國立交通大學

資訊科學與工程研究所

博士論文

多層式多重族群基因規劃法與其應用

Layered Multi-Population Genetic Programming and Its Applications

研究生：林忠億

指導教授：楊維邦 博士

錢炳全 博士

中華民國九十六年六月

多層式多重族群基因規劃法與其應用

Layered Multi-Population Genetic Programming and Its Applications

研究生：林忠億
指導教授：楊維邦 博士
錢炳全 博士

Student: Jung Yi Lin
Supervisor: Dr. Wei-Pang Yang
Dr. Been-Chian Chien

國立交通大學
資訊科學與工程研究所
博士論文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philogophy
in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月

多層式多重族群基因規劃法與其應用

研究生：林忠億　　指導教授：楊維邦 博士
錢炳全 博士

國立交通大學資訊科學與工程研究所　博士班

摘　要

基因規劃法是屬於演化式計算的一種機器學習方法。其利用模擬生物界之演化機制，以「適者生存」的概念求取滿足給定條件之最佳解。如何改進基因規劃法的效率，一直是個熱門的研究方向。

分類問題在知識工程中是一個很重要的問題。大部份的分類問題是不能由人力知識進行解決的，因此，如何從資料中找出分類的依據，是許多機器學習方法被提出的動機。特徵選擇與特徵產生是兩個處理特徵的研究領域，經由對特徵進行適當的處理，在解決分類問題時，可以提升處理效率與分類準確率。

本篇論文將合併基因規劃法、特徵選擇與特徵產生等三種研究方向，並提出可解決分類問題的多層式多族群基因演算法架構。傳統的基因規劃法採用單族群機制，由此族群進行演化模擬而得出最佳解。我們將延伸單族群基因規劃法至多族群基因規劃法，並且提出一種多層式的架構將族群進行整合。每一層將使用多個族群進行演化模擬，並於下一層進行整合與再演化。此多層式架構不僅可利用多個族群來求得更好的解，更利用多層式架構將解進行改善與調整。經由實驗，我們將證明此方法具有高度準確性與高效率。另外，為了提高每一族群之學習表現，我們也提出一個依據平均適應度與剩餘演化世代數之動態突變調整方法。為了解決多類別分類問題，我們提出一個基於統計理論的解決機制，讓基因規劃法不僅適用於多類別分類問題，更能提高分類準確率。應用此架構，我們也提出一種融合特徵選擇與特徵產生的方法，並以實驗證明此方法之分類準確率與特徵處理效果。

關鍵字：基因規劃法；多族群基因規劃法；分類問題；特徵選擇；特徵產生；演化式計算。

Layered Multi-Population Genetic Programming and Its Applications

Student: Jung Yi Lin

Supervisor: Dr. Wei Pang Yang
Dr. Been Chian Chien

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

This study focuses on a proposed method based on genetic programming (GP). Genetic programming is a prominent technique of *evolutionary computation* (EC). It mimics the evolution mechanism of biological environment to determine optimal solutions for given training instances. Many researchers have been devoted to enhance effectiveness and efficiency of genetic programming.

The applications of the proposed method include classification and feature processing. Classification problems play an important role in the development of knowledge engineering. Hidden relations that can be used as a basis for classification are often unclear and not easily elucidated. Thus, many machine learning algorithms have arisen to solve such problems. Feature selection and feature generation are two important techniques dealing with features. Feature selection is capable of removing useless, irrelevant, redundant, and noisy features. Feature generation generates new useful features that could improve classification accuracy.

In this study we propose a layered multi-population genetic programming method to solve classification problems. The proposed method that can complete feature selection and feature construction simultaneously is also proposed. The layered multi-population genetic programming method employs layer architecture to arrange multiple populations. A layer is composed of a number of populations. Each population evolves to generate a discriminant function. A set of discriminant functions generated by one layer will be integrated and be transformed by the successive layer. To improve the learning performance, an adaptive mutation probability tuning method is proposed. Moreover, a statistical-based method is proposed to solve multi-category classification problems.

Several experiments on classical classification problems and real-world medical problems are conducted using different configurations. Experimental results show that the proposed methods are accurate and effective.

Keywords: *Genetic programming; multi-population genetic programming; classification; classifier design; feature selection; feature construction; evolutionary computation.*

誌　謝

五年，說起來很長，卻轉眼即逝。

能完成博士論文，首先要感謝兩位指導教授楊維邦教授及錢炳全教授，以及柯皓仁教授；他們在這幾年所提供的協助與指導，讓我得以一步步的完成論文的撰寫與修改。三位教授都不吝於給予我他們在研究上的寶貴心得，慢慢引領我進入學術研究的殿堂。他們是我的啟蒙老師，也是未來的目標。

感謝口試委員唐傳義教授、洪炯宗教授、項潔教授、蔣榮先教授、袁賢銘教授及孫春在教授，老師們對於本論文不吝指教及提供許多寶貴的建議，由衷感謝。

感謝培成學長，夙賢學長，政容學長，還有正吉學長，不管是在課業上或是研究上，都給我很大的幫助與鼓勵。尤其感謝在實驗室裡和我相處時間很長的培成學長，他讓實驗室充滿笑聲與歡樂，讓研究生活一點也不枯燥。

感謝同窗葉鎮源同學，他認真且嚴謹的做學問態度，在和他進行論文討論時，帶給我許多啟發與正面影響。感謝學弟陳信源與張庭毅在生活上與課業上的支持，雖然張庭毅比我還早畢業，在我畢業時已經變成張老師了。

感謝亞歷桑那大學的陳炘鈞教授及其指導之人工智慧實驗室的成員們，感謝他們給了我一整年豐富刺激的美國生活。在美國認識的朋友們，也感謝你們給我的支持與鼓勵。

實驗室來來去去的碩士班學弟妹們，還有其它實驗室相識的學弟妹們，雖然相處的時間不長，但是一起修課，一起討論專題或作業的生活卻是深刻而難忘。

最後，我要感謝我的家人們：我的父母、姐姐與弟弟，感謝家人們多年來的支持，使我能很平順的完成學業。尤其是現在在天上的父親，在待人處事與生活哲學中，如果我有絲毫值得被人稱許的地方，都是我父親教導出來的。特別感謝莉虹在心靈上或物質上的支持與照顧，她的無微不至，讓我得以在這幾年之中不受外務干擾而能專注於研究工作上。

博士的求學生活，不只是研究，而是一個階段的人生，感謝在這一路上遇見的人，這是我會一生懷念的時光。終於拿到博士學位，這是一個結束，也是一個開始，謝謝大家！

I owned so many thanks to my friends,
without whom my Ph.D would not be possible.
You let me know that I am not alone.

Thanks to my family, your support is vital for me.

To Sally, without whom everything is not worth doing.

Jung Yi Lin

NCTU, HsinChu, Taiwan
July, 2007

Contents

中文摘要 i

Abstract in English ii

誌謝 iv

1 Introduction 1

1.1 Genetic Programming 1

1.2 Classification 3

1.3 Feature Selection and Feature Construction 3

1.4 Motivation and Contribution 4

1.5 Overview of Chapters 4

2 Literature Review 6

2.1 Genetic Programming 6

2.1.1 Single Population Genetic Programming 6

2.1.1.1 The population and individuals 7

2.1.1.2 The fitness function 8

2.1.1.3 Generations and selection methods 8

2.1.2 Multi-Population Genetic Programming 11
2.2 Classification Algorithms 13
2.2.1 Classifiers 13
2.2.2 A Brief Review of Classifiers 15
2.3 Genetic Programming and Classification 21
2.4 Feature Selection and Feature Construction 22
2.5 Genetic Programming, Feature Selection and Feature Construction . . . 25
2.6 Research Questions 25

3 Research Design **29**

3.1 Introduction 29
3.2 LAGEP: Evolving a Population 29
3.2.1 Individual Definitions 30
3.2.2 Fitness Function 31
3.2.3 Validation 32
3.2.4 Elitism Evolution Strategy and the Evolutionary Flowchart . . . 32
3.2.5 AMPT: Adaptive Mutation Probability Tuning 33
3.3 LAGEP: Evolving Layers 35
3.3.1 Layered Architecture 35
3.3.2 Advantages of Using LAGEP 38
3.3.3 The Testing Phase and Z-value measure method, ZM 40
3.3.4 LAGEP: An Example 41
3.4 LAGEP-FS: LAGEP with Feature Construction and Feature Selection . . 44

3.4.1 Architecture 45

3.4.2 Proposed Feature Selection Methods 47

3.4.3 An Example 50

4 Experiment Study **53**

4.1 Introduction 53

4.2 Medical Classification Problems 53

4.3 Hypotheses 55

4.4 Experiments of *AMPT* method 56

4.5 LAGEP Experiments 57

4.5.1 Experimental Settings 57

4.5.2 Analysis and Discussion 60

4.5.2.1 Comparing classification accuracy between SGP, LAGEP and four cited methods 60

4.5.2.2 Comparing classification accuracy between LAGEP and **ES1** 61

4.5.2.3 Comparing elapsed training time 62

4.5.2.4 Comparing classification accuracy between LAGEP settings 65

4.5.2.5 The improvement of score values 67

4.6 LAGEP-FS Experiments 70

4.6.1 Experimental Settings 71

4.6.2 Analysis and Discussion 71

4.6.2.1 Analyzing **ES1** 72

4.6.2.2 Analyzing two-layer LAGEP-FS settings 76

4.6.2.3 Analyzing three-layer LAGEP-FS settings 76

4.6.2.4 Analyzing LAGEP-FS settings that have the same number of populations 76

4.6.2.5 Analyzing **ES9** and **ES10** 76

4.6.2.6 The effectiveness of new features 77

5 Conclusion and Future Work **80**

Bibliography **82**

Vita **91**

List of Figures

1.1 The analogy between Evolution and problem-solving 2

2.1 An individual representing $7x_1 + \sin(x_2)$ 8

2.2 An example of the crossover operator 9

2.3 An example of the mutation operator 10

2.4 The flowchart of GP evolutionary processes 11

2.5 An example of a five-population multi-population GP with a circle topology 12

2.6 An illustration of SVM 18

2.7 An illustration of a NN unit 20

2.8 An example of multi-layer NN 21

2.9 Taxonomy of feature selection algorithms 24

2.10 Two tree representations of function $F(x) = \prod_{i=1}^{32} a_i$ 26

2.11 Combining functions G and H to form function F 27

3.1 Flowchart of GP elitism evolution processes 33

3.2 Growth curve of p_m using *AMPT*. 35

3.3 Flowchart of LAGEP. 36

3.4 An example of feature transformation 39

List of Tables

2.1 A comparison between PADGP and IMGP 14

2.2 A comparison of feature selection evaluation functions 23

3.1 LAGEP example: Nine training instances selected from *cancer* problem of Proben1 42

3.2 LAGEP example: New training set T_2 generated by L_1 43

3.3 LAGEP example: Training results 44

3.4 LAGEP example: Training results for target class **M** and **B** 45

3.5 LAGEP-FS example: Nine training instances selected from *cancer* problem of Proben1 50

3.6 LAGEP-FS example: New training set generated by L_1 51

3.7 LAGEP-FS example: Training results 52

4.1 Summary of selected problems 54

4.2 Experiment setting **ES1** used for *AMPT* experiments 56

4.3 Learning score and paired t-test comparisons between **ES1** with *AMPT* and **ES1** without *AMPT* 58

4.4 Accuracy and paired t-test comparisons between **ES1** with *AMPT* and **ES1** without *AMPT* 59

4.5 Experimental settings of traditional single population GP and LAGEP . 60

4.6 Accuracy comparisons of six experimental settings and four methods . . 63

4.7 Paired t-test results between **ES1** and five other experimental settings . . 65

4.8 The average training time in seconds of six experimental settings and 18 problems. 66

4.9 Comparison of training performance between **ES1** and LAGEP settings . 68

4.10 The average score value of populations of **ES2** and the paired t-test results 70

4.11 Ten experimental settings of LAGEP-FS 72

4.12 Accuracy and average number of used features of ten experimental settings in CAN problems 73

4.13 Accuracy and average number of used features of ten experimental settings in DBT problems 74

4.14 Accuracy and average number of used features of ten experimental settings in HRT problems 75

4.15 Comparison of new feature effectiveness 78

Chapter 1

Introduction

1.1 Genetic Programming

Information technology is helpful when dealing with large quantities of data. People and enterprizes are generating an ever increasing amount of data stemming from applications as diverse as biological micro arrays, web documents, news articles, personal profiles, and diagnostic records. Both the extremely varied categories of data and the vast number of features characterizing each category contribute to the prohibitive difficulty of efficiently locating desired information. However, such information can be obtained using computational models developed in the field of information technology (IT).

Machine learning [56] is a developing research area focusing on providing a computational model to optimize given problems. Popular machine learning research includes neural network (NN), decision tree algorithms, and *evolutionary computation* (EC). Genetic programming (GP), which is an important technique of evolutionary computation, has been proposed for decades and is a popular topic of research. Evolutionary computation techniques simulate evolutionary mechanisms in biological environments. Following the principle "*survival of the fittest*", an individual having the best fitness value, specified by a properly designed fitness function, is determined as the solution. Researchers have been working diligently to improve the effectiveness

Evolution		Problem-solving
individual	$\longleftrightarrow$	Candidate solution
Fitness	$\longleftrightarrow$	Quality
Environment	$\longleftrightarrow$	Problem

Figure 1.1: The analogy between Evolution and problem-solving

and efficiency of GP, such as new fitness functions, new architecture, new individual expressions, and new evolutionary mechanisms.

EC research can be categorized into a branch of artificial intelligence, machine learning, or problem solving. In [47], an analogy between *problem-solving* and evolutionary computation is illustrated as Figure 1.1. According to the common concept of machine learning [56]:

> A computer program is said to *learn* from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

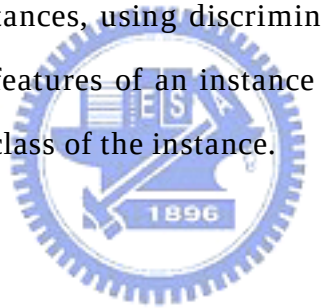
EC can be viewed as a field of machine learning because its individuals adapt to "fit" E, where *fit* means gaining high performance by the measure P.

Currently, multi-population GP is in early stages of development and is being investigated by many researchers. Traditional GP maintains a single population to obtain a solution. Multi-population GP, on the other hand, possesses greater diversity among individuals and is capable of reducing computational effort by using multiprocessor parallel computing techniques. Having higher individual diversity is important because it increases the possibility of finding better solutions. Multi-population GP usually obtains better results than traditional GP does. GP will be described in Chapter 2.

1.2 Classification

Classification is a widely used application in data mining tasks. Given a dataset and a classifier, classified results usually contain some interesting information, for example, "*what kind of customer would buy a that is very safe yet expansive?*" can be modelled as a classification problem. A classification algorithm can yield a list of customer candidates who meet specified conditions from past transaction records .

