)}{df_1(x)} = -\frac{\alpha}{1-\alpha} \tag{6.8}$$

Figure 6.4 shows the Pareto front with three points on it A, B and C. A solution is chosen by shifting a line, whose slope depends on the value of α, parallel to it itself until it touches the Pareto front, while minimizing the value of $fit(x)$. Figure 6.4 shows 3 lines for 3 different values of the weight α. However, as shown in Figure 6.4, it is not possible for this fitness function to choose point B as the line tangent to the Pareto front at point

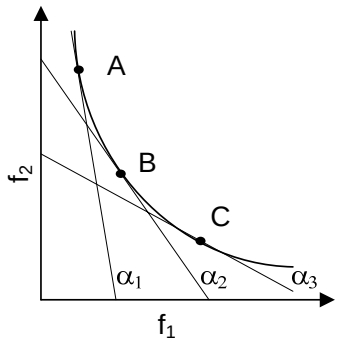


Figure 6.3: Solutions selected on the convex Pareto front

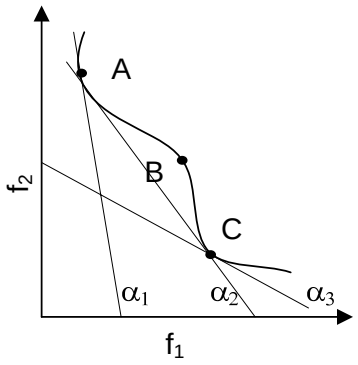


Figure 6.4: Solution selected on a Pareto front with concavity
Pareto front and lines corresponding to different values of the weight, α. The fitness function cannot select points in the concave region for any value of the α.

B does not minimize $fit(x)$. Therefore, for any weight combination, this fitness function will not be able to choose point B.

The second problem is illustrated in Figure 6.5, where it can be easily seen that uniformly varying the slope of the line will cause the selection of points to be concentrated in region where the second derivative of the Pareto front $\frac{d^2 f_2(x)}{df_1(x)^2}$ is largest.

There are other methods that aggregate the objectives into a scalar fitness function. We list here briefly some of these methods:

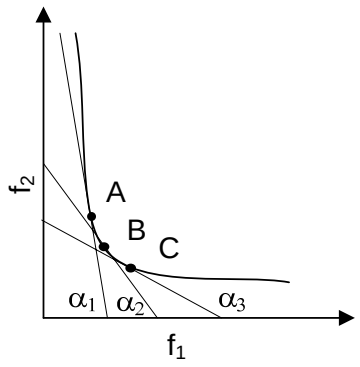


Figure 6.5: The distribution of solutions on the Pareto front
The Distribution will be dependent on the weight, α. Running the GA for uniformly separated values of α does not necessarily find uniformly distributed solutions ont the Pareto front. Solutions will be concentrated around parts where the slope is changing the fastest.

- Weighted goal programming. This method defines the fitness function as the sum of the deviations of the objective functions from target values as follows:

$$fit(x) = \sum_{i=1}^{L} w_i |f_i(x) - T_i|^n \tag{6.9}$$

  Where T_i is the target value for objective i and n is usually set at 1.

- Lexicographic Ordering. In this method, the designer ranks the objectives in order of importance. The optimum solution is then obtained by minimizing the objective functions, starting with the most important and proceeding according to the assigned order of importance.

- Weighted Min-Max Approach. This method compares relative deviations from separately attainable optima. It tries to find a solution that gives the smallest values of the relative deviations from optima of all the objective functions. In other

words, it tries to minimize the maximum deviation from the optimum of any of the objectives. The weighting allows incorporating preference in the selection process.

6.3.2 Early Multiobjective Approaches

The aggregation of objective causes the evolutionary process to move the population towards a single point on the Pareto front. The more recent research in multiple-objective optimization avoids combining the objectives into a single objective. Rather, they treat the objectives separately, and solutions are evaluated with respect to each one of the objectives at every generation. Therefore, these approaches are more suited for finding multiple Pareto solutions. These approaches do not normally require a mechanism that determines the relative importance of objectives. The aggregation methods of selection, mentioned above, are often referred to as *a-priori methods* because they normally incorporate preference before hand. Alternatively, the methods that attempt to produce the whole Pareto front and give the option to the user to decide from a set of optimal solutions are referred to as *a-posteriori methods*.

One early example of the a-posteriori method is the pioneering work of Schaffer [Sch85] where Vector Evaluated Genetic Algorithm (VEGA) was introduced. In VEGA, the selection step generates a number of sub-populations by performing proportional selection according to each objective in turn. Then these sub-populations are combined to obtain a new population, on which the genetic operators, crossover and mutation, are applied.

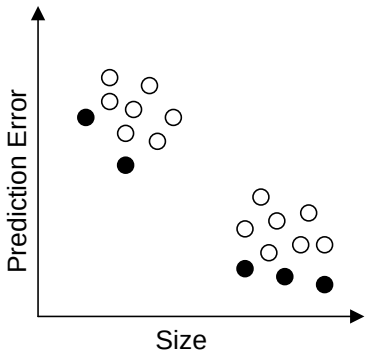


Figure 6.6: Speciation in a two objectives optimization problem

VEGA has a major drawback which is that its selection scheme is biased towards some Pareto optimal solutions. If, for example, there is an individual that encodes a good compromise solution for all the objectives, but it is not the best in any of them, it will be discarded. However, that individual should really be preserved because it encodes a Pareto-optimal solution. Due to its selection mechanism, the population in VEGA tends to split into different groups of individuals, referred to as species, each of them particularly strong in one of the objectives. Schaffer called this effect *speciation*. The effect of speciation is undesirable because it results in poor coverage of the Pareto front.

Another approach was devised by Hajela and Lin [HL92] that used the weighted-sum method for fitness assignment. Each objective is assigned a weight, where the weights are not fixed but rather encoded in the genotype. The diversity of the weight combinations is promoted by phenotypic fitness sharing. As a consequence, the EA evolves solutions and weight combinations simultaneously.

6.3.3 Pareto-based Ranking

Some authors suggested ranking of solutions based on their Pareto optimality as a method to minimize the problems observed in VEGA, in which the GA converges to a subset of the Pareto front. In this scheme, Pareto optimal solutions are equally assigned the highest fitness, and therefore, they have increased chance of survival and breeding. The rest of the population is assigned fitness values that depend on their closeness to the Pareto front. This idea was initially suggest by [Gol89] and later implemented (sometimes with modifications) by several authors. In [Gol89], the author suggests finding the set of solutions in the population that are Pareto nondominated by the rest of the population. These individuals are then assigned the highest rank and eliminated from further contention. Another set of Pareto nondominated individuals are determined from the remaining population and are assigned the next highest rank. This process continues until the population is suitably ranked (see Figure 6.7).

This ranking scheme was found to be $O(\lambda^3)$ where λ denotes the population size. In addition this ranking scheme requires another mechanism to differentiate between solutions in the same rank based on crowdedness of the region they are located at. This is important to ensure diversity. Some of these mechanisms are discussed later.

In [FF93] the authors suggest a modified scheme than the one suggested by [Gol89] for implementing the Multiobjective Genetic Algorithm (MOGA). In MOGA, the rank of a certain individual corresponds to the number of chromosomes in the current population by which it is dominated. For an individual x at generation t which is dominated by $p(t)$ individuals in the current population, the individual's rank is given by $rank(x, t) =$

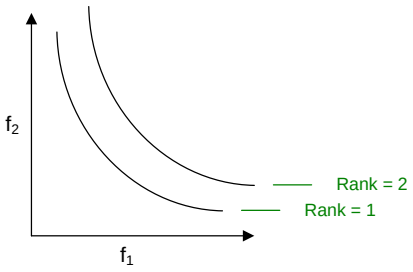


Figure 6.7: Pareto Ranking

$1 + p(t)$. The authors combine this ranking technique with fitness sharing (discussed later) to improve diversity of solutions.

In [HNG] the authors use a tournament selection scheme based on Pareto dominance in their implementation of Niched-Pareto Genetic Algorithm (NPGA). Two random individuals are picked from the population for selecting a winner in a tournament selection. The two individuals are compared to a randomly selected comparison set. If one solution is nondominated by the comparison set while the other is dominated, the non-dominated solution wins the tournament. If both competitors are either dominated or non-dominated, a niche count is found for each individual in the entire population. The niche count is calculated by simply counting the number of solutions in the population within a certain distance from an individual. The individual with least niche count is selected.

Also, based on Goldgerg's suggestion [Gol89], Srinivas introduced a genetic algorithm referred to as Non-dominated Sorting Genetic Algorithm (NSGA) [SD94]. The selection in NSGA is based on a ranking procedure where all nondominated individuals are classified into one category and given a large dummy fitness value. The fitness value is then

shared between individuals in this category. Sharing is achieved by degrading fitness values by dividing the original fitness value of an individual by a quantity proportional to the number of individuals around it. Then this group of classified individuals is ignored and another layer of nondominated individuals is considered. These individuals are given equal fitness value that is kept smaller than the minimum given, after sharing, for the previous group. This process continues until the whole population is classified. This ranking procedure is referred to as nondominated ranking.

In a similar fashion, as a successor to NPGA [HNG], Erickson introduced a revised version referred to as Niched Pareto Genetic Algorithm 2 (NPGA 2) [EMH01]. This algorithm uses Pareto ranking but keeps tournament selection (solving ties through fitness sharing as in the original NPGA). Niche counts in the NPGA 2 are calculated using individuals in the partially filled next generation, rather than using the current generation.

6.3.4 Diversity Preserving Mechanisms in Multimodal Optimization

The success of a genetic algorithm in finding good optimal solutions depends on the ability to preserve diversity of the population. Maintaining a high level of diversity means that the GA is better covering the solution space, and therefore, is more likely to find global optima. Diversity is an important issue for single-objective problems, but is especially important for the multiobjective case, because in the later we are looking for, not one, but a set of optimum solutions.

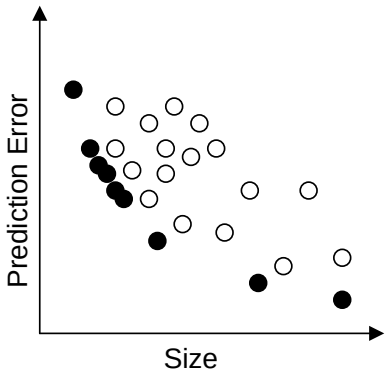


Figure 6.8: Genetic drift problem

It has been pointed out in the literature that GAs tend to converge to single solution [DG89] due to the accumulation of stochastic errors associated with its genetic operators when using a finite population. This phenomenon is sometimes referred to as *genetic drift*.