A classification algorithm or a classification model is called a *classifier*. It can be compared and evaluated based on its predictive accuracy, speed, robustness, scalability, and interpretability. Details of such criteria are described in Chapter 2. Some of the classifiers produce classification rules during the training process. Given unclassified objects could be classified into appropriate classes according to rules in the set. However, for numerical instances, using discriminant functions is a better choice. A discriminant function uses features of an instance to evaluate a real value, which in turn determines the proper class of the instance.



1.3 Feature Selection and Feature Construction

Feature selection is an important technique for dealing with raw features. It focuses on eliminating useless, irrelevant, redundant, and noisy features. Usually, feature selection is performed as a preprocessor of classifiers. Classifiers can achieve lower training error rates from data with selected features only. There are two primary advantages of using feature selection: to avoid noisy and to improve efficiency. First, some classifiers might be adversely affected by noise. Such noisy features make classifiers unable to generate good classification results. Second, due to the small quantify of features, classifiers are efficient and not time-consuming. This stems from the fact that the complexity, and thus time-consumption, of classification increases with the number of features. For instance, colon dataset [2] is a dataset of 62 tumor tissue samples. Each

sample has 2000 gene features. However, it is unnecessary that using all features to distinguish normal tissues from tumor tissues. Using feature selection, the classifier could obtain similar classification accuracy efficiently from tens of selected features.

Feature construction is used to construct new features from original features. This technique is useful when the representation of current features is different from the input requirement of the classifiers used or when current features are not capable of deriving good classification results.

1.4 Motivation and Contribution

In this work, we extend current multi-population GP concepts to construct a layered architecture multi-population GP model. The proposed layered multi-population GP will be used for solving classification problems and achieving feature selection tasks. Single population GP has been used to generate classifiers in many different ways. We believe that multi-population GP can be used to build effective and efficient classifiers.

The main contribution of the dissertation is to propose a novel *general-purpose genetic programming learning model*. This is accomplished by a layered multi-population GP technique. The layered architecture is inspired by neural network. We expect that our GP learning model will outperform traditional GP systems. In this work, we aim to solve classification problems through the proposed model. We also intend to use the model to achieve feature selection and feature generation. The learning model can be further investigated by researchers in evolutionary computation fields. The classification results will be worth referencing for other researchers in data mining fields.

1.5 Overview of Chapters

The remainder of this dissertation is organized as follows. In Chapter 2, we briefly

review related research involving GP, classification, feature selection and feature construction. Chapter 3 describes our research design. In this chapter, we start from the single population GP, and then proceed to explain the proposed layered multi-population GP method in detail. A brief example is provided in this chapter as well. Experiments and evaluations are given in Chapter 4. We conduct many experiments on classification, feature selection, and feature construction to illustrate the performance of proposed method. A discussion of the experimental results are also given at Chapter 4. Finally, Chapter 5 concludes this work and points out directions for further research.

Chapter 2

Literature Review

2.1 Genetic Programming

In this section, we first introduce single population genetic programming. Multiple-population genetic programming will be introduced in Section 2.1.2

2.1.1 Single Population Genetic Programming

Genetic programming (GP) was proposed by Koza in 1992 [40]. This evolutionary computation has rapidly developed and led to such diverse applications as sumbolic regression, robot-controlled programs, data mining, circuit design, and classification [40] [41] [42] [43] [44] [4] [20]. GP can discover underlying relationships among given data. Such relationships can be represented in variety of forms such as functional expressions or graphs.

GP, as well as *genetic algorithm* (GA), mimics evolutionary phenomena and follows the principle "*survival of the fittest*". In biological environments, creatures that survive and reproduce must exhibit advantageous characteristics. Given recurrent elimination, the survivors are fittest individuals in the population. Inspired by such phenomena, evolutionary computation techniques utilize similar elements: *population*, *individuals*,

the *fitness function*, *generations*, and the *selection method*.

2.1.1.1 The population and individuals

An *individual* stands for a potential solution for the given optimization problem. A *population* is a set of individuals. Traditionally, an individual is represented by a *tree-structured* expression; also the expression used by Koza [40]. Different expression structures have been proposed, such as[28] [78]. However, for numerical feature data, using the tree-structured expression to represent a discriminant function is a good choice because it is compact and efficient. An individual is comprised of three sets: a *variable set*, a *constant set*, and a *operation set*.

- Variable set

  The variable set contains variable symbols that stand for features, for example, in the case that a given instance has four features, we can use x_1, x_2, x_3, and x_4 to denote those four features.

- Constant set

  A constant set is a collection of constants. Constants could be mathematical constants, such as π, e, or real-valued numbers of a predefined interval.

- Operation set

  The operation set is a set of arithmetic operations, trigonometric functions, logical operations, and other domain-specific user-defined functions. An individual is represented by a *binary tree* if all of the operations are binary operations.

Example

Assume a variable set $\{x_1, x_2, x_3, x_4\}$, a constant set $\{1, 2, \ldots, 10\}$, and operation set $\{+, -, \times, \div, \sin, \cos\}$. An individual constructed from these components represented as $7x_1 + \sin(x_2)$ is shown in Figure 2.1.

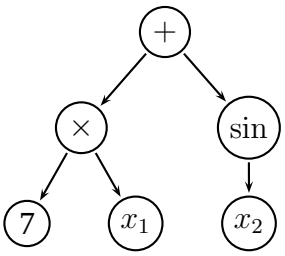


Figure 2.1: An individual representing $7x_1 + \sin(x_2)$

The number of individuals in a population is called *population size*. Larger population size make the population cover a highly diverse individuals that in turn allows the optimal solution to be found. However, larger population size also results in a larger searching space which could increase computation efforts or even a waste [23] [22].

2.1.1.2 The fitness function

The fitness function is a domain-specific problem-driven function that evaluates performance of individuals. The *fitness value* of an individual is derived from the fitness function. For instance, the fitness value of an individual in a regression problem can be calculated by the mean value of square errors (*MSE*). The function used for calculating the *MSE* is the fitness function in this case. An individual's fitness value in this problem is its *MSE* value.

2.1.1.3 Generations and selection methods

A *generation*, or *iteration*, indicates a process of replacing current population. This process is accomplished by means of several *genetic operators* such as *reproduction*, *crossover*, and *mutation*. New individuals are produced after one or two individuals applied one of these operators. Each of them operates in accordance with a predefined probability. When encountering a complex problem, more generations are required to find the optimal solution.

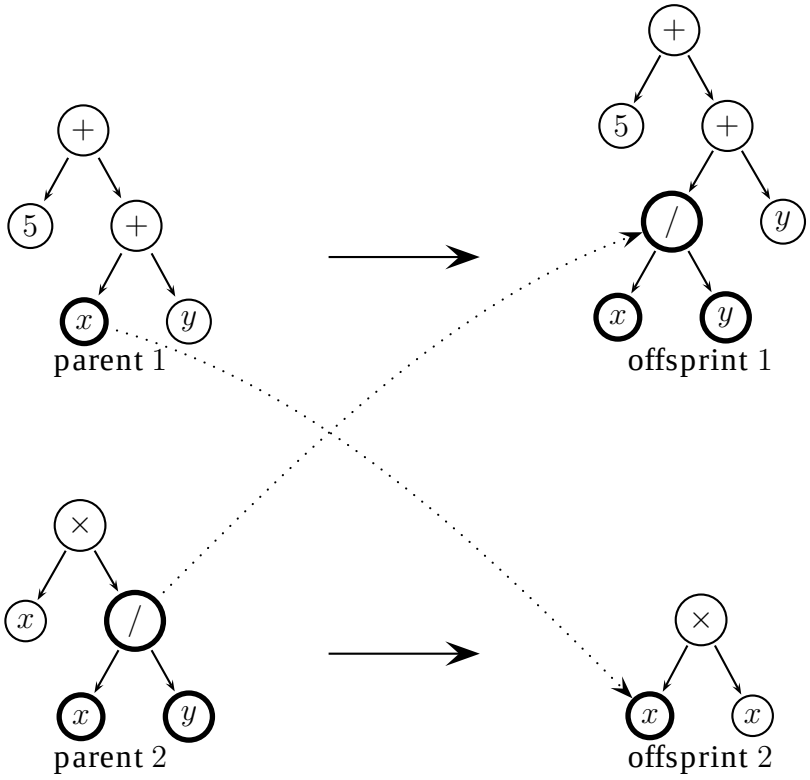


Figure 2.2: An example of the crossover operator. Parents $(5 + (x + y))$ and $(x \times (x/y))$ generate two offspring $(5 + ((x/y) + y))$ and $(x \times x)$

- Reproduction

  The idea behind the reproduction operator is that a good individual in the population should not be eliminated. This operator is responsible for propagating a selected individual within a new population.

- Crossover

  The crossover operator mimics the natural mating process but it always generates two offspring. This operator selects two individuals from the population to be *parents* through the *selection method*. Parents are not necessary different. The *crossover points* of each parent individual are decided randomly. Parents swap their subtree to generate two new individuals. An example is shown at Figure 2.2, the two individuals selected from the population, $(5 + (x + y))$ and $(x \times (x/y))$, are parents. After the crossover operator is executed, two offspring $(5 + ((x/y) + y))$ and $(x \times x)$ are produced.

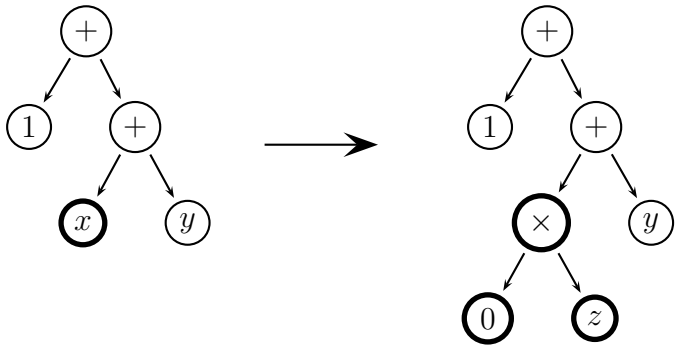


Figure 2.3: An example of the mutation operator. Assume that give variable set is $\{x, y, z\}$, the operation set is $\{+, -, \times, /\}$, and the constant set is $\{0, 1\}$.

- Mutation

  The reproduction operator and crossover operator use information from individuals in the current population. Through these two operators, terminals and functions not extant in the current population will not emerge in the new population . Mutation is the only way to create an individual with different elements. It also can be used to escape local optimums because mutants may result in different structures to explorer different area of the searching space. The mutation operator first selects a node of the individual, called *mutation point*. It then replaces the subtree rooted at the mutation point by a new subtree. An example shown in Figure 2.3 assumes that a give variable set is $\{x, y, z\}$, the operation set is $\{+, -, \times, /\}$, and the constant set is $\{0, 1\}$. The individual $(1 + (x + y))$ generates a mutant $(1 + (0 \times z) + y)$ by the mutation operator.

Choosing the appropriate individual is problematic since there is no courtship simulation. Many selection methods have been proposed to improve selection performance such as fitness proportional selection, truncation selection, ranking selection, and tournament selection [40][4][55]. Here, we utilize the commonly used method: tournament selection[55]. Tournament selection first selects k individuals from the population randomly. Then an individual I_i is selected via a probability p_i associated with its rank among the k individuals compared to their fitness values. In the case

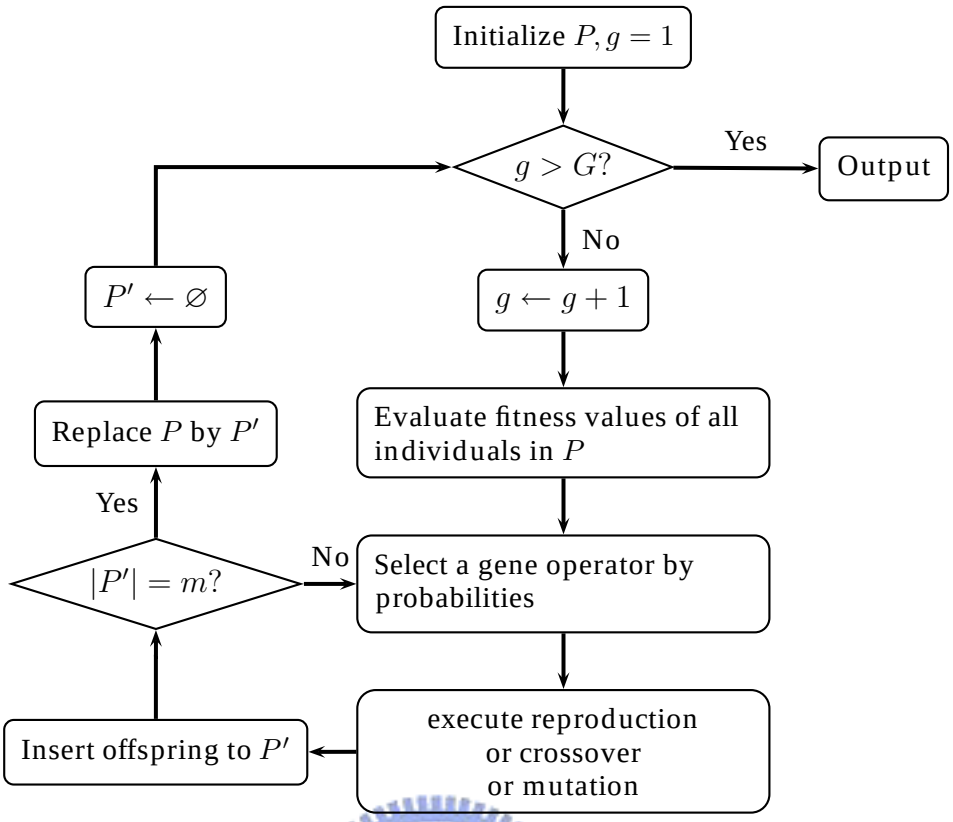


Figure 2.4: The flowchart of GP evolutionary processes. G is the maximum generation, P is the current population, P' is the population of next generation, and m is the population size.

$k = 1$, tournament selection acts equally with the random selection which selects an individual from the population randomly. In the case that the tournament selection always selects the best individual against the k individuals, it called the *deterministic* tournament selection.

After the passage of a sufficient number of generations, the population would have individuals with high fitness values. Consequently, these individuals are taken as as results. However, if the fitness values are not satisfied yet, the process of evolution could be continued until such specified conditions are reached. The basic evolution flowchart is shown in Figure 2.4.

2.1.2 Multi-Population Genetic Programming

Multi-population GP [3] [62] [7] [6] [18] [19], which employs several populations to

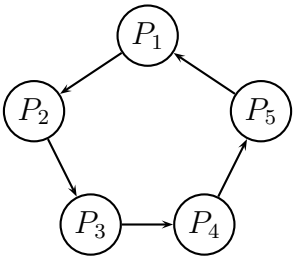


Figure 2.5: An example of a five-population multi-population GP with a circle topology. An arrow indicates the direction of migration between two populations.

solve optimization problems, has been proposed. However, multi-population GP is still an area of research under investigating. There are two primary multi-population GP models [19]: parallel and distributed GP (*PADGP*), and isolated multipopulation GP (*IMGP*).