In MOEA this can be explained as follows: when the number of individuals at a certain region in the objective space (referred to as niche) increases, they tend to produce more copies during the next selection and reproduction cycle. This would in turn lead to an even further increase of the individuals in that niche. Therefore, a self-perpetuating effect keeps accumulating individuals of a particular niche one generation after another leading to convergence at one point on the Pareto front. To counter this issue, researchers devised a number of techniques such as fitness sharing, crowding and other mating restriction techniques. Most MOEAs discussed so far implement a form of the mechanisms described below to preserve diversity:

- Fitness sharing: To help maintain the diversity in the population and avoid the effect of genetic drift, fitness sharing was introduced by Goldgerg [GR87]. The sharing mechanism penalize the individuals based on the number of individuals

that are in close proximity. It was based on the theory that organisms in a common niche will compete for the same amount of resource, and as a result the amount of resource which each organism can get will be reduced by the existence of other organisms in the same niche. In [GR87], the author suggests implementing sharing defined by the following sharing function:

$$\phi(d_{ij}) = 1 - \frac{d_{ij}}{\sigma_{share}} \tag{6.10}$$

if $d_{ij} < \sigma_{share}$, otherwise $\phi(d_{ij}) = 0$. And the fitness of each individual is updated as follows:

$$fit_s(i) = \frac{fit(i)}{\sum_{j=0}^{M} \phi(d_{ij})} \tag{6.11}$$

where σ_{share} is the niche radius and M is the number of solutions within same niche as solution i. Implementing such a mechanism requires the calculation of distance, d_{ij}, between individuals. This has been done in the input variable domain (see [SD94]) and the objective functions domain (see [FF93]). The individuals are said to be in the same niche if they are located within a certain distance referred to as the niche radius σ_{share}. A number of authors proposed theoretical methods for estimating the niche radius (see [FF93]). The main problem with sharing is that it requires the specification of a sharing radius, σ_{share}, which is a problem dependent parameter.

- Crowding: Crowding is a mechanism in which after a new chromosome is created, one old chromosome in the population, which is most similar to the new chromosome is chosen to be replaced, where similarity is based on the genotypical distance (see [Coe00]).

- Mating restriction: To avoid excessive competition between distant individuals as the population distributes itself around multiple regions of optimality Goldberg [Gol89] suggested the use of mating restrictions, where mating is allowed only locally. Since different regions of optimality on the tradeoff surface may be very different genotypically [FF95], mating of individuals that are different in the objective space may be less likely to produce good offsprings. Therefore, enforcing local mating can be argued to allow creation of more stable niches, and eventually more efficient search. This technique is said to preserve diversity by encouraging the formation of stable sub-populations, referred to as niches. Each sub-population is superior in a certain tradeoff aspect of the problem.

More recent MOEAs avoid the implementation of a separate diversity preserving mechanism. Instead, the fitness assignment strategy is adjusted to penalize crowded regions. The more recent MOEAs have also been characterized by the use of Pareto elitism. These MOEAs are discussed below.

6.3.5 Pareto Elitism

Elitism is a selection mechanism that aims at preserving good performance over successive generations. In multiobjective optimization, elitism refers to preserving nondominated solutions found along the evolutionary process. Elitism has been adopted in the more recent MOEA research. According to [ZT99], elitism can speed up the performance of the GA significantly, and also can help preventing the loss of good solutions once they are found.

In [IM96], the authors use a random weighted sum approach (with elitism) to produce the Pareto front. The weights are generated randomly each time an individual is selected. Therefore, in every generation, selection pressure is applied in multiple directions towards the Pareto front. The nondominated set of solutions is stored externally and updated every generation.

SPEA (Strength Pareto Evolutionary Algorithm) introduced by Zitzler and Thiele in [ZT99], implements Pareto-elitism by storing nondominated solutions in an externally maintained archive. As in MOGA the fitness of an individual depends on the number of solutions that dominates it. However, in SPEA, for each individual in the external set, a strength value is computed. This strength is proportional to the number of solutions a certain individual dominates. The fitness of each member of the current population is then computed as the sum of the strengths of all external non-dominated solutions that dominate it. Specifically, the chromosomes in the archive A and population P are assigned fitness values based on dominance relationship. In archive A, the fitness is assigned as follows:

$$Fit_A(x) = \frac{|y \in P, x \prec y|}{|P|}, x \in A \tag{6.12}$$

That is, the fitness of a solution x in A is the number of solutions in P that x dominates, divided by the number of solutions in P. The fitness of solutions in P is assigned as follows:

$$Fit_P(y) = 1 + \sum_{x \in A, x \prec y} Fit_A(x), y \in P \tag{6.13}$$

That is, the fitness of solution y in P is 1 plus the sum of the fitness of all solutions in A that dominate solution y.

The mechanism of such a fitness assignment mechanism automatically penalizes crowded solution regions and serves as a mechanism of encouraging diversity in the Pareto set without the need to specifying other parameters (such as those related to the fitness sharing mechanism). A clustering technique is implemented to keep the size of the external archive small.

In [ZLT01] a revised version of SPEA is introduced, referred to as SPEA2. The revised version incorporates a fine-grained fitness assignment strategy which takes into account for each individual the number of individuals that dominate it and the number of individuals by which it is dominated. It uses a nearest neighbor density estimation technique which guides the search more efficiently, and it has an enhanced archive truncation method that guarantees the preservation of boundary solutions. In particular, each individual is assigned a strength value that is equal to the number of solutions it

dominates. After that, a raw fitness, $R(x)$, is assigned for each individual to be the sum of the strengths of all its dominators in both A and P. The raw fitness is then adjusted as follows. For each individual, x, the distance, in objective space, to the k-th nearest neighbor is found and denoted as $\sigma_k(x)$. The value of k is chosen to be the square root of the sum of the size of the archive and population. The fitness of each individual is then calculated using the following equation:

$$Fit(x) = R(x) + \frac{1}{\sigma_k + 2} \tag{6.14}$$

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) [DAP], [DPA02] was introduced as a revised successor to the original NSGA [SD94]. The authors present a more efficient algorithm that uses elitism and a crowded comparison operator that keeps diversity without the need to specify a sharing parameter. Its elitist mechanism consists of combining the best parents with the best offspring obtained.

In [FES03] the authors points out to the problem of using an elite archive of fixed size. It is shown that limiting the size of the elite archive can produce "retreating" or "oscillating" estimates of the Pareto front. This happens as the archive is truncated when its size exceeds the limit and then new solutions are added that might be dominated by solutions that were eliminated during the truncation process. Therefore, the authors recommend keeping all nondominated solutions found during the evolutionary search. To speedup processing of large number of nondominated individuals, a tree data structure is introduced and used for fast searches, additions and deletion to the archive.

MO-GART adopts a fitness assignment that is similar to the one introduced in SPEA2 [ZLT01]. Also, as is the case for a number of previously proposed multiobjective evolutionary approaches, an external elitist archive of continuously updated Pareto solutions is maintained. The size of the external archive is not fixed and the truncation procedure suggested in SPEA2 is not used. In our implementation, we use a mechanism that ensures that boundary solutions (best solutions in each objective) are always selected as parents for the next generation. This technique was suggested in [FES03] and was found to be effective in improving the efficiency of MO-GART.

6.4 Multiobjective Evolutionary ART Architectures

This section proposes a multiobjective approach for evolving ART architectures. The resulting architectures are collectively referred to as MO-GART. MO-GART uses a multiobjective evolutionary approach to find networks that achieve Pareto-optimal performance in terms of two objectives: minimizing classification error and minimizing complexity (size) of the ARTMAP classifier. MO-GART operates by applying, repeatedly, genetic operators on an initial population of trained ART networks. The pseudo-code of the basic steps of MO-GART is shown in Figure 6.9.

The main difference between AG-ART introduced in Chapter 4 and MO-GART is the selection operator. In AG-ART the selection operator bases the selection of parents using a fitness function that combines the two objectives using a weighted sum. In MO-GART selection is based on Pareto optimality and therefore MO-GART is capable of finding

```
P(0) ← Generate-Initial-Population();
A(0) ← Initialize-Empty-Archive();
for t ← 1 to Gen_max do
    Evaluation();
    Update-Archive(P(t), A(t));
    if stopping criteria met then exit for;
    P'(t) ← Selection(P(t), A(t));
    P(t) ← Reproduction(P'(t));
end
return A(t);
```

Figure 6.9: Pseudo-code of MO-GART Algorithm

multiple solutions on the Pareto front, in one run. Also, MO-GART uses a continuously updated Pareto archive where the Pareto solutions found so far are stored.

The algorithm starts by generating an initial population, $P(0)$, of ARTMAP networks (FAM, EAM or GAM), each one of them trained with a different value of the baseline vigilance parameter $\bar{\rho_a}$, and order of training pattern presentation. The networks are encoded into chromosomes in similar fashion as AG-ART, where each component (gene) represents a category (hidden node) of an ART network. Each component contains the weight information for the category. The chromosomes in MO-GART are variable length, where the length is equal to the number of categories in the network represented by the chromosome (see Figures 3.12, 3.13, 3.14).

Also, MO-GART initializes an empty secondary population, $A(0)$, that will be used to store nondominated solutions found during the evolution. In each generation, each solution in the population is evaluated according to each objective function. That is, the error rate of each ARTMAP network is evaluated by running it against a validation set. The second objective, complexity, is represented by the number of categories present in

each network. Once networks in population P are evaluated, the archive A is updated by adding to it the solutions in P that are nondominated by solutions in A. Also, solutions in A that are now dominated by solutions just added from P, are removed from the archive A. This mechanism ensures elitism.

The algorithm runs for a maximum number of generations defined by Gen_{max}. However, to avoid running MO-GART for an unnecessarily large number of generations, the evolution is also stopped when the archive A is not updated for 10 consecutive generations.

The selection process creates a temporary population P', where the parent chromosomes used to create the next generation are selected. The chromosomes in the archive A and population P are assigned fitness values based on a dominance relationship. In this scheme each individual is assigned a strength value that is equal to the number of solutions it dominates. After that, a raw fitness, $R(x)$, is assigned for each individual to be the sum of the strengths of all its dominators in both A and P. The raw fitness is then adjusted as follows. For each individual, x, the distance, in objective space, to the k-th nearest neighbor is found and denoted as $\sigma_k(x)$. The value of k is chosen to be the square root of the sum of the size of the archive and population. The fitness of each individual is then calculated using the following equation:

$$Fit(x) = R(x) + \frac{1}{\sigma_k + 2} \tag{6.15}$$

The parents are then chosen using a deterministic binary tournament selection with replacement, as follows: For each parent, randomly select two chromosomes from the

combined set of A and P, and choose, the chromosome with the smallest fitness value. Boundary solutions, which are networks with smallest error rate and smallest size, are ensured to be copied in the set of parents.

Once the selection step determines the parents, reproduction operators are used to create individuals for the next generation, as described in Section 4.3. The chromosomes in $P(t)$ are then replaced by chromosomes created by performing crossing over pairs of parents in $P'(t)$.