- PADGP

  PADGP mimics multiple population evolutionary environments. Intuitively, an environment containing many populations would produce better individuals than a single population. Given a proper migratory mechanism, good individuals of a population could enhance the performance of others. Many different topologies of PADGP have been proposed, such as the circle topology and the random topology. Figure 2.5 shows the circle topology in which circles stand for populations [6]. The arrows in Figure 2.5 indicate the direction of migration between two populations. Basically, only the individuals that exceed a predefined fitness threshold can migrate to other populations. The incoming good individuals not only increase the diversity but also provide positively effect to the population. However, a system implementing the migration mechanism is complex.. A parallel computation environment is more able to accomplish this task. The parameters of migration are experimentally investigated by [19].

- IMGP

  A multi-population GP without migratory mechanism is called an *isolated* multi-population GP. It simulates creature populations living on isolated islands be-

tween which no communication exists. Simulating an isolated multi-population GP is easier because it is actually a number of independent single population GP systems, and evolutionary process of all populations can be run in serially.

A sub-population, also called a *dime*, of PADGP searches different regions of the searching space initially. Through the migratory mechanism, dimes falling in the local optimum can escape via crossover with migrants. Note that it is very possible that migrants are selected since they have high fitness values. However, it is also possible that performance of all dimes grows slowly because the migrants already stay at a local optimum region. On the other hand, each population of IMGP evolves independently. They explore different regions of the searching space and escape local optimums by mutation only. If one of them approaches the global optimum, such population will not be obstructed by migrants.

Fernández et al. [19] performed several experiments with PADGP, IMGP, and traditional single population GP (*SGP*). Their experiments show that PADGP performs better than IMGP, and that both PADGP and IMGP obtain better performance than TSGP does in most situations. A comparison between PADGP and IMGP is shown in Table 2.1.

2.2 Classification Algorithms

2.2.1 Classifiers

The term *classifier* usually indicates a classification algorithm or a classification model. Classifiers can be compared and evaluated through following criteria [24]:

1. Predictive accuracy: A good classifier should have the ability of predicting the class of new or previously unseen objects accurately.

	Pros	Cons
IMGP	1. It is not necessary to change the standard single GP population evolution algorithm. 2. Different populations can use different configurations such as individual length or operators.	1. Small isolated populations have limited diversity of individuals. 2. Good populations do not benefit others. Some populations may have good individuals but others may be stuck on local optimums.
PADGP	1. More effective of finding good solutions. 2. Individuals migrated from other populations can help the destination population escape local optimum provided migrants have higher fitness.	1. Many parameters need to be considered such as (a) Communication topology (b) Frequency of migration (c) The number of migrants and each of them affects performance.

Table 2.1: A comparison between PADGP and IMGP

2. Speed: The computation costs of a good classifier in constructing and executing should be low.

3. Robustness: Given noisy data or data with missing values, a good classifier should still make correct predictions.

4. Scalability: A good classifier should be constructed efficient even for large mount of data.

5. Interpretability: A good classifier should provide high level of understanding and insight.

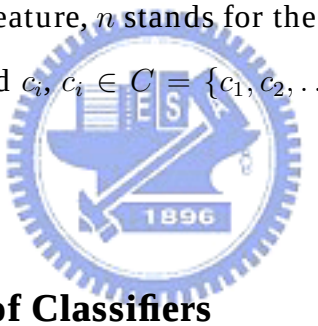
In general, the predictive accuracy is the most important factor. Most classifiers are developed or improved with the goal of achieving high accuracy. There are many accurate evaluation methods. Most of classification models are supervised in that they require a set of training instances. They use a set of training data T to change parameters or to adapt themselves. This process is called *training phase*. The *testing phase* is when the trained classifier acts on unclassified data to predict their proper class.

Usually, a *training instance* is an object whose class label is already known. Here we define a training instance and the training set T by

$$T = \{x_i | 1 \leq x_i \leq N\}, \tag{2.1}$$

$$x_i = \{a_{i1}, a_{i2}, \ldots, a_{in}, c_i\} \tag{2.2}$$

where a_i stands for the *i*-th feature, n stands for the number of features, N is the number of training instances, and c_i, $c_i \in C = \{c_1, c_2, \ldots, c_K\}$ stands for the class label of the instance x_i.



2.2.2 A Brief Review of Classifiers

In this section, we will briefly reviews some well-known and commonly-used classifiers such as the instance-based classifier, the naïve Bayesian, the support vector machine, and the neural network.

Instance-based Classifiers

The most famous instance-based classifier is the k-nearest neighbor classifier (*kNN*). It is an intuitive and widely used classification approach of learning by analogy [15] [10] [46] [57] [24]. In the training process, the nearest neighbor classifier stores the given training instances in an n-dimensional space; where n is the number of features, or say *dimensions*, of training instances. In the classification process, *kNN* calculates the distances between the new object and other training instances. The class label of the new

object is determined by nearest k training instances, for example, if $k = 5$ and the most frequent class label of such 5 instances is class C_1, the new object is classified into class C_1. *kNN* is simple and accurate. Similar instances usually are separated into close regions of the data space. However, *kNN* is time-consuming while evaluating distances in high-dimensional vector space. If there are noisy features, the distance between objects might be a useless measure.

The Naïve Bayesian Classifier

The Bayesian classifier is based on Bayes' theorem [27] [14] [24] [57] [64]:

Theorem 1. $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$

The Bayesian network is a network-like scheme derived from Bayesian classifier concepts. The Bayesian theorem provides a probability model of conditional probability for constructing a classifier. Considering a given data object $y = (g_1, g_2, \ldots, g_n)$, the probability of $y \in c_k$ can be represented by $P(c_k|y)$ which we can re-write it as:

$$P(c_k|y) = \frac{P(c_k)P(y|c_k)}{P(y)} = \frac{P(c_k)P(g_1, g_2, \ldots, g_n|c_k)}{P(g_1, g_2, \ldots, g_n)} \tag{2.3}$$

where $P(c_k)$, $P(g_1, g_2, \ldots, g_n|c_k)$, and $P(g_1, g_2, \ldots, g_n)$ can be derived from T. However, estimating $P(c_k)P(g_1, g_2, \ldots, g_n|c_k)$ from T can be difficult or computationally expansive. The Naïve Bayesian classifier assumes that the attribute values are *conditionally independent*. Such assumption makes the estimation easy to construction without decreasing accuracy. The class of y can be therefore determined by equation 2.4, i.e., y is classified into the class which has the highest probability.

$$\underset{c_k}{argmax} = P(c_k)P(g_1, g_2, \ldots, g_n|c_k) = P(c_k)\prod_i P(g_i|c_k), \tag{2.4}$$

where $1 \leq k \leq K, 1 \leq i \leq n$.

Support Vector Machine (SVM)

Support vector machine (*SVM*) [26] [76] [70] [67] is a popular classifier currently. For training data (x_i, y_i), where x_i is a training instance represented in $\mathbb{R}^n$ space, and $y_i \in \{+1, -1\}$ is the class label, SVM tries to find an optimal *hyperplane* to separate training instances if training instances are linearly separable. In the case that the training instances are not linearly separable, SVM maps them into a higher-dimensional *feature space* H via a mapping $\Phi : \mathbb{R}^n \rightarrow H$; where H is a Hilbert space. Then SVM tries to find a separating hyperplane with maximum *margin* on H. The margin is evaluated by *support vectors* that are specific training instances. Since the hyperspace model is used, here we denote a training instance x_i as a vector $\overrightarrow{x_i}$. The objective is to find a hyperplane in the form:

$$\overrightarrow{w} \cdot \overrightarrow{x_i} + b = 0 \tag{2.5}$$

where $\overrightarrow{w}$ is the margin, b is a constant.

There are many possibilities of $\overrightarrow{w}$ but only the one that forms *maximum margin* is desirable. This becomes an optimization problem:

$$\begin{aligned} &\min \ \frac{1}{2}\|\overrightarrow{w}\|^2 \\ &\text{subject to} \ \ y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} + b) \geq 1, 1 \leq i \leq n \end{aligned} \tag{2.6}$$

Cortes and Vapnik [9] proposed the *soft margin* method which tolerates errors. They modified the optimization problem by introducing a slack-variable ξ which measures the misclassification degree w.r.t a training instance $\overrightarrow{x}$:

$$y_i(\overrightarrow{w} \cdot \overrightarrow{x} + b) \geq 1 - \xi_i, 1 \leq i \leq n$$

The optimization problem becomes

$$\begin{aligned} &\min \ \frac{1}{2}\|\overrightarrow{w}\|^2 + \zeta \sum_i \xi_i \\ &\text{subject to} \ \ y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} + b) \geq 1 - \xi_i, \xi \geq 0 \end{aligned} \tag{2.7}$$

where ζ is the penalty parameter.

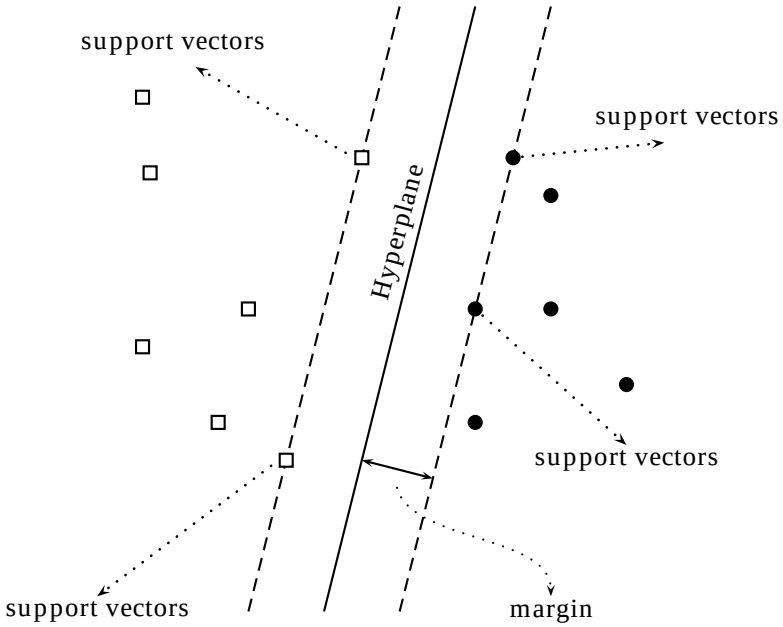


Figure 2.6: An illustration of SVM. Square points and circle points stand for instances of different classes.

Once the $\overrightarrow{w}$ is obtained, we have a formula for a given unknown data object $\overrightarrow{x_j}$ by

$$\text{class of } \overrightarrow{x_j} = \begin{cases} +1 & \text{if } \overrightarrow{w} \cdot \overrightarrow{x_j} + b \geq 1 \\ -1 & \text{if } \overrightarrow{w} \cdot \overrightarrow{x_j} + b \leq -1 \end{cases} \tag{2.8}$$

The training phase of SVM attempts to find the hyperplane. After the hyperplane is found, the built model can be used to classify unknown objects. We illustrate SVM in Figure 2.6.

Different SVM architectures can be obtained through different *kernels*. This idea is derived from the fact that SVM only deals with dot-products between data. Using the *kernel trick*, the dot-products can be replaced by a kernel. Given a kernel function K and two vectors $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$ representing objects x_i and x_j in H, K is the dot-product function:

$$K(x_i, x_j) = \langle \Phi(\overrightarrow{x_i}), \Phi(\overrightarrow{x_j}) \rangle \tag{2.9}$$

Moreover, provided the data are normalized to unit norm in H, i.e., $\forall x, \|\Phi(\overrightarrow{x})\| = 1$,

we can then define the distance between $\overrightarrow{x_i}$ and $\overrightarrow{x_j}$ by $d\left(\Phi(\overrightarrow{x_i}), \Phi(\overrightarrow{x_j})\right)$:

$$\begin{aligned} d\left(\Phi(\overrightarrow{x_i}), \Phi(\overrightarrow{x_j})\right) =& \langle \Phi(\overrightarrow{x_i}) - \Phi(\overrightarrow{x_j}), \Phi(\overrightarrow{x_i}) - \Phi(\overrightarrow{x_j}) \rangle \\ =& 2\left(1 - K\left(\overrightarrow{x_i}, \overrightarrow{x_j}\right)\right) \end{aligned} \tag{2.10}$$

In summary, there are two main properties of using kernels: First, the exact form of the function Φ is not necessary but rather the kernel function K. Once a *valid* kernel is chosen, SVM can apply such a kernel to data and complete the classification. A kernel is valid if and only if its corresponding kernel matrix is symmetric and positive semi-definite [54]. Second, a kernel can be thought of as a *similarity measurement*. The larger the $K(\overrightarrow{x_i}, \overrightarrow{x_j})$, the more similar x_i and x_j.

SVM is a rapidly developing area because it provides a general framework for kernels. For a given dataset, researchers only need to decide what kind of kernels should be used. Moreover, the classification accuracy obtained by SVM is usually high.

Neural Network

Neural network (*NN*) [30][34][8] or neural computing, is inspired by the related biological concepts. NN adjusts its own parameters to suit a given problem.

The NN is constructed with *nodes*, which are called *units*, connected by *links*. Each link has a weight associated with it. Units and links form the topology of the NN model. A unit consists of three components: input function, activation function, and output. The input function, denoted as $\sum$, is a linear function that computes the weighted sum of input links. It then sends the value to the activation function, which is denoted as g. The activation function transforms the weighted sum to an activation value as the output of this unit, denoted as a_i. The activation function is the most important element of the process. Such function can be a step function, a sign function, or a sigmoid function[11]. A typical sigmoid function is:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.11}$$

At the last step, the unit sends a_i to all output links. We illustrate a unit in Figure 2.7, where $w_{j,i}$ is the weight of the link from unit j to unit i, and $w_{i,k}$ is the weight of the

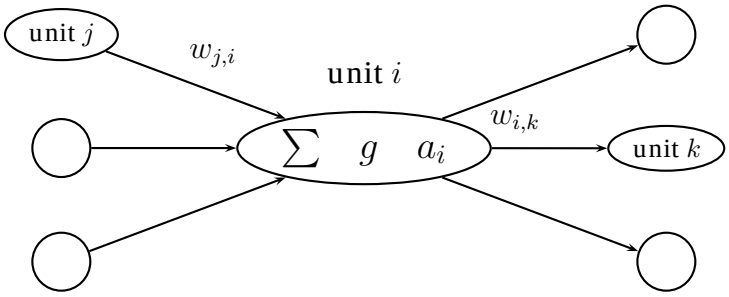


Figure 2.7: An illustration of a NN unit.

link from unit i to unit k.

A multi-layer NN is organized in layers: an input layer, hidden layers, and an output layer. A multi-layered NN can represent any continuous function with single hidden layer, and represent any function with two hidden layers. We show a multi-layer NN in Figure 2.8; such multi-layer network has one hidden layer, four input units, three hidden units, and six output units.

A learning method called *back-propagation* is widely used to update weights of links in multi-layer NN. The back-propagation first evaluates the error for the output units. The evaluation of this error is accomplished by employing a gradient descent. It starts in the output layer, propagates the error value back to previous layers, and then updates the weights.

Classification problems are appropriate to NN. Many classifiers based on NN are proposed [8] [49] [60] [73] [79]. Typically, the input layer has n units, that is the number of features of an object. The classification result is obtained from the output vector generated by the output layer. The number of units in the output layer depends on the classification scheme used . In a one-to-many classification scheme, the output layer should have K units that represent how many classes exists. In the case that a one-to-one scheme is chosen, the output layer would have two units: *accept* and *reject*.

Parameters of an NN classifier should be carefully decided; for example, the *learning rate* and the number of hidden units. The learning rate is used to control the adaption degree. Small learning rates are capable of finding optimal weight but delay the speed of training. Many disadvantages are caused by using too many hidden units. One problem is that the efficiency is decreased. The back-propagation algorithm is

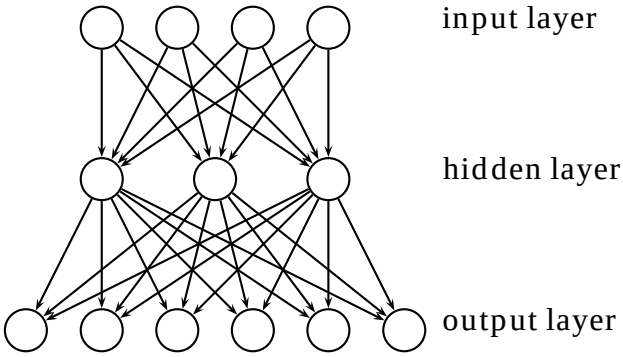


Figure 2.8: An example of multi-layer NN. This example shows one input layer, one hidden layer, and one output layer. The input layer, the hidden layer, and the output layer contain four units, three units, and six units, respectively.

slow to converge with since more computations are required. Excessive hidden units may cause the *overfitting* problem become more serious. However, it is not difficult to determine the proper number of hidden layers. Hayashi, et al. [25] claimed that *"Never try a multilayer model for fitting data until you have first tried a single-layer model."* Since a NN with two hidden layers can represent any function, using more than two hidden layers is unnecessary. The constructed NN classifier and its result are both difficult to be directly interpreted by humans. This makes the NN classifier a black box that has difficulty presenting to represent knowledge to users.

2.3 Genetic Programming and Classification

Many accurate and efficient classifiers have been developed based on GP in recent years [4] [6] [17] [21] [35] [38] [39] [51] [58] [59] [71] [75] [5]. Since GP itself cannot be directly used as a classifier, researchers have used it to find optimal solutions with certain classifier structures.

To generate classification rules, Freitas [21] proposed the Tuple-Set-Descriptor (TSD), a logical formula to represent an individual. Bojarczuk et al. [5] used GP to generate classification rules for medical problems. Falco et al. [17] published a more complete work for generating classification rules by GP. They used a real-valued range opera-

tion to deal with real-valued features. Kotani and Sherrah [39][71] used GP to perform feature selection before using other classification methods. Multi-category classification problems are more difficult than two-class classification problems. Kishore et al. [35] have considered such problems as multiple two-class classification problems and then generated corresponding expressions. They called it *GPCE*. A GPCE is actually a discriminant function, which is efficient in dealing with real-valued features. Their method requires K runs for a K-class classification problem because only one GPCE is generated with respect to a target class in each run. Muni et al. [58][59] proposed a novel method to solve K-class classification problems in a single run. Each individual in their work is represented by a multitree structure. Adapting one multitree individual is equivalent to K traditional individuals simultaneously. Once the evolutionary process ends, the K trees of an individual represent K discriminant functions. Loveard and Ciesielski [51] proposed five classification frameworks for solving multi-category classification problems including binary decomposition, static range selection, dynamic range selection, class enumeration, and evidence enumeration. Tsakonas [75] compares four different structures, including decision trees, fuzzy rules, neural networks, and fuzzy Petri-nets; all evolved by GP with several different classification problems selected from [65]. Brameier and Banzhaf [6] used linear GP and multi-population GP techniques. Individuals are represented as strings of C-like codes. The output of the codes stands for the classification result. They compared their method to NN in many classification problems also cited from [65]. For most problems, GP achieves significantly better classification accuracy than NN does.

2.4 Feature Selection and Feature Construction

Feature selection is an important technique of pattern recognition dealing with raw features. It focuses on removing useless, irrelevant, redundant, and noisy features. The classification accuracy of data with selected features is better than that with all features.

Table 2.2: A comparison of feature selection evaluation functions (cited from [12]).

Evaluation function	Generality	Time complexity	Accuracy
Distance measure	yes	low	-
Information measure	yes	low	-
Dependance measure	yes	low	-
Consistency measure	yes	moderate	-
Classifier error rate	no	high	very high

Research dealing with feature selection has been proposed [1] [12] [31] [32] [36] [37] [45] [66]. John [32] divides feature selection methods into *filter* groups and *wrapper* groups. Filter methods measure the degree of feature relevance and decide whether to leave or remove it, for example, two similar features can be identified by their correlation value. By contrast, wrapper methods[31] [37] cooperate with particular classifiers to find features that affect the performance of the classifiers. Dash [12] did a solid survey of many feature selection methods. They categorized feature selection methods into five categories based on the evaluation of their features discrimination ability: *distance measure*, *information measure*, *dependence measure*, *consistency measure*, and *classifier error rate*. TABLE 2.2 shows the five categories cited from [12].

Both Dash and Jain proposed taxonomies of feature selection methods as shown in Figure 2.9. In [31], experiments with 15 different feature selection algorithms on 18 extracted features of SAR images are made. Kudo and Sklansky also compared the differences between many feature selection methods in [45]. There compared 16 different feature selection methods with a 1-NN classifier, including sequential algorithms and branch-and-bound algorithms. Moreover, they also investigated various techniques for choosing feature selection methods for different problem types. Kittler and Pudil [36][66] proposed many feature selection techniques, such as (generalized) sequential forward selection, (generalized) sequential backward selection, (generalized) plus l-

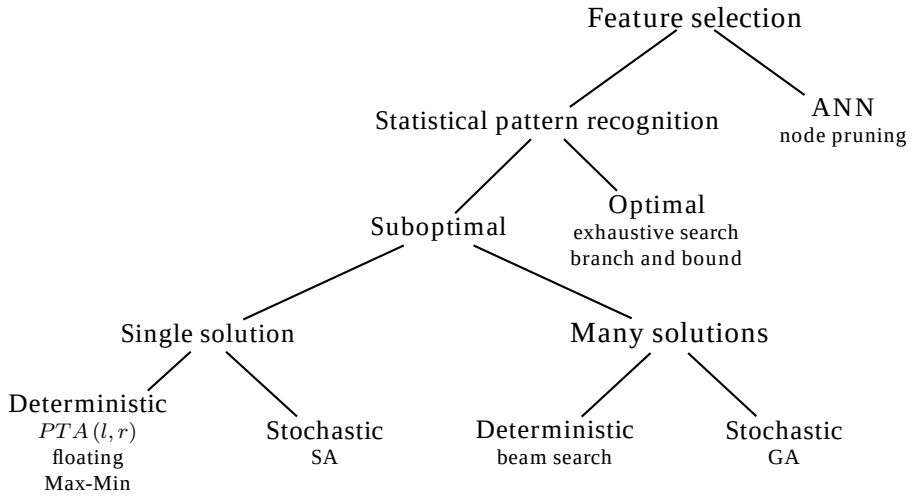


Figure 2.9: Taxonomy of feature selection algorithms (cited from [31]).

take away r selection, sequential forward floating selection, and sequential backward floating selection. Pernkopf compared the accuracy of Bayesian network classifiers and kNN classifiers with different feature selection methods in [64].

A different way to deal with features is *feature construction*, or *feature generation* [33] [48] [52] [53] [63] [77] . This method generates representative features or classification-oriented ones [52]. The former is mainly applied when the given data are not represented in a distinguishable form for the classifier on hand, e.g., handwriting and images. The most common feature extraction methods are *principle component analysis* (*PCA*) [33] and *linear discriminant analysis* (*LDA*). Ma et al., [52] proposed a general feature extraction framework and two nonlinear feature extraction algorithms, kernel function and mean-STD-norm, cooperating with a SVM classifier. Mao and Jain proposed several artificial neural network models such as PCA-networks, LDA-networks, and *nonlinear discriminant analysis* (*NDA*) networks in [53]. Wang [77] investigated the performance of PCA, LDA, minimum classification error, and generalized minimum classification error with SVM classifiers on vowel databases.

2.5 Genetic Programming, Feature Selection and Feature Construction

Research on feature selection and feature construction using evolutionary computation techniques have grown rapidly. In [61] [68] [72], genetic algorithm is applied to achieve feature selection. Articles focusing on using GP can be found in [29] [39] [59] [16] [69] [71] [74] . Sherrah [71] used GP to perform feature selection before applying particular classifiers. The unused features in the result are removed. Kotani [39] and Hong [29] used GP to generate new features and employed other classifiers with the generated features to perform classification tasks. SVM and ANN were used as classifiers with generated features on bearing faults problems. Rizki [69] proposed a hybrid evolutionary learning algorithm, *HELPR*, to perform feature extraction from raw input data. Smith [74] combined GP and GA to complete feature generation and feature selection with the C4.5 classifier. Otero et al. [16] used GP to achieve feature construction from inadequate features. Muni [59] proposed an approach to construct a multi-class classifier with an online feature selection named GP_{mtfs}, which is a multitree GP [58] based feature selection. They proposed two crossover algorithms with the multitree GP technique to find minimum number of useful features.

2.6 Research Questions

Using functional expressions to represent individuals is effective in GP. The tree-expression representation is a common data structure for functional expressions[40]. Two problems occur when individuals are represented by tree-expression. The first problem is that it is difficult to choose appropriate operations for a given problem because salient characteristics of the problem are completely unknown. If the operator set contains many operations, there is a greater possibility of discovering optimal solutions; but the searching space becomes larger and therefore may become an imprac-

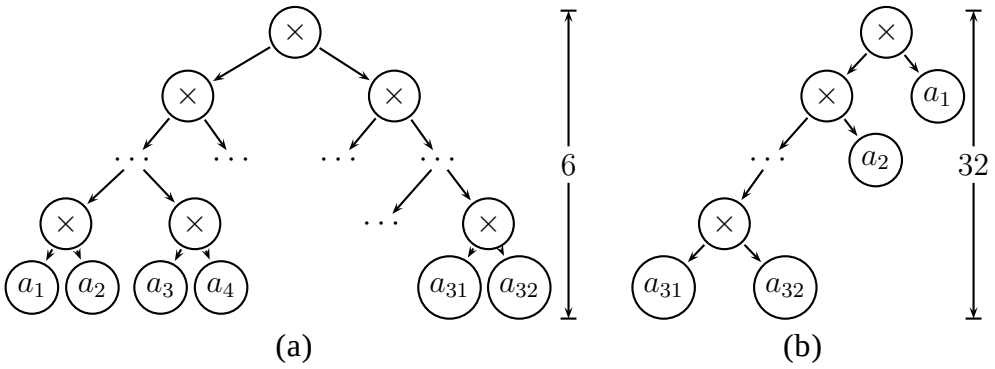


Figure 2.10: Two tree representations of function $F = \prod_{i=1}^{32} x_i$. (a) A balance tree uses 6 of heights. (b) A skew tree uses 32 of heights.

ticable situation. Fortunately, as shown in [35], GP with an operation set comprising only basic arithmetic operations, i.e. $\{+, -, \times, /\}$, is capable of generating classification results comparable to that of an operation set comprising additional operations. Second, since we do not have prior knowledge about the optimal solution, as a matter of course its length is unknown. Unfortunately, the tree depth is an important parameter of tree-structured individuals. The predefined individual length, just as the length of a string-expression individual or the number of available nodes of a tree-expression individual, is usually chosen according to heuristic or empirical assumptions. The following is an example of a classification problem containing 32 dimensional data, i.e., a training instance x is represented by $x = (a_1, a_2, \ldots, a_{32})$. Suppose that an optimal solution F is known as $F(x) = \prod_{i=1}^{32} a_i$. F can be represented by a balance tree or a skew. The balanced tree has a height of six, and the skew tree has a height of 32, as shown in Figure 2.10. An individual can contain at most $2^{32} - 1 = 4,294,967,295$ nodes if the predefined maximum depth is 32. A population containing so many large trees is highly complex and would be thereby impracticable. On the other hand, if the predefined maximum depth is fixed at six, it is very difficult to generate the ideal balanced tree. Moreover, the function F will never be obtained if the maximum depth is less than six.

Using an acceptable and practicable individual size is a simple but dangerous way to avoid this problem. This problem has motivated us to develop this work. Since

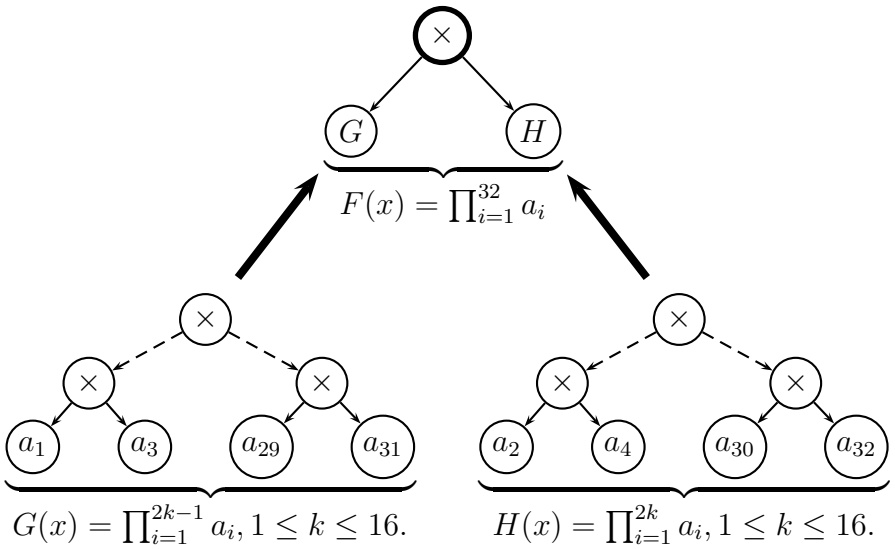


Figure 2.11: Combining functions G and H to form function F. G and H are small functions combined by a tree rooted by a $\otimes$ node.

a long function can be viewed as a composite of a number of small functions, it is possible to combine a number of small GP solutions into a large one. Therefore, it is desirable to generate those small solutions with a practicable size of individuals and then use them to compose a larger solution. For example, consider the function F (as above) and two functions $G(x) = \prod_{i=1}^{2k-1} a_i$ and $H(x) = \prod_{i=1}^{2k} a_i$, where $1 \leq k \leq 16$. Clearly, F can be represented by $F = (G \times H)$, as shown in Figure 3 where the tree representations of G and H have at most a height of 16 rather than 32. Functions G and H can be generated by two separate GPs and then can be combined together to form F. Here we attempt to develop a method by which we can determine a proper node to combine small functions; for example, the root node $\otimes$ in Figure 2.11.