As mentioned above, the evolutionary process continues until one (of the two) stopping criterion is triggered. MO-GART does not return a single trained ART classifier, but rather, a number of ART classifiers that were present in the archive A at the last generation of the evolutionary process. These classifiers have achieved varying levels of accuracy and complexity. These alternatives are then presented to the user to make the final decision of choosing one (or more) of these classifiers. For example, if the user is mostly interested in accuracy, then the network that produced the best accuracy is chosen.

6.5 Evaluation of The Approach

In this section MO-GART's performance is compared to that of other popular ART architectures. The objective of this comparison is not to compare different multiobjective evolutionary approaches; rather, the objective here is to compare the accuracy and size of several neural network architectures against the one proposed. In particular, MO-

GART's performance is compared to that of other popular ART architectures, which have been proposed in the literature with the intent of addressing the category proliferation problem, such as ssFAM, ssEAM, and ssGAM. This section also compares the performance of MO-GART to the previously introduced genetic ART architectures (AG-ART) that did not use a multiobjective evolutionary approach. In the AG-ART case, the Pareto front is produced by varying the weight in the fitness function. The experiments were conducted on 11 datasets. More information about these datasets can be found in Appendix B.

Since in this work we are not only focusing on generalization performance, but also on the size (complexity) of the network produced, it becomes more complicated to compare and rank networks. To provide a fair comparison, a comparison approach that considers the two objectives simultaneously is used. Since the existence of the two, sometimes competing, objectives result in multiple good solutions rather than one "best" solution, in this comparison, multiple solutions are assessed (sets of solutions) produced by the different algorithms, under consideration. In other words, for each classification algorithm, a number of classifiers are produced that have attained the two objectives (good generalization and small size) at different levels of success. Then the *non-dominated solutions* are chosen. As discussed earlier, a non-dominated solution is defined to be a network, where no other network achieves better generalization utilizing equal or smaller number of categories. The comparison between algorithms is then based on the quality of the non-dominated set that was produced by each algorithm. The comparison also includes the time it takes each algorithm to produce the non-dominated set of solution networks.

Experiments were conducted for the three MO-GART architectures: MO-GFAM, MO-GEAM, and MO-GGAM for each of the 11 databases. The average computation time in seconds (over 10 replications) needed to produce the solutions, is referred to as the *Total Run Time*, and reported in Tables 6.2 and 6.4.

6.5.1 Comparison with ssART

For each of the ssFAM, ssEAM, and ssGAM, and for each of the 11 databases, a number of experiments were performed with different settings of their network parameter values. In particular, 1,800 different parameter settings were considered for ssFAM, 6,480 different parameter settings for ssEAM and 6,000 different parameter settings for ssGAM. It should be emphasized that the parameter ranges used were determined based on experience of what are good parameter settings for these ART networks. The parameter settings were chosen to provide varying levels of accuracy and complexity in these networks. Solutions that are Pareto-optimal with respect to those two objective were finally chosen. The total computation time required to obtain these network solutions for each database and each method, which is the sum of training and validation CPU times in seconds for all the tried settings, is reported in Table 6.2, and referred to as the *Total Run Time*.

A one-to-one comparison of the results reported in Table 6.2 reveals that the *Total Run Time* of the MO-GART networks is smaller, in most instances an order of magnitude smaller than the *Total Run Time* of their corresponding counterparts, ssART networks. To compare the generalization performance of MO-GFAM and ssFAM, MO-GEAM and

ssEAM, and finally MO-GGAM and ssGAM a metric that compares the network solutions obtained by the ss-network (for all different parameter settings) and the network solutions obtained by the MO-GART is used. This metric has been used before in similar situations (see [ZT99]). This metric is defined as follows:

$$C(A,B) = \frac{|b \in B : \exists a \in A, b \prec a|}{|B|} \tag{6.16}$$

This metric measures the fraction of members in set B that are dominated by at least one member in set A. Therefore, $C(A,B) = 1$ means all members in B are dominated by members in A. In this case the approach that produced set A is a clear winner. It is obvious that $C(B,A)$ must also be considered in order to properly compare the two sets.

Since the calculated values of *C(A, B)* and *C(B, A)* are dependent on the seed used to evolve the population of FAMs, EAMs and GAMs in the MO-GART approach, network solutions were produced by changing the seed 10 times. Consequently, 10 different values of the C metric were produced for each comparison pair. Table 6.1 compares the average values (over the 10 replications) of *C(MO-GFAM, ssFAM)* versus *C(ssFAM, MO-GFAM)*, and *C(MO-GEAM, ssEAM)*, versus *C(ssEAM, MO-GEAM)*, and *C(MO-GGAM, ssGAM)* versus *C(ssGAM, MO-GGAM)*. It is obvious from the table that the average values of *C(MO-GFAM, ssFAM)* are larger than *C(ssFAM, MO-GFAM)* values, which indicates that networks produced by MO-GFAM are more likely to dominate networks produced by ssFAM, and therefore, the networks produced by MO-GFAM are expected to be of higher quality. Similar conclusions can be drawn for MO-GEAM versus ssEAM and MO-GGAM versus ssGAM. This result is expected since MO-GART uses

a multiobjective approach that is designed to produce a high quality Pareto front. To provide a fair comparison, the performance of the most accurate networks is shown in Tables 6.5, 6.6 and 6.7. As it can be easily seen, the MO-GART networks were able to consistently find more accurate networks using, in most instances, much smaller network sizes.

6.5.2 Comparison with AG-ART

For AG-ART it is not possible to produce the nondominated solutions in one run. Rather, it is necessary to run the algorithm multiple times to produce the different nondominated solutions. This was done by running AG-ART using five different settings for the fitness weight. This process is repeated 10 times to account for the stochasticity of the genetic algorithm. The average time it took to produce 1 set of nondominated solutions is reported in Table 6.4, and referred to as the *Total Run Time*. The result in Table 6.3 shows an advantage of MO-GART over AG-ART in terms of solution quality. Also, a one-to-one comparison of the results reported in Table 6.4 reveals that the *Total Run Time* of the MO-GART networks is smaller than the *Total Run Time* of their corresponding counterparts, AG-ART networks. Tables 6.5, 6.6 and 6.7 compares the most accurate network obtained from MO-GART and AG-ART. As it can be seen, MO-GART in most cases was able to find a better solutions than AG-ART. Therefore, achieving better solution quality at a lower computational cost, in addition to producing multiple optimal

Table 6.1: C-metric values for MO-GART vs. ssART

Database Name	C(MO-GFAM, ssFAM)	C(ssFAM, MO-GFAM)	C(MO-GEAM, ssEAM)	C(ssEAM, MO-GEAM)	C(MO-GGAM, ssGAM)	C(ssGAM, MO-GGAM)
1Ci/Sq	0.550	0.380	0.954	0.158	0.941	0.053
g4c-25	1.000	0.050	1.000	0.000	0.950	0.000
g6c-15	1.000	0.000	1.000	0.000	0.900	0.000
glass	0.533	0.342	0.956	0.058	0.871	0.000
Iris	0.500	0.233	0.900	0.183	0.871	0.245
page	1.000	0.033	1.000	0.000	1.000	0.025
pendigits	0.542	0.191	0.826	0.143	0.639	0.236
pima	1.000	0.000	0.780	0.050	0.983	0.150
sat	1.000	0.000	0.922	0.066	0.950	0.030
seg	0.700	0.180	0.772	0.265	0.879	0.000
wav	1.000	0.000	1.000	0.000	1.000	0.000

Table 6.2: Total run time for MO-GFAM, MO-GEAM and MO-GGAM compared to total run time for ssFAM, ssEAM, ssGAM

Database Name	MO-GFAM	ssFAM	Gain	MO-GEAM	ssEAM	Gain	MO-GGAM	ssGAM	Gain
1Ci/Sq	22.6406	216.42	9.56	63.6581	1167.67	18.34	39.4752	1431.35	36.26
G4C-25	4.4593	130.92	29.36	11.4466	916.78	80.09	9.1047	314.49	34.54
G6C-15	5.6252	145.16	25.80	12.4733	508.22	40.74	9.8514	266.77	27.08
glass	0.1281	2.82	21.98	0.2092	5.72	27.34	0.2812	3.52	12.50
Iris	1.4202	21.30	15.00	4.7829	123.56	25.83	7.4811	109.25	14.60
page	4.2141	69.52	16.50	8.497	484.15	56.98	8.7017	125.68	14.44
pendigits	462.578	7864.27	17.00	997.23	58865.05	59.03	129.90	20050.39	154.35
pima	0.1173	3.68	31.41	0.4063	75.88	186.76	0.2265	32.75	144.57
sat	170.1563	2034.29	11.96	382.3469	17162.45	44.89	84.8639	4302.16	50.69
seg	10.2047	85.55	8.38	41.0937	1331.47	32.40	7.4577	509.99	68.38
wav	23.103	763.99	33.07	68.2825	8612.65	126.13	8.3548	1199.58	143.58

solutions at once; justifying the proposed approach of using a multiobjective approach to evolve ART architectures.

6.6 Summary

This chapter introduced a multiobjective evolutionary approach to optimize ARTMAP neural networks in terms of two objectives: classification accuracy (higher is better) and classifier complexity (smaller is better). In particular, a MOEA is applied to optimize the performance of three well known ART architectures: Fuzzy ARTMAP, Ellipsoidal ARTMAP, and Gaussian ARTMAP. The resulting architectures are referred to as MO-GFAM, MO-GEAM and MO-GGAM, and collectively as MO-GART.

The MO-GART approach presents a solution to the category proliferation problem in ART. Other approaches to solve the category proliferation problem in ART have been proposed before, such as the semi-supervised ART (ssART) approach (ssFAM, ssEAM, and ssGAM). An extensive comparison of MO-GART and the ssART approach concluded that the MO-GART approach is more elegant (does not require tweaking of the ART network parameters), more effective (produces higher accuracy and smaller size network solutions), and more efficient (faster) than the ssART approach. The results, presented in Tables 6.1 and 6.2, indicate that MO-GART offers clear advantages compared to ssART; it is worth noting that ssART is a class of well performing ART classifiers that compares very favorably with other ART and non-ART classifiers. The advantage of MO-GART compared to AG-ART is that MO-GART focuses on two objectives at once.

Table 6.3: C-metric values for MO-GART vs. AG-ART

Database Name	C(MO-GFAM, AG-FAM)	C(AG-FAM, MO-GFAM)	C(MO-GEAM, AG-EAM)	C(AG-EAM, MO-GEAM)	C(MO-GGAM, AG-GAM)	C(AG-GAM, MO-GGAM)
1Ci/Sq	0.482	0.429	0.417	0.382	0.467	0.362
G4C-25	0.600	0.333	0.550	0.400	0.600	0.542
G6C-15	0.500	0.300	0.700	0.408	0.750	0.408
glass	0.700	0.333	0.675	0.350	0.733	0.358
Iris	0.700	0.150	0.700	0.267	0.600	0.148
page	0.550	0.533	0.717	0.242	0.550	0.408
pendigits	0.350	0.270	0.235	0.218	0.300	0.196
pima	1.000	0.200	0.550	0.400	0.783	0.450
sat	0.592	0.134	0.417	0.199	0.433	0.176
seg	0.348	0.325	0.443	0.341	0.525	0.384
wav	0.767	0.150	0.700	0.183	0.600	0.500

Table 6.4: Total run time for MO-GFAM, MO-GEAM and MO-GGAM compared to total run time for AG–FAM, AG-EAM, AG-GAM

Database	MO-GFAM	AG-FAM	Gain	MO-GEAM	AG-EAM	Gain	MO-GGAM	AG-GAM	Gain
1Ci/Sq	22.64	68.51	3.03	63.66	152.13	2.39	39.48	142.29	3.60
G4C-25	4.46	52.59	11.79	11.45	82.99	7.25	9.10	89.32	9.81
G6C-15	5.63	92.01	16.36	12.47	127.15	10.19	9.85	137.82	13.99
glass	0.13	1.71	13.32	0.21	2.26	10.81	0.28	2.64	9.38
Iris	1.42	13.48	9.49	4.78	21.56	4.51	7.48	22.02	2.94
page	4.21	52.97	12.57	8.49	60.91	7.17	8.70	75.03	8.62
pendigits	462.57	1142.43	2.47	997.23	4304.84	4.32	129.90	537.62	4.14
pima	0.12	0.95	8.10	0.41	2.41	5.94	0.23	1.91	8.45
sat	170.16	508.21	2.99	382.35	1234.62	3.23	84.86	329.07	3.88
seg	10.20	41.93	4.11	41.09	88.45	2.15	7.46	64.13	8.60
wav	23.10	147.23	6.37	68.28	369.38	5.41	8.35	83.94	10.05

Table 6.5: Most accurate networks and their sizes: FAM

	MO-GFAM		AG-FAM		ssFAM	
	PCC	Size	PCC	Size	PCC	Size
1Ci/Sq	97.97	31	98.07	31	98.10	78
G4C-25	76.00	4	74.94	4	74.22	4
G6C-15	84.59	6	84.75	6	82.49	9
glass	76.56	6	76.56	6	73.44	7
Iris	95.19	2	94.96	2	94.56	2
page	96.45	5	95.59	6	94.77	6
pendigits	98.27	271	98.20	282	97.14	66
pima	82.67	2	79.31	4	73.28	4
sat	89.12	175	88.90	310	84.20	51
seg	95.43	25	94.86	22	94.14	32
wav	86.30	3	85.90	4	75.65	16

Table 6.6: Most accurate networks and their sizes: EAM

	MO-GEAM		AG-EAM		ssEAM	
	PCC	Size	PCC	Size	PCC	Size
1Ci/Sq	99.76	2	99.70	2	97.40	99
G4C-25	75.54	4	75.14	4	73.90	4
G6C-15	84.69	6	85.01	6	83.23	24
glass	75.31	6	75.00	6	73.44	17
Iris	95.24	2	95.04	2	94.65	2
page	96.40	5	95.09	5	94.44	24
pendigits	98.90	331	98.31	331	96.60	179
pima	83.33	4	78.88	3	75.00	6
sat	88.34	198	87.85	203	85.50	141
seg	93.86	52	93.71	128	91.57	83
wav	86.35	5	87.15	4	79.80	12

Table 6.7: Most accurate networks and their sizes: GAM

	MO-GGAM		AG-GAM		ssGAM	
	PCC	Size	PCC	Size	PCC	Size
1Ci/Sq	99.80	2	99.83	2	94.63	26
G4C-25	75.92	4	75.24	4	74.84	23
G6C-15	85.17	6	84.97	6	85.07	20
glass	76.00	8	73.44	9	68.75	14
Iris	94.90	2	94.75	2	95.21	7
page	96.38	5	96.34	6	94.52	7
pendigits	98.10	88	97.83	108	97.43	87
pima	82.67	2	76.72	2	72.41	3
sat	88.75	106	88.35	118	87.00	81
seg	92.59	13	92.71	17	91.29	31
wav	87.15	4	87.50	3	85.35	11

Consequently, MO-GART does not require multiple GA runs to produce multiple good solutions to the classification problem, under consideration, and hence it is more efficient than the AG-ART approach (as Table 6.4 reveals). Finally, MO-GART is more elegant than AG-ART because it does not require a user intervention to specify a-priori the preference towards one objective (accuracy) versus the other (size).

CHAPTER 7
CONCLUSION

In this dissertation a family of new evolved ART neural networks was introduced, discussed, analyzed, and evaluated for a variety of classification problems. This family of new evolved ART networks originated from Daraiseh's work [Al 06] and has been consistently improved to its most prominent representative, MO-GART. MO-GART is multi-objective evolved ART, employing automatic adaptation of the GA parameters, and appropriate data-sampling for the evaluation of the fitness function.

In the following we present the MO-GART approach in detail, compare it with other competitive classifiers, such as SVM and CART, and offer some directions for further research.

7.1 Putting It All Together: MO-GART

In this section the final product of this dissertation is described. The following lists the proposed algorithm completely. The algorithm starts by generating an initial population, $P(0)$, of ARTMAP networks (FAM, EAM or GAM), each one of them trained with a different value of the baseline vigilance parameter $\bar{\rho}_a$, and order of training pattern presentation. The networks are encoded into chromosomes where each component (gene)

represents a category (hidden node) of an ART network. Each component contains the weight information for the category. The chromosomes are variable length, where the length is equal to the number of categories in the network represented by the chromosome (see Figures 3.12, 3.13, 3.14).

$P(0) \leftarrow$ Generate-Initial-Population();
$A(0) \leftarrow$ Initialize-Empty-Archive();
for $t \leftarrow 1$ **to** Gen_{max} **do**
    Evaluation();
    Update-Archive($P(t)$, $A(t)$);
    if *stopping criteria met* **then** exit for;
    $P'(t) \leftarrow$ Selection($P(t)$, $A(t)$);
    $P(t) \leftarrow$ Reproduction($P'(t)$);
end
return $A(t)$;

Figure 7.1: Pseudo-code of MO-GART Algorithm

Also, MO-GART initializes an empty secondary population, $A(0)$, that will be used to store nondominated solutions found during the evolution.

In each generation, each solution in the population is evaluated according to each objective function. That is, the error rate of each ARTMAP network is evaluated by running it against a validation set. The error rate is the misclassification rate of the classifier (individual). In the first generation, all available validation patterns are used. In all other iterations, the number of validation patterns to be used, denoted as n_v, is calculated using the following formula:

$$n_v = (\frac{z_{\alpha/2}}{Halfwidth})^2 \hat{p}_{err}(1 - \hat{p}_{err}) \tag{7.1}$$

In this work, α is set at 95% and the Halfwidth is chosen to be $\pm 1\%$ of error rate. Furthermore, $\hat{p}_{err}$ is set as the best error rate observed in the previous generation. Then, n_v patterns are sampled randomly from the validation set. The misclassification rate is used as an estimate for the error rate of the classifier.

The second objective, complexity, is represented by the number of categories present in each network. Once networks in population P are evaluated, the archive A is updated by adding to it the solutions in P that are nondominated by solutions in A. Also, solutions in A that are now dominated by solutions just added from P, are removed from the archive A.

The algorithm runs for a maximum number of generations defined by Gen_{max}. However, to avoid running MO-GART for unnecessarily large number of generations, the evolution is also stopped when the archive A is not updated for 10 consecutive generations.

The selection process creates a temporary population P', where the parent chromosomes used to create the next generation are selected. The chromosomes in the archive A and population P are assigned fitness values based on dominance relationship. In this scheme each individual is assigned a strength value that is equal to the number of solutions it dominates. After that, a raw fitness, $R(x)$, is assigned for each individual to be the sum of the strengths of all its dominators in both A and P. The raw fitness is then adjusted as follows. For each individual, x, the distance, in objective space, to the k-th nearest neighbor is found and denoted as $\sigma_k(x)$. The value of k is chosen to be

the square root of the sum of the size of the archive and population. The fitness of each individual is then calculated using the following equation:

$$Fit(x) = R(x) + \frac{1}{\sigma_k + 2} \tag{7.2}$$

The parents are then chosen using a deterministic binary tournament selection with replacement, as follows: For each parent, randomly select two chromosomes from the combined set of A and P, and choose, the chromosome with the smallest fitness value. Boundary solutions, which are networks with smallest error rate and smallest size, are ensured to be copied in the set of parents.

The confidence factor is calculated for every individual in the population, for every category j of the $p-th$ ART network, that is mapped to label k, as follows:

$$CF_j^k(p) = 0.5A_j^k(p) + 0.5S_j^k(p) \tag{7.3}$$

where $A_j^k(p)$ is a measure of accuracy of classification achieved by category j, in the $p-th$ network, that is mapped to label k. Furthermore, $S_j^k(p)$ is a measure of probability of selection of category j in the $p-th$ network, that is mapped to label k.

The accuracy measure, $A_j^k(p)$, is defined as follows: the probability of correct classification for category j divided by the maximum probability of correct classification for any category in the same network ($p-th$ network) that predicts the same class label, k. This measure assumes higher values for categories that are performing relatively well. In particular, if the number of validation samples that selected this category, and were

correctly classified by it, is denoted by $P_j^k(p)$, and the number of validation samples that selected this category is denoted by $C_j^k(p)$, then,

$$A_j^k(p) = \frac{P_j^k(p)/C_j^k(p)}{\max_j(P_j^k(p)/C_j^k(p))} \tag{7.4}$$

Also $S_j^k(p)$ is defined as the probability of selection by the validation patterns of a category, j, of the $p-th$ network, that is mapped to label k. The probability of selection of category j, of the $p-th$ network, that is mapped to label k, is the number of validation patterns that selected this category, $C_j^k(p)$, divided by the maximum number of patterns $C_{j_{max}}^k(p)$ that selected any category j that predicts the same classification label, k, for the $p-th$ network:

$$S_j^k(p) = C_j^k(p)/C_{j_{max}}^k(p) \tag{7.5}$$

This measure achieves higher values for categories that were selected more often using the validation patterns. The scaling ensures that $A_j^k(p) \in [0,1]$, $S_j^k(p) \in [0,1]$ and therefore $CF_j(p) \in [0,1]$. In addition, in every network, at least one category has $A_j(p) = 1$, and at least one category (but not necessarily the same) has $S_j(p) = 1$.