We are concerned with the problem of mutation rate. In general, to prevent GP from acting as a random search algorithm, the mutation probability is much smaller than the reproduction probability and the crossover probability. However, a small probability of mutation not only limits the possibility of emergent new individuals but also causes individuals to remain in the local optimums for an extended period of time. An adaptive mutation probability tuning method would be useful and in fact necessary in this case.

As mentioned above, many different methods based on GP are proposed to solve problems of classification, feature selection, and feature construction. However, among those in our survey, only the work done by Muni et al. [59] integrates a *classifier*, a *feature selector*, and a *feature constructor* in one approach. Combining feature selection and feature construction can be useful because they are needed provided raw features are not suitable for the target task. One may use feature construction techniques to generate many features but most of them can be useless. On the other hand, features remains after feature selection can be insufficient.

In summary, we have following research questions:

RQ 1. How can discriminant functions generated by GP achieve high classification accuracy?

RQ 2. How to tune the mutation probability automatically and smoothly?

RQ 3. How to reduce training time?

RQ 4. How to combine several good short discriminant functions in order to obtain one long function?

RQ 5. How to accomplish feature selection and feature generation?

RQ 6. How to integrate feature selection and feature generation techniques into one system?

A layered multi-population GP method will be proposed and explained in the next section. The proposed method not only solves classification problems by generating short discriminant functions efficiently, but also is capable of integrating feature selection and feature construction.

Chapter 3

Research Design

3.1 Introduction

The method proposed in this paper is called layered genetic programming (*LAGEP*). LAGEP arranges multiple populations in a layered architecture. Populations in the same layer evolve themselves with an identical training set. They store the function values of their best individuals within the training set into a new dataset. Such a dataset becomes the training set of the successive layer. After all layers have finished the evolutionary process, the output of the final layer is the result of LAGEP.

In this chapter, we at first introduce the design of single population GP. A mutation probability tuning method is proposed. In section 3.3, the layered architecture is explained. To deal with multiple-class classification problems, a method called Z-value measure, ZM, is proposed to resolve the *conflict* problem. With a minor modification, LAGEP is capable of completing feature selection. This will be shown in section 3.4.

3.2 LAGEP: Evolving a Population

First, we recall some symbol definitions. A training instance is an object whose class

label is already known. We define a training instance and the training set T as:

$$T = \{x_i | 1 \leq x \leq N\},$$

$$x_i = \{a_{i1}, a_{i2}, \ldots, a_{in}, c_i\},$$

where a_i stands for the *i*-th feature, n stands for the number of features, N is the number of training instances, and c_i, $c_i \in C = \{c_1, c_2, \ldots, c_K\}$, stands for the class label of the instance x_i.

3.2.1 Individual Definitions

As mentioned in Chapter 2, a population subjects to GP is a set of individuals. We denote a population as P, and $P = \{I_1, I_2, ..., I_m\}$; where I_i is an individual and m is the number of individuals of P called *population size*. In this work, an individual is a discriminant function represented by a tree structure. The individual tree is composed of an operation set S_{op}, a variable set S_v, and constant set S_c. Variables are symbolic notations related to features of training instances. A_i stands for the i-th feature of given instances. S_c is a set of predefined constants. We define S_c as ten floating numbers in $[0, 1]$ because the attribute values of classification datasets used in this paper are normalized to $[0, 1]$. (The classification dataset will be described in Chapter 4.) S_{op} can contain logarithmic operations or trigonometric functions, but we use only simple arithmetic operations for two reasons: First, Kishore et al. [35] performed experiments to show that the classification accuracy of using only simple arithmetic operations is sufficiently high. Second, using simple operations are able to reduce computational cost because individuals generated by a compact operations set are simple and efficient. Therefore, S_{op}, S_v, S_c, and I are defined as follows:

$$S_{op} = \{+, -, \times, /\}$$

$$S_v = \{A_1, A_2, \ldots, A_n\}$$

$$S_c = \{0.1, 0.2, 0.3, \ldots, 1.0\}$$

$$I = (S_{op}, S_v, S_c)$$

Note that the division "/" of S_v is a protected division. It returns 1.0 when the denominator is zero.

The structure of individuals is that of a *binary* tree because operations are binary operations. The maximum number of available nodes of an individual is predefined and is called the *individual length*, denoted as IL.

3.2.2 Fitness Function

The fitness function `FitnessFunc` is a function used to evaluate the fitness of every individual. When we perform the training task of a classification problem, one needs to know which class is the *target class*. The target class is the class label for which one trains the system to find solutions. Training instances are divided into *positive instances* if they belong to the target class and *negative instances* if they do not. For a given training instance x, we say of an individual I_i:

I_i *recognizes* training instance x if and only if $I_i(x) \geq 0$;

I_i *repels* training instance x if and only if $I_i(x) < 0$.

We try to find an individual that recognizes positive instances and repels negative instances. An individual is capable of classifying a set of instances. We define a function `Acc` with an individual I and a dataset S by:

$$\texttt{Acc}\,(I, S) = \frac{\text{the number of objects of } S \text{that are correctly classified by } I}{|S|} \tag{3.1}$$

The fitness function, `FitnessFunc`, used in this work is made by

$$f_i = \texttt{FitnessFunction}\,(I_i) = \texttt{Acc}\,(I_i, T)\,. \tag{3.2}$$

We use such a fitness function for two reasons. First, an accurate discriminant function is desired. Second, `FitnessFunc` will be computed many times so it should be as simple as possible.

3.2.3 Validation

The *overfitting problem* occurs when the trained solution excessively adapts to the training set. During the training phase, it is difficult to detect whether the overfitting occurs or not because the test instances are totally unknown. *Validation* process can be used to avoid overfitting. The validation process uses a set of validation instances, V, to check the generalization of individuals. A good individual should derive good performance from the training set, i.e., a high fitness value, and derive high classification accuracy from the validation set. When *all generations* finish, the validation performance of the best individual from within each generation can be obtained with V. For this purpose, we define the measure $score_i$ of an individual I_i as:

$$score_i = \texttt{ScoreFunc}(I_i) = f_i + Acc\,(I_i, V)\,. \tag{3.3}$$

The population's best individual is the one that has greatest score and is denoted as Φ [75].

3.2.4 Elitism Evolution Strategy and the Evolutionary Flowchart

Many evolutionary strategies have been proposed. In this work we use the *elitism* strategy in which only favored individuals remain to next generation. At first, reproduction is no longer governed by probability. Once the fitness evaluation process has finished, a predefined number, r, of superior individuals are reproduced into next generation directly. In order to maintain the diversity of the population, r should be small.

The crossover operator and the mutation operator are also modified. For the crossover, we compare the fitness values of the parent individuals, f_i and f_j, to the fitness values of two offspring, $f_{i'}$ and $f_{j'}$. The best two of these four individuals will be inserted into the next population. Similarly, in the case of mutation, only the one which has highest fitness value between the parent and the mutant can survive.

We use the deterministic tournament selection method to select individuals. This

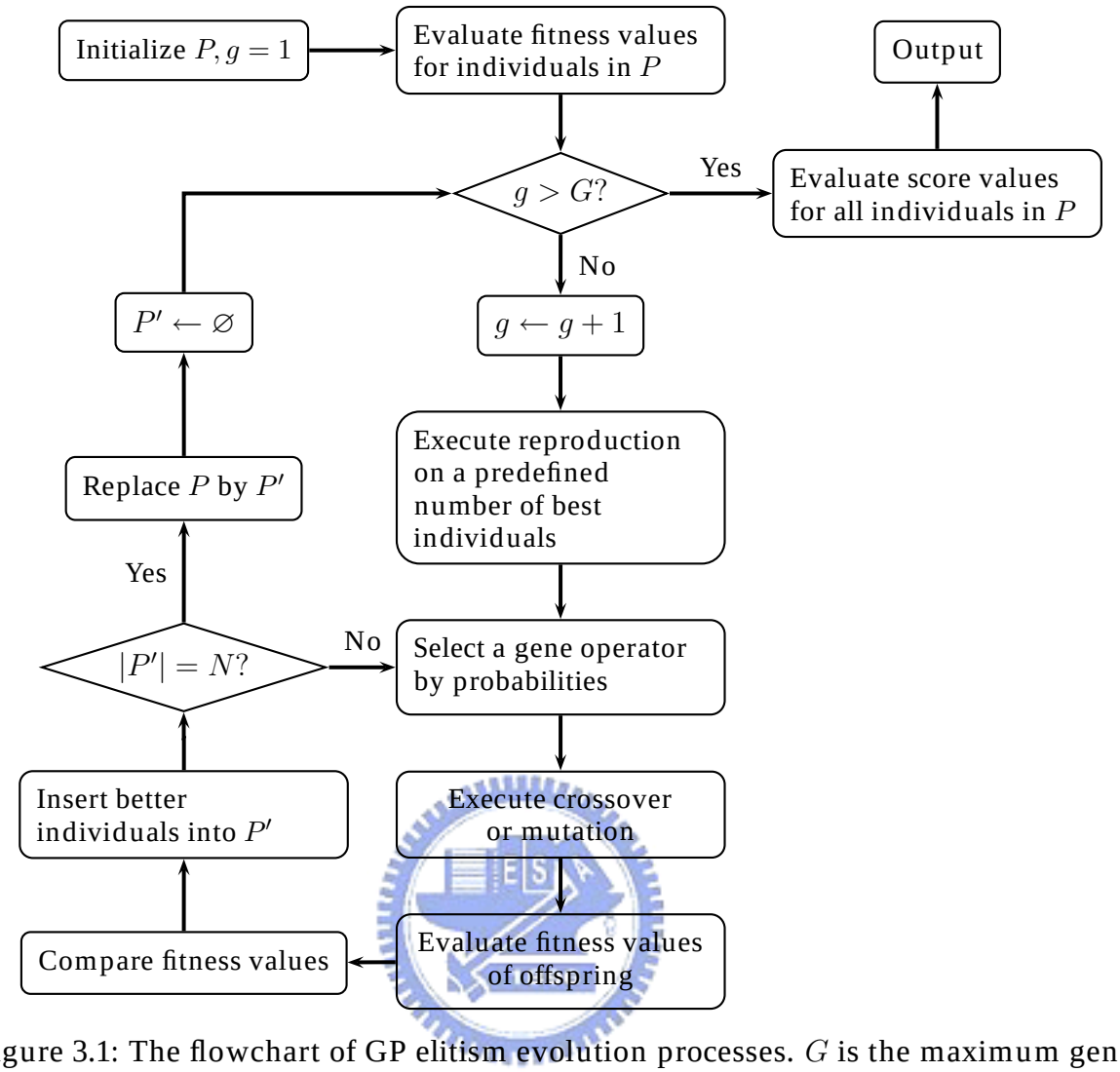


Figure 3.1: The flowchart of GP elitism evolution processes. G is the maximum generation, P is the current population, P' is the population of next generation, and N is the population size.

method at first chooses a number of individuals, called *tournament size*, from the population at random and then returns the individual with the highest fitness value.

The modified GP evolution flowchart using elitism is shown in Figure 3.1.

3.2.5 AMPT: Adaptive Mutation Probability Tuning

The mutation operator is capable of generating individuals with new structures and is mainly used to escape local optimums. Given a high probability of mutation, the population tends to generate diverse individuals instead of utilizing solutions present

in extant individuals. Moreover, a high probability of mutation makes the GP system becomes a random search model, with it is difficult to converge and generate stable results. Such problems rarely arise however because the probability of mutation is usually much lower than the probability of crossover. As a result, there are insufficient opportunities for an individual to be mutated so that the diversity of the population is limited. We denote probabilities of executing crossover and mutation with p_c and p_m, respectively. Since there is no guide to define p_c and p_m, we propose a method called *adaptive mutation probability tuning* (*AMPT*) to raise p_m so that the mutation operator can perform more frequently when the generation increases.

The *AMPT* method tunes p_m according to fitness values. In case individuals have similar fitness values, *AMPT* is triggered to increase p_m. Otherwise, the population uses the initially given p_m. Moreover, because $p_m + p_c = 1$, to increase p_m implies to decrease p_c. *AMPT* is performed every generation. At generation g the *AMPT* considers remaining generations to change p_m and p_c by:

$$(p_m, p_c) = \begin{cases} (p_m, p_c) & \text{if } \frac{f_{\text{MAX}}}{f_{\text{AVERAGE}}} > 2, \\ \left(\frac{\alpha}{p_c+\alpha}, 1-\frac{\alpha}{p_c+\alpha}\right), \alpha = p_m \left(\frac{p_c}{p_m}\right)^{\frac{g}{G}} & \text{otherwise.} \end{cases} \tag{3.4}$$

where G is the maximum generation, f_{MAX} is the fitness value of the best individual in generation g; and f_{AVERAGE} is the average fitness value of all individuals in generation g. From this formula, p_m increases smoothly and achieves a value of $\frac{1}{2}$ at the final generation, which means that the mutation operator and the crossover operator have the same chance to be selected.

We draw the curve of p_m in Figure 3.2 under the following conditions: $G = 100$, $(p_m, p_c) = (0.05, 0.95)$; and f_{MAX} should never exceed $2f_{\text{AVERAGE}}$ during these 100 generations. This curve shows that p_m increases smoothly and terminates at 0.5 at the 100th generation.

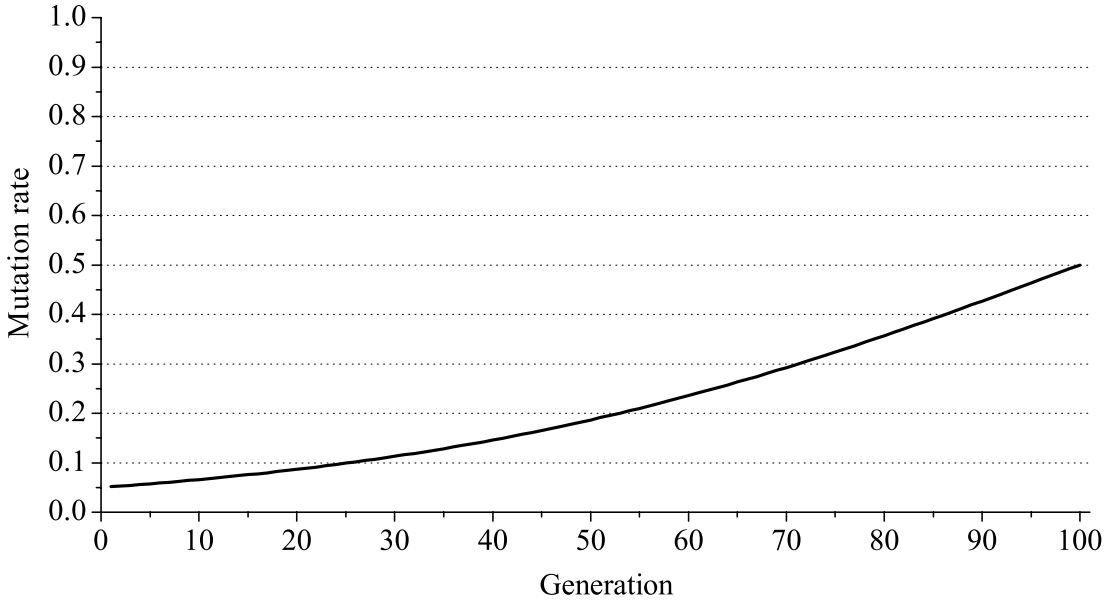


Figure 3.2: The growing curve of p_m using *AMPT*.