Afterwards, with probability of $1 - CF_j^k(p)$, categories are delete from every chromosome in the temporary population, $P'(t)$. Also, every chromosome in P' gets mutated using Gaussian random number of mean of 0 and standard deviation of $SF_j^k(p) = 0.05(1 - CF_j^k(p))$. The mutation is applied as described below:

- In *MO-GFAM*, for each category, either its u or v endpoint is selected randomly (with 50% probability) and then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

- In *MO-GEAM*, for each category, every component of the ellipsoidal center m gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Furthermore, the mutated category's axis ratio μ or radius r is selected with 50% probability. We add a small number, to the axis ratio or the radius. The small number is drawn from a Gaussian distribution. However if μ, or r, due to mutation, become larger than 1, they are set back to the value of 1, while if they become smaller than zero we set their value to 0.0001.

- In *MO-GAM*, for each category, either its mean vector m, or standard deviation vector σ is selected randomly (50% probability). Then every component of this selected vector is mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

The chromosomes in $P(t)$ are then replaced by chromosomes created by performing crossing over pairs of parents in $P'(t)$ as follows: For each parent, p, p', a random cross-over point is chosen, designated as n, n', respectively. Then, all the categories with index

Table 7.1: Performance of MO-GART on some data sets. Most accurate network reported.

	MO-GFAM			MO-GEAM			MO-GGAM		
	PCC	Size	Traing Time	PCC	Size	Traing Time	PCC	Size	Traing Time
1Ci/Sq	97.43	30	5.90	99.60	4	16.51	99.60	2	6.75
G4C-25	75.56	4	3.41	75.40	4	9.12	75.70	4	4.12
G6C-15	84.83	6	2.76	84.17	6	6.57	84.71	6	4.94
glass	76.00	7	0.16	73.33	7	0.23	76.00	12	0.23
Iris	94.67	2	0.74	94.92	2	1.90	94.79	2	1.23
page	96.59	5	1.34	93.94	5	5.34	95.09	6	2.55
pendigits	98.07	220	200.22	98.73	704	825.48	98.97	238	72.99
pima	84.00	6	0.43	82.67	3	0.39	83.33	3	0.20
sat	88.87	202	98.93	87.97	388	251.22	88.99	124	50.23
seg	95.29	27	9.67	95.14	327	31.82	92.00	56	6.56
wav	84.80	3	17.85	86.00	3	64.59	87.10	7	7.67

greater than n' in the chromosome p' and all the categories with index less than or equal to index n in the chromosome with index p are moved into an empty chromosome within the new generation. Notice that crossover is done at level 1 of the chromosome. This operation is pictorially illustrated in Figure 3.15.

7.2 Evaluation of The Approach

This section lists the results of running the algorithm listed in the previous section for a number datasets. The results were obtained using 11 datasets. Details about these datasets can be found in Appendix B. The results are summarized in Table 7.1 and they include the three performance measures of interest in this dissertation: accuracy of the classifier, complexity and training CPU time.

When the results in Table 7.1 are compared to those in Chapter 6, it is easy to notice the gain in efficiency achieved through this combined approach. For example, the training time for the Pendigits dataset was reduced from 462 seconds to 200 seconds.

Table 7.2 shows the results obtained using SVM and CART classifiers. These results were obtained by other members of the Machine Learning Lab at the University of Central Florida. Comparing the results in Tables 7.2 and 7.1 one can make the following observations:

- The accuracy of the overall MO-GART networks is very competitive with the SVM and CART. It can be seen that MO-GART and SVM have a significant advantage in terms of network accuracy when compared to CART.

- The training time of CART is very small compared to MO-GART and SVM. However, MO-GART has a significant advantage over SVM in terms of training time.

- MO-GART has a significant advantage in terms of network complexity when compared to SVM and CART. The advantage can be better shown by considering other Pareto networks that MO-GART produced (not shown in Table 7.1). For example, for the Pendigits dataset, MO-GGAM can achieve accuracy of 97% using 90 categories, where CART used 109 nodes for a 93% accurate tree. Another example is the Pima dataset. MO-GFAM was able to achieve 77% accuracy using 2 categories, where CART achieved 73% for the same number of nodes. The availability of multiple Pareto networks to choose from (in addition to the most accurate one) is an additional advantage of the proposed approach over SVM and CART.

Table 7.2: Performance of SVM and CART on some data sets

	SVM			CART		
	PCC	Size	Training Time	PCC	Size	Training Time
1Ci/Sq	99.67	88	136.41	97.57	28	0.02
G4C-25	75.24	264	42.09	73.50	4	0.00
G6C-15	84.83	450	37.93	80.42	6	0.02
glass	65.63	63	0.39	64.06	4	0.00
Iris	95.06	76	20.04	94.02	4	0.00
page	95.30	150	20.49	93.84	7	0.02
pendigits	99.54	929	616.88	93.37	109	0.28
pima	73.71	64	1.24	73.71	2	0.00
sat	90.25	1081	263.78	84.35	22	0.30
seg	97.29	230	27.28	93.43	17	0.05
wav	87.45	574	74.28	75.20	14	0.09

The comparison of GART, SVM and CART provided above, is fair because it used the same databases and datasets per database for training, validation and testing of these architectures.

The accuracy of C4.5, as listed in [DIE00], for the Satellite dataset is 84.9%, for the Segmentation dataset is 96.8%, and for the Waveform dataset is 76.6%. As it can be seen from Table 7.1 MO-GART was able to achieve competitive accuracy. In [KBS97] the authors use the simple Bayes classifier to produce classification accuracy of 85.20% on satellite, 93.12% on segmentation, 78.57% on waveform, 70.11% on glass and 75.9% on pima. It can be seen from Table 7.1 that MO-GART was able to achieve better network accuracy on all these problems. Furthermore, in [YA01], the authors use a decision tree variant to produce classification accuracy of 92% (size: 37) on segmentation, 83% (size: 65) on waveform, 60% (size 14) on glass, 75.2% (size: 3.4) on pima and 95% (size: 37) on pendigits. As it is evident from Table 7.1 MO-GART results consistently outperform these results.

In [LLS00], the authors compared the accuracy and size of a 33 classifiers belonging to the tree, statistical and neural types classifiers. Three of the datasets that Lim, Loh and Shih have experimented with are the Satellite, the Segmentation and the Waveform datasets that has been tested in Table 7.1. The MO-GFAM results on the Satellite dataset are: 88.87 classification accuracy, needing 202 categories (other MO-GFAM solutions include: 84.68% with 7 categories, 86.40% with 21 categories). The MO-GGAM results on the Satellite dataset are 88.99 with 124 categories. The accuracy results reported on the Satellite dataset by [LLS00] are: Minimum classification accuracy of 60% and maximum classification accuracy of 90%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 8, while the median tree size was 63. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 63 and 216. The MO-GFAM results on the Segmentation dataset are: 95.29% classification accuracy, needing 27 categories. The accuracy results reported on the Segmentation dataset by [LLS00] are: Minimum classification accuracy of 48% and maximum classification accuracy of 98% (achieved by the nearest neighbor classifier, which performs no data compression). Furthermore the tree type classifiers (22 of them) created a minimum tree size of 6, while the median tree size was 39. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 69 and 42. The MO-GFAM results on the Waveform dataset are: 84.8% classification accuracy, needing 3 categories for MO-GFAM and 87.1% and 7 categories for MO-GGAM. The accuracy results reported on the Waveform dataset by [LLS00] are: Minimum classification accuracy of 52% and maximum classification accuracy of 85%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 3, while the median tree size was

16. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 14 and 54.

7.3 Discussion and Future Work

In the conclusion of this work a number of final remarks are made about the work that has been done. Also a number of areas of improvements are identified that have significant promise to build on and improve the work achieved in this dissertation.

1. The evolution of ART architectures has significantly improved the network accuracy and size of these networks. On the other hand, the training time of the evolved ART networks (MO-GFAM, MO-GEAM, or MO-GGAM) is significantly larger than that of ART (FAM, EAM, or GAM). However, to be able to obtain good performance from ART, the analyst must experiment with different parameter settings. It was shown in this dissertation that the GA is more efficient at finding the optimal network than experimenting with the parameters of ART networks. In addition, the genetic optimization of ART may achieve performance that might not be attainable by original ART training rules. Genetic operators allow mixing of networks (Crossover), reducing the size (Pruning), and altering categories (Mutation).

2. The accuracy of the evolved ART architecture is limited by the maximum possible accuracy that can be achieved by the ART structure. In other words, if the shape of the categories puts a limit on the accuracy of ART for a given classification task,

then the evolution of ART would not be able to exceed that limit, given a certain network size. For example, for 1Ci/Sq problem, FAM (or evolved FAM) would not be able to achieve the accuracy of GAM or EAM using the same number of categories. Also, GAM has a disadvantage with the SEG dataset.

3. The genetic search will lead to an optimal solution as long as the genetic operators (crossover, pruning and mutation) allow the formation of any possible network. In other words, if every possible network (solution) is reachable, then the genetic search will eventually find the optimal solution (where optimality is defined as the maximum accuracy achievable with the ART structure, using a certain number of categories, for a given classification task). However, this guarantee is only theoretical in the sense that the user of the algorithm will have a limited amount of time (or computational budget) for the GA to converge. Therefore, techniques that improve the search efficiency of the genetic algorithms are essential for its viability.

4. The evaluation relaxation technique proposed in this dissertation might not be effective when the number of validation points is small to start with, or when the classification accuracy achieved is poor because the problem is hard. For example, using Equation (7.1), it can be seen the n_v is 1824 when the error rate is 95% and 9604 when the error rate is 50%. This is when a half width of 0.01 and a confidence level of 95% are considered. Therefore, for harder problems, it might be necessary to accept a wider half width to be able to converge using a small CPU run time.

5. The adaptation technique proposed in this dissertation was shown that it improves the success ratio of the genetic operators and the overall performance of the genetic

algorithm. The proposed approach gives higher probability of invoking an operator to under-performing categories. The adaptation mechanism is related to the important issue of the exploration ability of the genetic algorithm. When the genetic algorithm reaches close to the optimal solution and the confidence factors of the categories become high, the probabilities of invoking these operators become very low. This might cause small change from one generation to the next, which will trigger the stopping criteria. This in turn might cause premature convergence in some cases. It might be beneficial to have a mechanism that detects such situations and temporarily increases the operator probabilities for some individuals to keep the search process alive.

6. A very promising benefit of this research might be achieved by applying the genetic algorithm techniques developed in this dissertation to other neural network architectures such PNN, GRNN and others. In particular, the evaluation relaxation techniques can be readily applied to wide range of evolutionary neural networks. The adaptation mechanism can be also applied if a suitable confidence factor is found for other architectures. Also the multiobjective technique that was shown to work well for evolving ART networks can be used to evolve other neural network architectures. What is also important, is that the methodologies that were used to evaluate each of the techniques can be reused to develop similar evolutionary algorithms that are applied to neural network architectures other than ART. These methodologies refer to the way default parameters were found in Section 3.3, the

way genetic operators were evaluated in Section 4.5, and the way sets of Pareto solutions are compared in Section 6.5.