3.3 LAGEP: Evolving Layers

At first, we illustrate the LAGEP architecture in Figure 3.3, which provides an overview of the proposed method. Then we will introduce LAGEP in section 3.3.1. To solve the multi-class problem, a conflict resolution method is proposed in section 3.3.3. Finally, an example is given to demonstrate LAGEP.

3.3.1 Layered Architecture

LAGEP composes a number of *layers*. Each layer contains a set of populations. A new training set is produced by the best individuals generated from such populations with the training set of the layer. Populations in the successive layers will use the new training set to evolve individuals. The results are obtained from the final layer.

A layer has a particular variable set and a particular training set. A layer L_i is defined by:

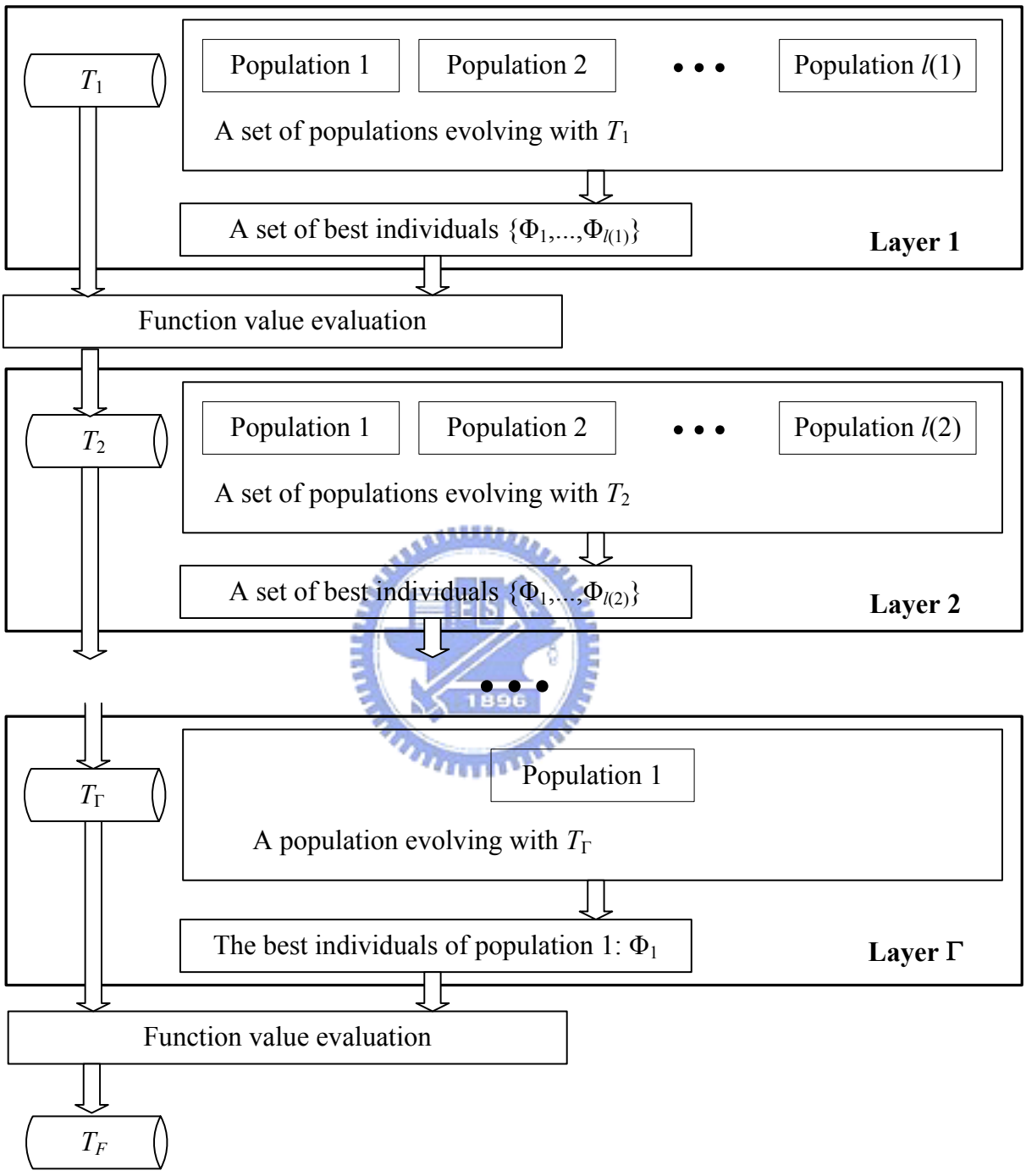


Figure 3.3: Flowchart of LAGEP. Γ is the number of layers, $l(i)$ stands for the number of populations in layer i.

$$L_i = \left(P_{i1}, P_{i2}, \ldots, P_{il(i)}, T_i, S_v^i\right)$$

where P_{ij} is a population, $l\left(i\right)$ is the number of populations in L_i, and S_v^i is the variable set used for populations of L_i.

A layer is an isolated multi-population GP (IMGP) model. We use the IMGP model [18] [19] because it is simpler than PADGP. Every population in the IMGP model is independent of others. The evolutionary algorithm of each single population does not require any change.

A particular training set T_i of L_i is prepared and is used for the evolutionary processes of L_i's populations. For the first layer L_1, its training set T_1 is actually the original given training set. Each population, as mentioned in previous sections, outputs an individual whose score is the highest. Since an individual is a discriminant function, a set of discriminant functions Φ^* of L_i is derived:

$$\Phi_i^* = \left\{\Phi_{i1}, \Phi_{i2}, \ldots, \Phi_{il(i)}\right\}$$

Based on these discriminant functions and training instances of T_i, a new training set T_{i+1} and a new variable set S_v^{i+1} can be constructed by

$$T_{i+1} = \left\{ x_{(i+1)j} \;\middle|\; \begin{array}{l} x_{(i+1)j} = \left(a_{(i+1)j1}, \ldots, a_{(i+1)jk}, \ldots, a_{(i+1)jl(i)}, c_{(i+1)j}\right), \\ \qquad a_{(i+1)jk} = \Phi_{ik}\left(x_{ij}\right), c_{(i+1)j} = c_{ij}, x_{ij} \in T_i, 1 \leq j \leq N \end{array} \right\}$$

$$S_v^{i+1} = \left\{A_{(i+1)1}, A_{(i+1)2}, \ldots, A_{(i+1)l(i)}\right\}$$

An attribute of an instance $x_{(i+1)j}$ of T_{i+1} is made by a corresponding instance x_{ij} of T_i and a corresponding discriminant function Φ_{ij}. When T_{i+1} has been constructed, we say an *era* ends and then the layer L_{i+1} is able to begin its evolutionary process with T_{i+1}.

We define LAGEP as:

$$\text{LAGEP} = \left\{L_i \,|1 \leq i \leq \Gamma\right\}$$

where Γ is the number of layers. The last layer is designed to have only one population. Although having more populations is possible, in this work, we use this design to

simplify the LAGEP system. The result of LAGEP is a single discriminant function and is denoted as Φ. Such a function evaluates the training set to generate the set of training results TF. For a K-class classification problem, LAGEP has to be trained K times with K different target classes. The K different training results $\{TF_1, TF_2, \ldots, TF_K\}$ are collected to represent training results with respect to K classes.

3.3.2 Advantages of Using LAGEP

In this section, we describe the advantages of using LAGEP. First, we show that the result of LAGEP is a composed of small discriminant functions. The result of the last layer of LAGEP is a function Φ which can be represented by:

$$\Phi = \Phi_\Gamma \left(A_{\Gamma 1}, A_{\Gamma 2}, \ldots, A_{\Gamma j}, \ldots, A_{\Gamma l(\Gamma-1)} \right)$$

where

$$\begin{aligned}
A_{\Gamma j} &= \Phi_{(\Gamma-1)j} \left(A_{(\Gamma-1)1}, A_{(\Gamma-1)2}, \ldots, A_{(\Gamma-1)j}, \ldots, A_{(\Gamma-1)l(\Gamma-2)} \right), \Phi_{(\Gamma-1)} \in \Phi^*_{(\Gamma-1)} \\
A_{(\Gamma-1)j} &= \Phi_{(\Gamma-2)j} \left(A_{(\Gamma-2)1}, A_{(\Gamma-2)2}, \ldots, A_{(\Gamma-2)j}, \ldots, A_{(\Gamma-2)l(\Gamma-3)} \right), \Phi_{(\Gamma-2)j} \in \Phi^*_{\Gamma-2} \\
A_{(\Gamma-2)j} &= \Phi_{(\Gamma-3)j} \left(A_{(\Gamma-3)1}, A_{(\Gamma-3)2}, \ldots, A_{(\Gamma-3)j}, \ldots, A_{(\Gamma-3)l(\Gamma-4)} \right), \Phi_{(\Gamma-3)j} \in \Phi^*_{\Gamma-3} \\
&\vdots \\
A_{3j} &= \Phi_{2j} \left(A_{21}, A_{22}, \ldots, A_{2j}, \ldots, A_{2l(1)} \right), \Phi_{2j} \in \Phi^*_2 \\
A_{2j} &= \Phi_{1j} \left(A_{11}, A_{12}, \ldots, A_{1j}, \ldots, A_{1n} \right), \Phi_{1j} \in \Phi^*_1.
\end{aligned}$$

A_{1j} comes from the original given training set. Such expansion shows that the function is a long function composed of a number of functions generated by layers.

Second, a layer has a high probability of having a higher fitness value than the preceding layer. The training instance $x_{(i+1)j} \in T_{i+1}$ is derived from $x_{ij} \in T_i$ through the set of discriminant functions, i.e., $x_{ij} \xrightarrow{\Phi^*_i} x_{(i+1)j}$. Spaces of T_{i+1} and T_i may be different. We illustrate this situation in Figure 3.4. In Figure 3.4, light gray points and black points are training instances belonging to class C_1 and C_2, respectively. Training instance of T_1 may have interleaved disorderly in the space. After the discriminant

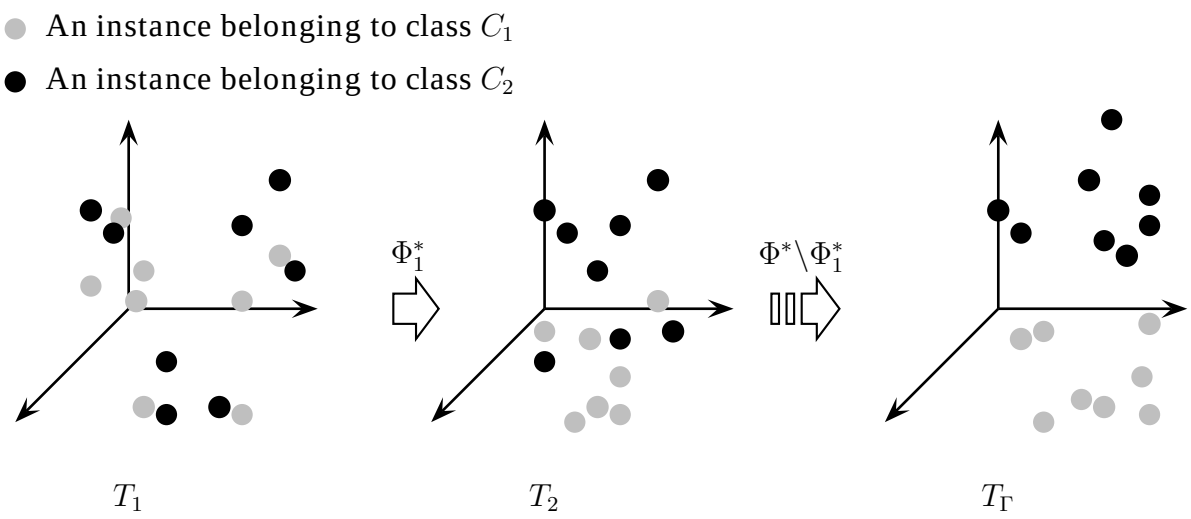


Figure 3.4: An example of feature transformation.

function Φ_1^* transform T_1 into a new space T_2, training instances can distribute separately with respect to different class. Finally, as what we expect, training instances can be easily separated because those they have the same class are closely distributed over the space.

Populations of L_i produce a set of functions to separate instances of T_i. The functional value of a discriminant function for an instance provides classification information. The layer architecture is capable of sending such information to next layer, L_{i+1}, to discover a set of discriminant functions. Therefore, most of the positive instances will have positive attributes and most of negative instances will have negative attributes. Training instances in later layers should be easier to be classified. Consider two populations, P_{ij} and $P_{(i+1)k}$, where $P_{ij} \in L_i$ and $P_{(i+1)k} \in L_{i+1}$. Let TP, FP, FN, and TN be the numbers of true positive, false positive, false negative, and true negative, respectively, obtained by the function Φ_{ij}. We have:

1. The feature $A_{(i+1)j}$ is positive in $TP + FP$ training instances of T_{i+1}.
2. The feature $A_{(i+1)j}$ is negative in $TN + FN$ training instances of T_{i+1}.
3. For positive instances, there are TP of them having positive $A_{(i+1)j}$.
4. For negative instances, there are TN of them having negative $A_{(i+1)j}$.

Considering an instance $I \in P_{(i+1)k}$, I and Φ_{ij} will have the same fitness value as long as $I = A_{(i+1)j}$. Since an individual containing only one variable is very likely to appear

in a population, with sufficient number of individuals, the fitness value of the best individual in $P_{(i+1)k}$ is very likely to be better than the maximum fitness value of P_{ij}.

3.3.3 The Testing Phase and Z-value measure method, ZM

Defining the test set as U, to predict the class of a test instance $y_i \in U$, we substitute y_i for K trained LAGEPs responding to K different classes. The classification result of y_i is represented by a vector r_i:

$$r_i = (r_{i1}, r_{i2}, \ldots, r_{iK}), r_{ij} = \begin{cases} 1 & \text{if } \Phi_k(y_i) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

A problem called *conflict* occurs when $\sum_j r_{ij} \neq 1$, y_i is classified into two or more classes. This problem can be avoided by executing functions in the proper sequence, or be resolved by using additional techniques. Researchers have developed creative methods to solve the conflict problem. Kishore et al. [35] proposed a method to evaluate *strength of association* (*SA*) measured by each GP classification expression (GPCE). Ambiguous data is assigned to the class whose corresponding GPCE has the largest SA. They further used heuristic rules to improve accuracy. Muni et al. [58] proposed two methods to resolve the conflict problem: a heuristic rule-based scheme and a weighting scheme. Here we propose a method called ZM using the Z-value measurement based on statistical theorems. ZM employs means and standard deviations of training results to estimate the Z-value of a conflict test instance. The class of such instance is determined to be class C_i if its corresponding Z-value is the minimum. Furthermore, in case the test instance is not classified to any class, we also use the Z-value to find a probable class rather than to directly assign it to a special class *REJECT*. An instance is assigned to class *REJECT* only when it has multiple minimum Z-values.

We introduce the main steps of Z-value measure in this section. First, we compute the vectors μ, standing for the mean, and σ^2, standing for the standard deviation, of the training results $\{TF_1, TF_2, \ldots, TF_K\}$. Since every instance x_i in a TF is a scalar, μ

and σ^2 can be computed by

$$\mu = (\mu_1, \mu_2, \ldots, \mu_K), \text{ where}$$
$$\mu_i = \frac{\sum_{x_k \in TF_i, c_k = C_i} x_k}{\text{the number of training instances belonging to class } C_i}$$
$$\sigma^2 = \left(\sigma_1^2, \sigma_2^2, \ldots, \sigma_K^2\right), \text{ where}$$
$$\sigma_i^2 = \frac{\sum_{x_k \in TF_i, c_k = C_i} (x_k - \mu_i)^2}{\text{the number of training instances belonging to class } C_i}$$

Second, for a test instance y_i, if $\sum_j r_{ij} = 1$, we do not have to execute ZM for y_i. If $\sum_j r_{ij} = 0$, we change all elements of r_i to 1. A K-dimensional vector Z_i is computed by

$$Z_i = (Z_{i1}, Z_{i2}, \ldots, Z_{iK}),$$
$$Z_{ij} = \begin{cases} \frac{|\Phi_{class=j}(y_i) - \mu_j|}{\sigma_j} & \text{if } r_{ij} = 1 \\ \infty & \text{otherwise} \end{cases}, 1 \le j \le K \quad (3.6)$$

where $\Phi_{class=j}$ stands for the function whose target class is C_j. We assign y_i to class C_j if Z_{ij} is the minimum among Z_i.

3.3.4 LAGEP: An Example

In this section, we illustrate LAGEP by solving a 2-class problem, i.e. $K = 2$, with nine training instances selected from the first distribution of *cancer* problems [65]. TABLE 3.1 shows the training set T where **M** stands for the class *malignant* and **B** stands for

Table 3.1: LAGEP example: Nine training instances selected from *cancer* problem of Proben1 [65].

$T = T_1$	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	Class
x_{11}	0.20	0.10	0.10	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{12}	0.20	0.10	0.10	0.10	0.20	0.10	0.30	0.10	0.10	**M**
x_{13}	0.50	0.10	0.10	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{14}	0.50	0.40	0.60	0.80	0.40	0.10	0.80	1.00	0.10	**B**
x_{15}	0.50	0.30	0.30	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{16}	0.20	0.30	0.10	0.10	0.30	0.10	0.10	0.10	0.10	**M**
x_{17}	0.30	0.50	0.70	0.80	0.80	0.90	0.70	1.00	0.70	**B**
x_{18}	1.00	0.50	0.60	1.00	0.60	1.00	0.70	0.70	1.00	**B**
x_{19}	1.00	0.90	0.80	0.70	0.60	0.40	0.70	1.00	0.30	**B**

the class *benign*. The settings of this example are shown as follows:

the target class is **M**

$$\Gamma = 2$$

$$IL = 15$$

$$\text{LAGEP} = (L_1, L_2)$$

$$L_1 = \left(P_{11}, P_{12}, P_{13}, T_1, S_v^1\right)$$

$$L_2 = \left(P_{21}, T_2, S_v^2\right)$$

$$S_v^1 = \{A_1, A_2, \ldots, A_9\}$$

$$S_v^2 = \{A_{11}, A_{12}, A_{13}\}\,. \tag{3.7}$$

Step1: L_1 performs evolutionary process. The set $\Phi_1^* = \{\Phi_{11}, \Phi_{12}, \Phi_{13}\}$ containing

Table 3.2: LAGEP example: New training set T_2 generated by L_1.

T_2	A_{11}	A_{12}	A_{13}	Class
x_{21}	0.9000	0.7000	1.9000	**M**
x_{22}	0.9000	0.3667	2.9000	**M**
x_{23}	0.9000	0.4000	1.9000	**M**
x_{24}	0.0000	−0.6000	−0.2000	**B**
x_{25}	0.9000	0.2000	1.9000	**M**
x_{26}	0.9000	2.7000	0.9000	**M**
x_{27}	−0.2000	0.1429	−0.3000	**B**
x_{28}	0.4286	−0.7429	0.3000	**B**
x_{29}	−0.1000	−0.9429	−0.3000	**B**

the three best individuals generated by L_1 is

$$\Phi_{11} = (A_9/A_8) - A_6$$

$$\Phi_{12} = (A_5/A_7) - (A_3 + A_1)$$

$$\Phi_{13} = (A_7/A_8) - A_8$$

A new training set T_2 can be constructed according to Φ_1^*. TABLE 3.2 shows T_2.

Step2: L_2 uses T_2 to evolve its population and generates function Φ_{21}

$$\Phi_{21} = A_{11} + A12$$

TABLE 3.3 shows that Φ_{21} achieves perfect classification accuracy on T_2 and shows the training results as well. The solution of LAGEP for class **M** can be expanded as $\Phi = A_{11} + A_{12} = (A_9/A_8) - A_6 + (A_5/A_7) - (A_3 + A_1)$. Here we omit the details of the training LAGEP with target class **B** and show its results along with the training results of LAGEP with target class **M** in TABLE 3.4. The validation process is also omitted to simplify this example.

For two given test instances y_1 and y_2, we use the two trained LAGEPs corre-

Table 3.3: LAGEP example: Training results.

TF	A_{21}	Class
x_{31}	1.6000	**M**
x_{32}	1.2667	**M**
x_{33}	1.3000	**M**
x_{34}	-0.6000	**B**
x_{35}	1.1000	**M**
x_{36}	3.6000	**M**
x_{37}	-0.0571	**B**
x_{38}	-0.3143	**B**
x_{39}	-1.0429	**B**

sponding to class **M** and **B** in y_1 and y_2 and obtain

$$\Phi_M(y_1) = 0.3, \Phi_B(y_1) = -0.4$$

$$\Phi_M(y_2) = 0.3, \Phi_B(y_2) = 0.2$$

We determine the class label of y_1 to be class **M**. The conflict problem occurs at y_2. Through the ZM method, Z-values of y_2 corresponding to class **M** and **B** are 1.5886 and 1.0215, respectively. y_2 is classified into class **B** because Z_{2B} is smaller than Z_{2M}.

3.4 LAGEP-FS: LAGEP with Feature Construction and Feature Selection

In this section, we propose a type of LAGEP called LAGEP-FS to achieve feature construction and feature selection *simultaneously*.

Feature construction is completed by producing new features using discriminant

Table 3.4: LAGEP example: Training results for target class **M** and **B**.

	M	**B**	Class
x_{31}	1.6000	−0.6000	**M**
x_{32}	1.2667	−0.6000	**M**
x_{33}	1.3000	−0.3000	**M**
x_{34}	−0.6000	0.0000	**B**
x_{35}	1.1000	−0.1000	**M**
x_{36}	3.6000	−0.4000	**M**
x_{37}	−0.0571	0.5000	**B**
x_{38}	−0.3143	1.5000	**B**
x_{39}	−1.0429	1.2000	**B**

functions generated from a layer as mentioned in previous sections. Instead of using new features only, LAGEP-FS combines original features and new features into a new training set T_{i+1} after L_i has finished its evolutionary process. Original features in existing individuals have chances to be used in successive layers so that the so-called *nesting problem* can be avoided. LAGEP-FS combines feature construction and feature selection in single system that can simplify these two major tasks of dealing with features.

Differences between LAGEP and LAGEP-FS are introduced in the following section. Two proposed feature selection methods are described afterward. Finally, a brief example shows how LAGEP-FS works.

3.4.1 Architecture

LAGEP-FS not only uses layers to generate new features but also keeps original features. As mentioned in LAGEP sections, layer L_i evolves with a particular training set

T_i, and generates a new training set T_{i+1} as

$$T_{i+1} = \left\{ x_{(i+1)j} \;\middle|\; \begin{array}{l} x_{(i+1)j} = \left(a_{(i+1)j1}, \ldots, a_{(i+1)jk}, \ldots, a_{(i+1)jl(i)}, c_{(i+1)j},\right), \\ a_{(i+1)jk} = \Phi_{ik}\left(x_{ij}\right), c_{(i+1)j} = c_{ij}, x_{ij} \in T_i, 1 \le j \le N \end{array} \right\}$$

$$S_v^{i+1} = \left\{A_{(i+1)1}, A_{(i+1)2}, \ldots, A_{(i+1)l(i)}\right\}$$

In LAGEP-FS, T_{i+1} and S_v^{i+1} have similar definitions:

$$T_{i+1} = \left\{x_{(i+1)1}, \ldots, x_{(i+1)j}, \ldots, x_{(i+1)n}\right\} \text{ where}$$

$$x_{(i+1)j} = \left(a_{(i+1)j1}, \ldots, a_{(i+1)jn}, a_{(i+1)j(n+1)}, \ldots, a_{(i+1)j(n+k)}, \ldots, a_{(i+1)j(n+k+l(i))}, c_{(i+1)j}\right),$$

$$a_{(i+1)j\alpha} = \begin{cases} a_{1j\alpha} & \text{if } 1 \le \alpha \le n+k, k = \sum_{q=1}^{i-1} l(q) \\ \Phi_{i(\alpha-m-k)}(x_{ij}) & \text{otherwise} \end{cases}$$

$$c_{(i+1)j} = c_{ij}, x_{ij} \in T_i, \Phi_{iq} \in \Phi_i^*, 1 \le j \le N$$

$$S_v^{i+1} = \bigcup_{j=1}^{i} S_v^j = \left\{A_1, \ldots, A_n, A_{11}, \ldots, A_{1l(1)}, A_{21}, \ldots, A_{(i+1)l(i)}\right\},$$

The definition of S_v^{i+1} may raise ambiguity in the case that $n > 10$ or $l(i) > 10$. A symbol A_{11} could be the eleventh original feature or the feature constructed by first layer. In order to avoid ambiguity, we define S_v^{i+1} as

$$S_v^{i+1} = \left\{A_1, \ldots, A_n, A_{n+1}, \ldots, A_{n+l(1)}, A_{n+l(1)+1}, \ldots, A_{n+k+l(i)}\right\}, k = \sum_{q=1}^{i-1} l(q)$$

Each new training instance $x_{(i+1)j} \in T_{i+1}$ have original features $\{a_{ij1}, \ldots, a_{ikm}\}$, constructed features $\left\{a_{ij(m+1)}, \ldots, a_{ij(m+k)}\right\}, k = \sum_{q=1}^{i-1} l(q)$ generated by $L_1, \ldots, L_{i-1}$, and new features $\left\{\Phi_{i1}(x_{ij}), \ldots, \Phi_{il(i)}(x_{ij})\right\}$ generated by preceding layer L_i.

After LAGEP-FS is completed, we will have $\sum_{q=1}^{\Gamma} l(q)$ features. The last feature is in fact constructed by the function $\Phi_{\Gamma 1}$, which is the training result.

LAGEP-FS uses a different fitness function and the constant set S_c. In order to let fitness values be more sensitive, the fitness function `FitnessFunc` in LAGEP-FS is

$$\textit{sensitivity} = \frac{TP}{TP+TN} = \frac{\text{number of correctly classified positive instances}}{\text{number of positive instances}}$$

$$\textit{specificity} = \frac{TN}{FP+TN} = \frac{\text{number of correctly classified negative instances}}{\text{number of negative instances}}$$

$$\texttt{FitnessFunc}(I_i) = \textit{sensitivity} \times \textit{specificity}$$

Moreover, since the number of used features is of the utmost concern, we reduce the probability of returning a constant when an individual acquires a leaf node. Therefore, S_c is enlarged to $S_c = \{1, 2, \ldots, 100\}$ because constants will not have much of a chance to be used.

3.4.2 Proposed Feature Selection Methods

Two feature selection methods, *M1* and *M2*, each based on different concepts are proposed. *M1* reduces fitness value of an individual using multiple features. *M2*, on the other hand, controls for the possibility of a feature being selected by tuning its weights during the evolutionary process.

M1

This method gives the individuals with more features a lower fitness value so as to reduce their probability of being selected. Instead of adopting a new fitness function, we assign new fitness values to individuals once a generation completed. Using a fitness function that concerns only the number of features selected in an individual may cause an unfit individual to be assigned a higher fitness. Since classification accuracy is the most important factor, *M1* takes a set of individuals rather than single individual into consideration.

First, we partition the population into several subsets. Each subset collects individuals of similar fitness values. For a *sorted* population $P = \{I_1, I_2, \ldots, I_m\}$, where $f_i \geq f_{i+1}$, P is partitioned into subsets $S_1, S_2, \ldots, S_k$ as:

$$S_1 = \{I_1, \ldots, I_i\}, S_2 = \{I_{i+1}, \ldots, I_{i+j}\}, \ldots, S_k = \{I_k, \ldots, I_m\}$$
$$f_i - f_j \leq 0.01 \forall I_i, I_j \in S_t, 1 \leq t \leq k \tag{3.8}$$

Afterward, individuals in a subset S_t are sorted by the **number of features** they have used in ascending order. For a set $S_t = \{I_{t1}, I_{t2}, \ldots, I_{tk}\}$ whose individuals are ordered

in descending order, the new fitness of an individual I_{tj} is

$$f'_{tj} = \left(\frac{(f_{\max} - f_{\min})\,(k - j + 1)}{k} \right) + f_{\min} \tag{3.9}$$

Example

Suppose that in partition $S_i = \{I_{i1}, \ldots, I_{i4}\}$, the numbers of features used in each individual are $(12, 18, 19, 33)$. The fitness values of I_{i1}, I_{i2}, I_{i3}, and I_{i4} are 0.792, 0.800, 0.797, and 0.799, respectively. Their new fitness values thus become

$$f'_{I_{i1}} = \left(\frac{(0.800 - 0.792)(4 - 1 + 1)}{4} \right) + 0.792 = 0.800$$
$$f'_{I_{i2}} = \left(\frac{(0.800 - 0.792)(4 - 2 + 1)}{4} \right) + 0.792 = 0.798$$
$$f'_{I_{i3}} = \left(\frac{(0.800 - 0.792)(4 - 3 + 1)}{4} \right) + 0.792 = 0.796$$
$$f'_{I_{i4}} = \left(\frac{(0.800 - 0.792)(4 - 4 + 1)}{4} \right) + 0.792 = 0.794$$

The fitness values of individuals in a partition become uniformly distributed. I_{i1} has the highest fitness value after *M1* because it used the least number of features. On the other hand, although I_{i4} has a fitness value 0.799, its new fitness value is the lowest one because it used the greatest number of features.

M2

Diversity of a population will be influenced by irrelevant features, especially when the given problem is a high-dimensional one. This is explained by the limitation in both the number of individuals of a population and the number of available nodes of an individual. Instead of tuning the fitness value of each individual, *M2* tunes the weights of all features, generation by generation. Features with lower weights are unlikely to be chosen when an individual acquires a new feature from the variable set.