7. In the evaluation relaxation technique proposed in Chapter 5, it was shown that taking a small subset of the validation set might be beneficial to the performance of the genetic algorithm. It is worthwhile to refine this research to find a lower limit of the number of validation points to be used to evaluate individuals in the genetic algorithm.

8. Also the evaluation relaxation technique proposed in Chapter 5 uses a problem approximation approach. There are other approaches that have been identified in the literature such as the evolutionary approximation techniques and in particular the fitness inheritance techniques. Fitness inheritance estimates the fitness of some individuals using their parents. This technique has the promise of significantly improving the performance of the Genetic Algorithm. However, the challenge here is the development of proper surrogate functions that are used to estimate the fitness of the offspring individuals based on the fitness of their parents.

9. In this work, to generate the initial population of ART networks, the vigilance parameters and the order of pattern presentation were varied. It might be beneficial to vary the category prediction error tolerance in ssART (see [ABG03]) and also the number of epochs used for training the initial population. This allows for greater diversity in the initial population which might help find better solutions.

10. There is a need to implement a post-MOEA phase where, from the optimal networks found, one is chosen as the representative winner network. Different criteria can be used. It is beneficial to add such capability to the network architectures proposed in this work as in many situations it might be the case where the end user is looking for one solution network for the classification problem.

11. The use of hybrid-GAs has been applied successfully for a number of applications. This can be beneficial in terms of finding the optimal solutions faster and also finding better solutions. For example in the case of evolving ART networks, it might be beneficial to allow the ART rules optimize some networks in the population by performing a local search.

7.4 Summary

The final product of this dissertation is a new family of ART-based architectures, called MO-GART, that are optimized using an elegant evolutionary approach. The introduced architectures are able to produce classifiers of good accuracy and small size, using a reasonable computational budget. They also have the advantage of requiring little user intervention because the algorithm parameters are automatically adapted. After training, the proposed algorithm produces a number of optimal classifiers, with varying levels of tradeoff between accuracy and size, allowing the user to make an informed decision as to which classifier to deploy for a given classification task.

A number of improvements were incrementally made to the genetic ART architecture proposed in [Al 06]. Each of these improvements were carefully justified and thoroughly evaluated. It was shown that each of these improvements were worthwhile, and are necessary to make the proposed architectures competitive, not only within the ART family, but also among other well-known classifiers.

MO-GART was compared to a number of classifiers. It was found that MO-GART can achieve superior classification accuracy when compared to other ART architectures. It was also found that MO-GART can achieve competitive classification accuracy when compared to other classifiers such as SVM, CART and C4.5. Unlike most classification algorithms, MO-GART focuses on finding the simplest classifier that can achieve a certain classification accuracy for a given classification task. In fact, to address this issue properly, the MO-GART algorithm produces a number of networks that achieve varying levels of accuracy and complexity. Another issue that influenced the design of MO-GART is the independence of its performance on user-defined parameters. The importance of this independence is the enhanced ability to use this algorithm for a wide range of classification tasks, and without the need to have significant experience with it. However, these advantages come at a cost in terms of the computational time needed to train MO-GART. Since other algorithms need the user to experiment with their parameters to find the set of values that gives the best results, it is fair to compare the training time of MO-GART to the time needed to run enough experiments using other algorithms. The claim in this dissertation is that a well-designed genetic search is more efficient at finding the optimal network compared to experimentation with parameters. Improving the efficiency of the

genetic search for the optimal ART network has occupied a significant portion of this dissertation. The MO-GART family of classifiers has competitive classification accuracy, exceptionally small size, and is able to find multiple optimal networks more efficiently than some other popular classification algorithms.

APPENDIX A
NOTATION

Table A.1: List of Notation

Symbol	Definition
λ	Population size in genetic algorithms. (Also referred to as Pop_{size} in some cases.)
Gen_{max}	Maximum number of generations of the genetic algorithm
G_{conv}	Number of generations until convergence
NC_{best}	The number of solution kept for elitism
$Pr(Mut)$	Probability of mutation
$Pr(Cat_{del})$	Probability of deletion
$Fit(p)$	Fitness of the $p-th$ solution
ρ_a	Vigilance parameter in ART
N_a	Number of categories in ART network
PCC	Percent of correct classification

APPENDIX B
DATASETS

In this work, experiments were conducted on 33 datasets, of which 20 are simulated datasets and 13 are real datasets. Each dataset was randomly divided into three subsets: training, validation and testing. The summarized specifics of each one of the datasets are depicted in Table B.

- **Gaussian Datasets (# 1-12):** These are 12 artificially generated, 2-dimensional, and Gaussianly distributed datasets, belonging to 2-class, 4-class, and 6-class problems, with 5%, 15%, 25%, and 40% overlap between the classes. Note that 5% overlap means that the optimal Bayesian Classifier would have 5% misclassification rate on the Gaussianly distributed data. There are a total of 3x4=12 Gaussian datasets. These datasets are named as **G#c-##** where the first number is the number of classes and the second number is the percentage of class overlap. For example, G2c-05 means that the Gaussian dataset is a 2-class and 5% overlap dataset.

- **Structures within a Structure Datasets (# 13 - 19):** These are artificial datasets that were inspired by the circle - in the - square problem. This problem has been extensively examined in the neural networks literature. Seven different datasets were generated by changing the structures (type, number and probability) that we were dealing with. The data-points within each structure of these artificial datasets are uniformly distributed within the structure. The number of points within each structure is chosen in a way that the probability of finding a point within this structure is equal to a pre-specified number.

Table B.1: Datasets used for experimentation, and their characteristics

	Database Name	# Training Instances	# Validation Instances	# Test Instances	# Attributes	# Classes	% Major Class
1	G2c-05	500	5000	5000	2	2	50.00%
2	G2c-15	500	5000	5000	2	2	50.00%
3	G2c-25	500	5000	5000	2	2	50.00%
4	G2c-40	500	5000	5000	2	2	50.00%
5	G4c-05	500	5000	5000	2	4	25.00%
6	G4c-15	500	5000	5000	2	4	25.00%
7	G4c-25	500	5000	5000	2	4	25.00%
8	G4c-40	500	5000	5000	2	4	25.00%
9	G6c-05	504	5004	5004	2	6	16.67%
10	G6c-15	504	5004	5004	2	6	16.67%
11	G6c-25	504	5004	5004	2	6	16.67%
12	G6c-40	504	5004	5004	2	6	16.67%
13	4Ci/Sq	2000	5000	3000	2	5	20.00%
14	4Sq/Sq	2000	5000	3000	2	5	20.00%
15	1Ci/Sq	2000	5000	3000	2	2	50.00%
16	1Ci/Sq/70:30	2000	5000	3000	2	2	70.00%
17	5Ci/Sq	2000	5000	3000	2	6	16.67%
18	2Ci/Sq/75:20:5	2000	5000	3000	2	3	75.00%
19	2Ci/Sq/50:30:20	2000	5000	3000	2	3	50.00%
20	MOD-IRIS	500	4800	4800	2	2	50.00%
21	ABALONE	501	1838	1838	7	3	33.30%
22	PAGE	500	2486	2487	10	5	89.80%
23	Optdigits	1823	2000	1797	64	10	10.00%
24	Pendigits	4494	3000	3498	16	10	10.00%
25	Sat	2000	2436	2000	36	6	24.19%
26	Seg	800	810	700	19	7	14.29%
27	wav	1000	2000	2000	21	3	33.33%
28	shuttle	3000	1000	54000	9	5	80.00%
29	glass	75	75	64	9	6	35.51%
30	pima	150	150	232	7	2	66.70%
31	Letter	3000	1000	16000	16	26	4.07%
32	Vehicle	250	250	346	18	4	25.70%
33	BUPA	100	100	145	6	2	42.00%

– **4Ci/Sq (# 13):** This is a four circle in a square problem, a five class classification problem. The probability of finding a data point within a circle or inside the square and outside the circles is equal to 0.2.

– **4Sq/Sq (# 14):** This is a four squares in a square problem, a five class classification problem. The probability of finding a data point within an inner square or inside the outer square and outside the inner squares is equal to 0.2.

– **1Ci/Sq (# 15):** This is a one circle in a square problem, a two class classification problem. The probability of finding a data point within a circle or inside the square and outside the circle is equal to 1/2. The sizes of the areas in the circle and outside the circle and inside the square are the same. This is the benchmark circle in the square problem.

– **1Ci/Sq/30:70 (# 16):** This is a one circle in a square problem, a two class classification problem. The probability of finding a data point within a circle or inside the square and outside the circle is equal to 0.3 and 0.7, respectively. The sizes of the areas in the circle and outside the circle and inside the square are 0.3 and 0.7, respectively. This is a modified version of the circle in the square problem.

– **5Ci/Sq (# 17):** This is a five concentric circles in a square problem, a six class classification problem. The probabilities of finding a data point in the distinct regions created by these structures are equal.

– **2Ci/Sq/75:20:5 (# 18):** This is two circles in a square problem, a three class classification problem. One of the circles is smaller than the other. The

probability of finding a data point within the small circle, the large circle, and outside the circles but inside the square is 0.75, 0.20, and 0.05, respectively.

  - **2Ci/Sq/20:30:50 (# 19):** This is two circles in a square problem, a three class classification problem. One of the circles is smaller than the other. The probability of finding a data point within the small circle, the large circle, and outside the circles but inside the square is 0.2, 0.3, and 0.5, respectively.

- **Modified Iris (MOD-IRIS) Dataset (# 20):** This is a modified version of the IRIS dataset from the UCI repository (see [NHB98]). The original IRIS dataset has 150 data-points and 3 classes. The data corresponding to the class that is linearly separable was eliminated. This left 100 data-points and 2 classes. From the four input attributes of the original IRIS dataset only two attributes (attribute 3 and 4) were used because they seem to have enough discriminatory power to separate the 2-class data. Finally, in order to create a reasonable size dataset from these 100 points, data was generated by adding noise around each one of these 100 data-points (the noise was Gaussian of zero mean and small variance) to end up with approximately 10,000 points.

- **Modified Abalone (ABALONE) Dataset (# 21):** This dataset is originally used for prediction of the age of an abalone (see [NHB98]). It contains 4177 instances, each with 7 numerical attributes, 1 categorical attribute, and 1 numerical target output (age). This dataset was modified by discarding the categorical attribute, and grouping the target output values into 3 classes: 8 and lower (class 1),

9-10 (class 2), 11 and greater (class 3). This grouping of output values has been reported in the literature before.

- **Page Blocks (PAGE) Dataset (# 22):** This database represents the problem of classifying the blocks of the page layout in a document (see [NHB98]). It contains 5473 examples coming from 54 distinct documents. Each example has 10 numerical attributes (e.g., height of the block, length of the block, eccentricity of the block, etc.,) and one target (output) attribute, representing the type of the block (text, horizontal line, graphic, vertical line, and picture). One of the noteworthy points about this database is that its major class (text) has a high probability of occurring (above 80%). This dataset has five classes, four of them make only 10% of the total instances.

- **Optdigits (OPT) Dataset (# 23):** This dataset has vectors representing normalized bitmaps of handwritten digits from a preprinted form (see [NHB98]). The bitmaps were normalized using preprocessing programs. From a total of 43 people, 30 contributed to the training set and the remaining 13 to the test set. 32x32 bitmaps are divided into non-overlapping blocks of 4x4 and the number of pixels is counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0 to 16. This dataset has 64 attributes and 10 classes. The training set has 3823 records and the test set has 1797 records. In this work, the original training set was divided into a training set of 1823 records and a validation set of 2000 records.

- **Pendigits (PEN) Dataset (# 24):** This dataset has records representing handwritten digits (see [NHB98]). The dataset was created by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training and cross-validation and writer dependent testing, and the digits written by the remaining 14 writers are used for writer independent testing. This dataset has 16 attributes and 10 classes. The training set has 7494 records and the test set has 3498 records. In this work, the original training set was divided into a training set of 4494 records and a validation set of 3000 records.

- **Satellite Image (SAT) Dataset (# 25):** This dataset gives the multi-spectral values of pixels within 3x3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood (see [NHB98]). The aim is to predict the classification given the multi-spectral values. There are six classes and thirty-six numerical attributes. The training set consists of 4435 records while the test set consists of 2000 records. The original training set was divided into a training set of 2000 records and a validation set of 2435.

- **Image Segmentation (SEG) Dataset (# 26):** This dataset was used in the StatLog Project (see [NHB98]). The samples are from a dataset of seven outdoor images. The images are hand-segmented to create a classification for every pixel as one of brickface, sky, foliage, cement, window, path, or grass. There are seven classes, nineteen numerical attributes and 2310 records in the dataset.

- **Waveform (WAV) Dataset (# 27):** This is an artificial three-class problem based on three wave-forms (see [NHB98]). Each class consists of a random convex

combination of two waveforms sampled at the integers with noise added. There are twenty-one numerical attributes, and 3000 records in the training set. Error rates are estimated from an independent test set of 2000 records. The original training set was divided into a training set of 1000 records and a validation set of 2000 records.

- **Shuttle (SHU) Dataset (# 28):** This dataset contains 9 attributes all of which are numerical and five classes. Approximately 80% of the data belongs to one class (see [NHB98]). The training set has 43500 records and test set has 14500 records. The original training set was divided into a training set of 3000 records and a validation set of 1000 records. The rest were added to the test set.

- **Glass (GLS) Dataset (# 29):** This dataset is used to classify types of glass (see [NHB98]). It was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence, if it is correctly identified. This dataset has 214 instances, 10 numerical attributes and 6 classes.

- **Pima-Indian Diabetes (PIMA) Dataset (# 30):** This dataset classifies the patients that are females, at least twenty-one years old, of Pima Indian heritage and living near Phoenix, Arizona, USA (see [NHB98]). The problem is to predict whether a patient would test positive for diabetes given a number of physiological measurements and medical test results. There are 2 classes, 8 numerical attributes, and 768 records. However, many of the attributes, such as serum insulin, contain zero values which are physically impossible. These removed the serum insulin and

records that have impossible values in other attributes, resulting in 7 attributes and 532 records (this approach was followed by other researchers).

- **Letter Image Recognition (Letter) Dataset (# 31):** This dataset classifies the letter categories associated with vectors of 16 integer attributes extracted from raster scan images of the letters (see [NHB98]). There are 26 classes, 16 numerical attributes, and 20,000 records.

- **Vehicle Silhouettes (Vehicle) Dataset (# 32):** This dataset is used to to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette (see [NHB98]). There are 4 classes, 18 numerical attributes, and 846 records.

- **BUPA Liver Disorders (BUPA) Dataset (# 33):** This dataset classifies male individuals for liver disorders based on a number of blood tests and alcoholic beverages consumed (see [NHB98]). There are 2 classes, 6 numerical attributes, and 345 records.

LIST OF REFERENCES

[ABG03] G. C. Anagnostopoulos, M. Bharadwaj, M. Georgiopoulos, S. J. Verzi, and G. L. Heileman. "Exemplar-based pattern recognition via semi-supervised learning." In *Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN '03)*, volume 4, pp. 2782–2787, Portland, Oregon, USA, 2003.

[AG02] G.C. Anagnostopoulos and M. Georgiopoulos. "Category Regions as new geometrical concepts in Fuzzy ART and Fuzzy ARTMAP." *Neuarl Networks*, **15**(10):1205–1221, 2002.

[AGV02] G.C. Anagnostopoulos, M. Georgiopoulos, S.J. Verzi, and G.L. Heileman. "Reducing generalization error and category proliferation in ellipsoid ARTMAP via tunable misclassification error tolerance: boosted ellipsoid ARTMAP." In *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, volume 3, pp. 2650–2655, Honolulu, Hawaii, January 2002.

[Al 06] A. Al-Daraiseh. *Genetically Engineered Adaptive Resonance Theory (ART) Neural Network Architectures.* PhD thesis, University of Central Florida, Orlando, May 2006.

[Ana01] G.C. Anagnostopoulos. *Novel Approaches in Adaptive Resonance Theory for Machine Learning.* PhD thesis, University of Central Florida, Orlando, May 2001.

[ASP94] Peter J. Angeline, Gregory M. Saunders, and Jordan P. Pollack. "An evolutionary algorithm that constructs recurrent neural networks." *IEEE Transactions on Neural Networks*, **5**(1):54–65, January 1994.

[Bac92] Thomas Back. "Self-adaptation in genetic algorithms." In *Proceedings of the First European Conference on Artificial Life*, pp. 263–271, Washington DC, 1992. MIT Press.

[Bak85] James E. Baker. "Adaptive Selection Methods for Genetic Algorithms." In *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 101–111, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[BDH96] Jerzy Bala, Kenneth DeJong, J. Huang, Haleh Vafaie, and Harry Wechsler. "Using Learning to Facilitate the Evolution of Features for Recognizing Visual Concepts." *Evolutionary Computation*, **4**(3):297–311, 1996.

[BFO84] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[BT95] Tobias Blickle and Lothar Thiele. "A Comparison of Selection Schemes Used in Genetic Algorithms." Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[BV97] A. Burton and T. Vladimirova. "Utilisation of an adaptive resonance theory neural network as a genetic algorithm fitness evaluator." In *Proceedings of the 1997 IEEE International Symposium on Information Theory*, p. 209, Ulm, Germany, 1997. IEEE, New York, NY.

[Can02] Erick Cantu-Paz. "Feature Subset Selection By Estimation Of Distribution Algorithms." In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 303–310, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[CG87a] G. A. Carpenter and S. Grossberg. "ART 2: Self-organization of stable category recognition codes for analog input patterns." *Applied Optics*, **26**(23), 1987.

[CG87b] G. A. Carpenter and S. Grossberg. "A massively parallel architecture for a self-organizing neural pattern recognition machine." *Computer Vision, Graphics, and Image Processing*, **37**:54–115, 1987.

[CGM92] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. "Fuzzy ARTMAP: A Neural Network Architectur for Incremental Supervised Learning of Analog Multidimensional Maps." *IEEE Transactions on Neural Networks*, **3**:698–713, 1992.

[CGR91a] G. A. Carpenter, S. Grossberg, and J. H. Reynorlds. "ARTMAP: Supervised real-time learning and classification of non-stationary data by a self-organizing neural network." *Neural Networks*, **6**:565–588, 1991.

[CGR91b] G. A. Carpenter, S. Grossberg, and D. B. Rosen. "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system." **4**(6):759–771, 1991.

[CMN98] G. A. Carpenter, B. L. Milenova, and B. W. Noeske. "Distributed ARTMAP: A neural network for fast distributed supervised learning." *Neural Networks*, **11**(5):793–813, 1998.

[Coe00] Carlos A. Coello Coello. "An updated survey of GA-based multiobjective optimization techniques." *ACM Comput. Surv.*, **32**(2):109–143, 2000.

[Coe06] C.A. Coello Coello. "Evolutionary multi-objective optimization: a historical view of the field." *IEEE Computational Intelligence Magazine*, **1**(1):28–36, Feb. 2006.

[CT95] G. A. Carpenter and H. A. Tan. "Rule extraction: From neural architecture to symbolic representation." *Connection Science*, **7**:3–27, 1995.

[DAP] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II." In *Proceedings of the Parallel Problem Solving from Nature VI Conference*.

[DD97] I. Das and J.E. Dennis. "A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems." *Structural Optimization*, **14**:63–69, 1997.

[DG89] Kalyanmoy Deb and David E. Goldberg. "An Investigation of Niche and Species Formation in Genetic Function Optimization." In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[DIE00] THOMAS G. DIETTERICH. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization." *Machine Learning*, **40**:139–157, 2000.

[DPA02] Kalyanmoy Deb, Amrit Pratab, Samir Agrawal, and T. Meyarivan. "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II." *IEEE Transactions On Evolutionary Computation*, **6**(2):182–197, Apr 2002.

[EMH01] Mark Erickson, Alex Mayer, and Jeffrey Horn. "The Niched Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems." In *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, pp. 681–695, London, UK, 2001. Springer-Verlag.

[FAF95] Lawrence J. Fogel, Peter J. Angeline, and David B. Fogel. "An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines." In *Evolutionary Programming*, pp. 355–365, 1995.

[Fer05] Konstantinos P. Ferentinos. "Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms." *Neural Networks*, **18**(7):934–950, 2005.

[FES03] J. E. Fieldsend, R. M. Everson, and S. Singh. "Using Unconstrained Elite Archives for Multiobjective Optimization." *IEEE Transactions on Evolutionary Computation*, **7**(3):305–323, june 2003.

[FF93] Carlos M. Fonseca and Peter J. Fleming. "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization." In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 416–423, San Mateo, CA, 1993. Morgan Kaufmann.

[FF95] Carlos M. Fonseca and Peter J. Fleming. "An Overview of Evolutionary Algorithms in Multiobjective Optimization." *Evolutionary Computation*, **3**(1):1–16, 1995.

[FG88] J. Michael Fitzpatrick and John J. Grefenstette. "Genetic Algorithms in Noisy Environments." *Machine Learning Journal*, **3**(2-3):101–120, October 1988.

[Fog89] Terence C. Fogarty. "Varying the probability of mutation in the genetic algorithm." In *Proceedings of the third international conference on Genetic algorithms*, pp. 104–109, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[Fog93] D. B. Fogel. "Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe." In *Proc. Amer. Power Conf.*, pp. 875–879, 1993.

[FS05] J. E. Fieldsend and S. Singh. "Pareto evolutionary neural networks." *IEEE Transactions on Neural Networks*, **16**(2):338–354, march 2005.

[FW02] Xiuju Fu and Lipo Wang. "A GA-based RBF classifier with class-dependent features." In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pp. 1890–1894, Washington, DC, USA, 2002. IEEE Computer Society.

[GD91] D. Goldberg and K. Deb. "A comparative analysis of selection schemes used in genetic algorithms." *In Rawlins G. J. E. (Ed.) Foundations of Genetic Algorithms*, pp. 69–93, 1991.

[GDC] Eduardo Gomez-Sanchez, Yannis Dimitriadis, Jose Cano-Izquierdo, and Juan Lopez-Coronado. "MicroARTMAP: Use of Mutual Information for Category Reduction in Fuzzy ARTMAP." *IEEE Transactions on Neural Networks.*

[GDC92] D. E. Goldberg, K. Deb, and J.H. Clark. "Genetic Algorithms, Noise, and the Sizing of Populations." *Complex Systems*, **6**:333–362, 1992.

[GDC01] E. Gomez-Sanchez, Y.A. Dimitriadis, J.M. Cano-Izquierdo, and J. Lopez-Coronado. "Safe-μARTMAP: A new solution for reducing category proliferation in Fuzzy ARTMAP." In *Proceedings of the 2001 International Joint Conference on Neural Networks (IJCNN '01)*, volume 2, pp. 1197–1202, Washington, DC, USA, 2001.

[GHH94] Michael Georgiopoulos, Juxin Huang, and Gregory L. Heileman. "Properties of learning in ARTMAP." *Neural Networks*, **7**(3):495–506, 1994.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[GR87] David E. Goldberg and Jon Richardson. "Genetic algorithms with sharing for multimodal function optimization." In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pp. 41–49, Mahwah, NJ, USA, 1987. Lawrence Erlbaum Associates, Inc.

[Gro76] S. Grossberg. "Adaptive Pattern Classification and Universal Recoding, II: Feedback, Expectation, Olfaction, and Illusions." *Biological Cybernetics*, pp. 187–202, 1976.

[Han92] P. J. B. Hancock. "Pruning neural networks by genetic algorithm." In I. Aleksander and Eds. J. Taylor, editors, *Proceedings of the 1992 International Conference on Neural Networks*, volume 2, pp. 991–994, Amsterdam, Netherlands, 1992.

[HL92] P. Hajela and C.-Y. Lin. "Genetic search strategies in multicriterion optimal design." *Structural Optimization*, **4**:99–107, June 1992.

[HME97] R. Hinterding, Z. Michalewicz, and A. E. Eiben. "Adaptation in evolutionary computation: a survey." In I. Aleksander and Eds. J. Taylor, editors, *Proceedings of the IEEE International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pp. 65–69, Amsterdam, Netherlands, Apr 1997.

[HNG] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. "A Niched Pareto Genetic Algorithm for Multiobjective Optimization." In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1.

[Hol75] J. Holland. *Adaptation In Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, 1975.

[ILE99] I. Inza, P. Larranaga, R. Etxeberria, and B. Sierra B. "Feature sub-set selection by Bayesian networks based optimization." *Artificial Intelligence*, **123**:157–184, 1999.

[IM96] H. Ishibuchi and T. Murata. "Multi-objective genetic local search algorithm." In *Proceedings of IEEE International Conference Evolutionary Computation, 1996*, pp. 119–124, May 1996.

[J 91] L. Davis J. Kelly. "Hybridizing the Genetic Algorithm and the K-Nearest Neighbors Classification Algorithm." In *ICGA*, pp. 377–383, 1991.

[Jin05] Y. Jin. "A comprehensive survey of fitness approximation in evolutionary computation." *Soft Computing*, **9**(1):3–12, 2005.

[Jon75] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, 1975.

[Kas93] T. Kasuba. "Simplified Fuzzy ARTMAP." *AI Expert*, **14**:18–25, November 1993.

[KBS97] R. Kohavi, B. Becker, and D. Sommerfield. "Improving simple Bayes." In *Proceedings of the European Conference on Machine Learning*, 1997.

[KGA01] A. Koufakou, M. Georgiopoulos, G. Anagnostopoulos, and T. Kasparis. "Cross-validation in Fuzzy ARTMAP for large databases." *Neural Networks*, **14**:1279–1291, 2001.

[LLL03] H. Liu, Y. Liu, J. Liu, B Zhang, and G Wu. "Impulse force based ART network with GA optimization." In *Proceedings of the 2003 International Conference on Neural Networks and Signal Processing*, volume 1, pp. 499–502, 2003.

[LLS00] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms." *Machine Learning*, **40**(3):203–228, 2000.

[LTD02] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. "Combining convergence and diversity in evolutionary multiobjective optimization." *Evolutionary Computation*, **10**(3):263–282, 2002.

[MG95] Brad L. Miller and David E. Goldberg. "Genetic Algorithms, Tournament Selection, and the Effects of Noise." *Complex Systems*, **9**(3):193–212, 1995.

[MG96] Brad L. Miller and David E. Goldberg. "Genetic Algorithms, Selection Schemes and the Varying Effects of Noise." *Evolutionary Computation*, **4**(2):113–131, 1996.

[MH93] Francisco Javier Marin Martin and Francisco Sandoval Hernndez. "Genetic Synthesis of Discrete-Time Recurrent Neural Network." In *IWANN '93: Proceedings of the International Workshop on Artificial Neural Networks*, pp. 179–184, London, UK, 1993. Springer-Verlag.

[MH95] S. Marriott and R. F. Harrison. "A modified Fuzzy ARTMAP Architecture for the Approximation of Noisy Mappings." *Neural Networks*, **8**:619–641, 1995.

[Mil97] Brad L. Miller. *Noise, sampling, and efficient genetic algorthms*. PhD thesis, Champaign, IL, USA, 1997.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[MS93] Heinz Muhlenbein and Dirk Schlierkamp-Voosen. "Predictive models for the breeder genetic algorithm I: continuous parameter optimization." *Evolutionary Computation*, **1**(1):25–49, 1993.

[MS95] William Mendenhall and Terry L. Sincich. *Statistics for Engineering and the Sciences*. Prentice Hall, 1995.

[MW02] M. Manic and B. Wilamowski. "Robust Algorithm for Neural Network Training." In *Proceedings of the 2002 International Joint Conference on Neural Networks*, pp. 1528–1533, 2002.

[NHB98] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. "UCI Repository of machine learning databases.", 1998.

[PGC93] W. F. Punch, E. D. Goodman, M. P. L. ChiaShun, P. Hovland, and R. Enbody. "Further Research on Feature Selection and Classification Using Genetic Algorithms." In *International Conference on Genetic Algorithms*, pp. 557–564, 1993.

[PGD] E. Parrado-Hernadez, E. Gomez-Sanchez, and YA Dimitriadis. "Study of distributed learning as a solution to category proliferation in Fuzzy ARTMAP based neural systems." *Neural Networks*, **16**:1039–1057.

[PHU05] P. P. Palmes, T. Hayasaka, and S. Usui. "Mutation-Based Genetic Neural Network." *IEEE Transactions on Neural Networks*, **16**(3):587–600, 2005.

[Sch85] J. David Schaffer. "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms." In *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 93–100, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[SD94] N. Srinivas and Kalyanmoy Deb. "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms." *Evolutionary Computation*, **2**(3):221–248, 1994.

[SDJ98] Randall S. Sexton, Robert E. Dorsey, and John D. Johnson. "Toward global optimization of neural networks: a comparison of the genetic algorithm and backpropagation." *Decision Support Systems*, **22**(2):171–185, 1998.

[SDS95] Robert E. Smith, B. A. Dike, and S. A. Stegmann. "Fitness inheritance in genetic algorithms." In *SAC '95: Proceedings of the 1995 ACM symposium on Applied computing*, pp. 345–350, New York, NY, USA, 1995. ACM.

[SLG06] Kumara Sastry, Claudio F. Lima, and David E. Goldberg. "Evaluation relaxation using substructural information and linear estimation." In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pp. 419–426, New York, NY, USA, 2006. ACM.

[SP94] N. Srinivas and L. M. Patnaik. "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms." *IEEE Transactions On Systems, Man And Cybernetics*, **24**(4):656–667, apr 1994.

[SS89] W. Siedlecki and J. Sklansky. "A note on genetic algorithms for large-scale feature selection." *Pattern Recognition Letters*, **10**(5):335–347, 1989.

[T 95] P. K. Simpson T. W. Brotherton. "Dynamic feature set training of neural nets for classification." *Evolutionary Programming IV*, pp. 83–94, 1995.

[TG94] Dirk Thierens and David E. Goldberg. "Convergence Models of Genetic Algorithm Selection Schemes." In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pp. 119–129, London, UK, 1994. Springer-Verlag.

[TRL06] S. C. Tan, M. V. C. Rao, and C. P. L. Lim. "On the reduction of complexity in the architecture of Fuzzy ARTMAP with dynamic decay adjustment." *Neurocomputing*, **69**:2456–2460, 2006.

[VHG01] S. J. Verzi, G. L. Heileman, M. Georgiopoulos, and M. Healy. "Rademacher penalization applied to Fuzzy ARTMAP and Boosted ARTMAP." In *Proceedings of the IEEE-INNS International Joint Conference on Neural Network*, volume 2, pp. 1191–1196, Washington DC, 2001.

[WC96] B. A. Whitehead and T. D. Choate. "Cooperative-Competitive Genetic Evolution of Radial Basis Function Centers and Widths for Time Series Prediction." *IEEE Transactions on Neural Networks*, **7**(4):869–880, July 1996.

[Wil96] James R. Williamson. "Gaussian ARTMAP: A neural network for fast incremental learning of noisy multidimensional maps." *Neural Networks*, **9**(5):881–897, 1996.

[Wil97] James R. Williamson. "A constructive, incremental-learning network for mixture modeling and classification." *Neural Comput.*, **9**(7):1517–1543, 1997.

[YA01] Olcay Taner Yildiz and Ethem Alpaydin. "Omnivariate Decision Trees." *IEEE Transactions On Neural Networks*, **12**(6):1539–1546, November 2001.

[Yao99] X. Yao. "Evolving Artificial Neural Networks." *PIEEE: Proceedings of the IEEE*, **87**:1423–1447, 1999.

[YH98] Jihoon Yang and Vasant G. Honavar. "Feature Subset Selection Using a Genetic Algorithm." *IEEE Intelligent Systems*, **13**(2):44–49, 1998.

[YL98] X. Yao and Y. Liu. "Making use of population information in evolutionary artificial neural networks." *IEEE Transactions on Systems, Man, Cybernetics part B*, **28**:417–425, 1998.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm." Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[ZT99] Eckart Zitzler and Lothar Thiele. "Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach." *IEEE Transactions on Evolutionary Computation*, **3**(4):257–271, 1999.