For a population $P \in L_i$, once a generation is completed we calculate the variable usage of individuals. P is divided equally into two sets HF and LF; where HF is the set of better individuals and LF is the set of the remaining individuals. We further

define the variable usage φ^H and φ^L by

$$\varphi^H = \left(\varphi_1^H, \varphi_2^H, \ldots, \varphi_k^H\right)$$
$$\varphi^L = \left(\varphi_1^L, \varphi_2^L, \ldots, \varphi_k^L\right)$$
$$k = \begin{cases} \sum_{q=1}^{i-1} l(q) & \text{if } P \notin L_1 \\ n & \text{otherwise} \end{cases}$$

where φ_i^H and φ_i^L stand for frequency of variable A_i occurring in HF and LF, respectively. The weight of variable A_i used in L_{i+1} is

$$\text{weight of } A_i = \frac{\left\lceil \frac{\varphi_i^H + 1}{\varphi_i^L + 1} \right\rceil}{\left\lceil \frac{\sum \varphi^H + 1}{\sum \varphi^L + 1} \right\rceil} \tag{3.10}$$

Terminals always have positive weights. We do not remove any of them during the evolutionary process.

Example

Assume that each training instance has three features $\{A_1, A_2, A_3\}$. When a generation is completed, the frequencies of features occurring in HF and LF are $(10, 3, 12)$ and $(2, 8, 12)$, respectively. The weight of A_1 can be calculated by

$$\frac{\left\lceil \frac{10+1}{2+1} \right\rceil}{\left\lceil \frac{25+1}{22+1} \right\rceil} = \frac{\lceil 3.6666 \rceil}{\lceil 1.1304 \rceil} = \frac{4}{2} = 2.0$$

The weight of A_2 is

$$\frac{\left\lceil \frac{3+1}{8+1} \right\rceil}{\left\lceil \frac{25+1}{22+1} \right\rceil} = \frac{\lceil 0.4444 \rceil}{\lceil 1.1304 \rceil} = \frac{1}{2} = 0.5$$

The weight of A_3 is

$$\frac{\left\lceil \frac{12+1}{12+1} \right\rceil}{\left\lceil \frac{25+1}{22+1} \right\rceil} = \frac{\lceil 1.0000 \rceil}{\lceil 1.1304 \rceil} = \frac{1}{2} = 0.5$$

The probability of A_1 being selected is $2/3$. Clearly, A_1 has four times higher weight than A_2 and A_3 have.

Table 3.5: LAGEP-FS example: Nine training instances selected from *cancer* problem of Proben1 [65].

$T = T_1$	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	Class
x_{11}	0.20	0.10	0.10	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{12}	0.20	0.10	0.10	0.10	0.20	0.10	0.30	0.10	0.10	**M**
x_{13}	0.50	0.10	0.10	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{14}	0.50	0.40	0.60	0.80	0.40	0.10	0.80	1.00	0.10	**B**
x_{15}	0.50	0.30	0.30	0.10	0.20	0.10	0.20	0.10	0.10	**M**
x_{16}	0.20	0.30	0.10	0.10	0.30	0.10	0.10	0.10	0.10	**M**
x_{17}	0.30	0.50	0.70	0.80	0.80	0.90	0.70	1.00	0.70	**B**
x_{18}	1.00	0.50	0.60	1.00	0.60	1.00	0.70	0.70	1.00	**B**
x_{19}	1.00	0.90	0.80	0.70	0.60	0.40	0.70	1.00	0.30	**B**

3.4.3 An Example

This section demonstrates LAGEP-FS in the first distribution of *cancer* problems [65], as used in the LAGEP example used in section 3.3.4. TABLE 3.5 shows the nine training samples of the training set T where **M** stands for the class *malignant* and **B** stands for the class *benign*.

In this example, the target class is **M**; $\Gamma = 2$; $IL = 255$; LAGEP-FS$= (L_1, L_2)$, $L_1 = (P_{11}, P_{12}, T_1, S_v^1)$;$L_2 = (P_{21}, T_2, S_v^2)$, $S_v^1 = \{A_1, A_2, \ldots, A_9\}$; and $S_v^2 = \{A_1, A_2, \ldots, A_9, A_{11}, A_{12}\}$.

Step1: L_1 evolves with T for G generations. We obtain two discriminant functions

Table 3.6: LAGEP-FS example: New training set generated by L_1.

T_2	A_1	$\cdots$	A_9	A_{10}	A_{11}	Class
x_{21}	0.20	$\cdots$	0.10	0.1991	76.1066	**M**
x_{22}	0.20	$\cdots$	0.10	0.1991	76.1066	**M**
x_{23}	0.50	$\cdots$	0.10	1.6225	175.7748	**M**
x_{24}	0.50	$\cdots$	0.10	-7.4600	-24.2714	**B**
x_{25}	0.50	$\cdots$	0.10	0.2633	42.1644	**M**
x_{26}	0.20	$\cdots$	0.10	0.1364	22.7037	**M**
x_{27}	0.30	$\cdots$	0.70	-2.3482	-514.8546	**B**
x_{28}	1.00	$\cdots$	1.00	-440.0000	-9126.0333	**B**
x_{29}	1.00	$\cdots$	0.30	-192.1067	-1135.0109	**B**

Φ_{11} and Φ_{12} as

$$\Phi_{11} = A_9 - \left(\begin{array}{l} (((A_1 \times (70 \times (A_4 \times A_1))) \times ((A_1 + A_6) - (A_6 - A_2))) \times (A_1/A_2)) \times \\ ((A_1 + (A_6 + (A_2 + (A_4 \times A_3)))) - 1) \end{array} \right)$$

$$\Phi_{12} = \left(A_1 \times \left(\begin{array}{l} ((9 + (40/A_2)) - (8/\left((A_6/A_4) - (A_2/8)\right))) - \\ ((((29 - A_1) + A_2) \times (A_1 \times A_9)) \times (((23 \times A_6) \times (15 - A_6)) + A_4)) \end{array} \right) \right) - A_6$$

Note that we do not illustrate either of the feature selection methods *M1* or *M2* in this example. Such methods are performed during each generation not after all generations are completed.

The original nine features and the two new features, A_{11} and A_{11}, construct T_2, which is shown in 3.6.

Step2: Next, L_2 uses T_1 to evolve its populations and generates Φ_{21}, which is

$$\Phi_{21} = (41/\left(((A_{10} + ((80 \times A_{10}) + A_4))/A_8\right)/A_4)) + (9 \times (A_1/A_{11}))$$

Table 3.7: LAGEP-FS example: Training results.

	A_1	$\cdots$	A_9	A_{10}	A_{11}	A_{12}	Class
x_{31}	0.20	$\cdots$	0.10	0.1991	76.1066	0.0489	**M**
x_{32}	0.20	$\cdots$	0.10	0.1991	76.1066	0.0489	**M**
x_{33}	0.50	$\cdots$	0.10	1.6225	175.7748	0.0287	**M**
x_{34}	0.50	$\cdots$	0.10	-7.4600	-24.2714	-0.2398	**B**
x_{35}	0.50	$\cdots$	0.10	0.2633	42.1644	0.1259	**M**
x_{36}	0.20	$\cdots$	0.10	0.1364	22.7037	0.1161	**M**
x_{37}	0.30	$\cdots$	0.70	-2.3482	-514.8546	-0.1784	**B**
x_{38}	1.00	$\cdots$	1.00	-440.0000	-9126.0333	-0.0018	**B**
x_{39}	1.00	$\cdots$	0.30	-192.1067	-1135.0109	-0.0098	**B**

Then we combines Φ_{21} and features $A_1, \ldots, A_{11}, A_{12}$ of T_1 to build T_2 which is shown in TABLE 3.7, where A_{21} is the result of the training samples. Since A_{11} and A_{12} are made from Φ_{11} and Φ_{12} as mentioned above, the set of used original features is $\{A_1, A_2, A_3, A_4, A_6, A_8, A_9\}$.

Chapter 4

Experiment Study

4.1 Introduction

In this section we describe the experiments performed and analyze the experimental results of LAGEP and LAGEP-FS. To conduct the experiments described in this section, we developed a system based on the LAGEP Project [50] executed under an ACER VT7600GL, which is equipped with a single 3.0GHz processor and 1.5GB memory.

First we introduce the selected datasets. Then we have some hypotheses about the experiments based on research questions, as described in Section 2.6. We following conduct experiments to illustrate the performance of the proposed mutation probability tuning method *AMPT*. Classification accuracies and analysis of LAGEP with different settings are demonstrated. Finally, the performance experiment and analysis of LAGEP-FS is given.

4.2 Medical Classification Problems

We used six diagnostic problems selected from the PROBEN1 benchmark set of real-world problems [65], which was also used in [6]. These problems are originally from

Table 4.1: Summary of selected problems.

Problem		Number of classes	Number of features	Training instances	Validation instances	Test instances
heart	(HRT)	2	35	152	76	75
horse	(HRS)	3	58	182	91	91
cancer	(CAN)	2	9	350	175	174
diabetes	(DBT)	2	8	384	192	192
gene	(GEN)	3	120	1588	794	793
thyroid	(TRD)	3	21	3600	1800	1800

the UCI repository [13] and have been preprocessed by [65]. The values of all sets are normalized to the continuous range $[0, 1]$. Missing attributes are completed. Every attribute having m possible values is encoded by the *1-of-m* method, i.e., using m binary attributes instead of the single original attribute.

Each problem prepared by PROBEN1 has been divided into three subsets: training, validation, and test. The training set contains the first half of instances. The validation set includes the next 25% of instances. The last 25% of instances are test instances. Therefore, in this paper we do not need to use multi-fold cross-validation to re-separate each problem into a new training set and a new test set. Furthermore, every problem in PROBEN1 has three different compositions with different distributions of instances, i.e., instances are separated into the training set, the validation set, and the test set by three different orders. This should increase confidence that classification results are not influenced by the distributions of the training set, the validation set, and the test set. Therefore, we have 18 problems in total. We summarize these problems in TABLE 4.1.

4.3 Hypotheses

We have some hypotheses for the experiments. As mentioned in Section 2.6, we have six research questions:

RQ 1. How can discriminant functions generated by GP achieve high classification accuracy?

RQ 2. How to tune the mutation probability automatically and smoothly?

RQ 3. How to reduce training time?

RQ 4. How to combine several good short discriminant functions in order to obtain one long function?

RQ 5. How to accomplish feature selection and feature generation?

RQ 6. How to integrate feature selection and feature generation techniques into one system?

Therefore, we have hypotheses corresponding to the six research questions:

H 1. SGP with AMPT can achieve higher classification accuracy to SGP without AMPT.

H 2. Using discriminant functions can obtain high classification accuracy.

H 3. A LAGEP-based classification requires less training time than a SGP-based classification without loss of classification accuracy.

H 4. Populations in successive layers have better learning performance.

H 5. LAGEP can accomplish feature selection and feature generation simultaneously.

Following experiments are conducted to test these hypotheses.

Table 4.2: Experiment setting **ES1** used for *AMPT* experiments.

Parameter	Value
Max generation, G	100
Population size, m	2000
Individual length, IL	511
Tournament size	6
Crossover probability, P_c	0.8
Mutation probability, P_m	0.2

4.4 Experiments of $AMPT$ method

First, we show how performance was improved by using *AMPT*. *AMPT* is originally designed for each population in LAGEP, thus this experiment is conducted with a single population. TABLE 4.2 shows the experimental setting, which is denoted as **ES1**. The population size, maximum generation, and the tournament size are referring to that of [75]. However, the crossover probability and the mutation probability in [75] are 0.35 and 0.65, respectively. Such a combination of P_c and P_m is not suitable for this work. The mutation operator is not supposed to be the primary genetic operator because it is used to escape local optimums. Therefore, we consider the P_c and P_m from [58]. The crossover probability, mutation probability, and reproduction probability of [58] are 0.75, 0.15, and 0.1, respectively. In this paper, the reproduction operator is not performed by probability. We divide the reproduction probability, 0.1, equally into p_c and p_m. The number of available nodes of an individual used in [75] is 650 because the structure of individuals is not the binary tree structure. We use the most approximate number, 511, as the maximum number of available nodes of an individual.

Denoting the **ES1** with *AMPT* and without *AMPT* as **ES1.w** and **ES1.w/o**, respectively, we perform them on the 18 classification problems ten times. TABLE 4.3 and

TABLE 4.4 shows score and accuracy comparisons and paired t-tests between **ES1.w** and **ES1.w/o**. The three values in each cell stand for the highest accuracy, the average accuracy, and the standard variation.

TABLE 4.3 illustrates **ES1.w** outperforms **ES1.w/o** in 11 problems out of 18 problems. TABLE 4.4 shows that *AMPT* is capable of improving the classification performance of the single population GP. **ES1.w** has higher average accuracy than **ES1.w/o** with the exception the CAN3 problem. TABLE 4.4 shows that for most problems, **ES1.w** elucidates better discriminant functions. Hence, the hypothesis H1: "SGP with AMPT can achieve higher classification accuracy to SGP without AMPT." is therefore confirmed. Based on these observations, all of the following experiments use the *AMPT* method.

4.5 LAGEP Experiments

4.5.1 Experimental Settings

In this paper, we use five two-layer LAGEP experimental settings to illustrate the performance of LAGEP. TABLE IX shows six experimental settings including ES1. In this paper, we focus on investigating the behavior of the LAGEP architecture with different numbers of populations rather than with different numbers of layers. TABLE 4.5 shows that all settings use similar numbers of individuals. These settings refer [75] because we compare the results we obtain with the results therein. LAGEP is proposed to utilize short discriminant functions so as to obtain a longer one. We set the individual length of them to be 255. Every classification problem is performed with each experimental setting ten times.

Table 4.3: Learning score and paired t-test comparisons between **ES1** with *AMPT* and **ES1** without *AMPT*.

The three values in each cell stand for highest accuracy, average accuracy, and standard deviation.

Problem	ES1.w	ES1.w / o	Paired t-test	Problem	ES1.w	ES1.w / o	Paired t-test
HRT1	1.8618 1.7898 0.0346	1.8553 1.8003 0.0396	0.2283	DBT1	1.6406 1.5901 0.0270	1.6302 1.5882 0.0233	0.3969
HRT2	1.7566 1.6875 **0.0405**	1.7763 1.7020 **0.0328**	0.1157	DBT2	1.6250 1.5866 **0.0268**	1.6094 1.5810 **0.0199**	0.2555
HRT3	1.8092 1.7477 0.0312	1.7763 1.7414 0.0242	0.2583	DBT3	1.6094 1.5645 0.0279	1.6120 1.5616 0.0266	0.3727
HRS1	1.7967 1.6762 0.0926	1.8242 1.6791 0.1014	0.3695	GEN1	1.9225 1.7915 0.1283	1.9320 1.7884 0.1187	0.4304
HRS2	1.8242 1.6868 0.0933	1.8022 1.6960 0.0793	0.2055	GEN2	1.9332 1.7732 0.1376	1.9200 1.7688 0.1144	0.4235
HRS3	1.8297 1.6908 **0.0951**	1.8407 1.6868 **0.0910**	0.2921	GEN3	1.9402 1.7786 **0.1227**	1.9188 1.7870 **0.1146**	0.3997
CAN1	1.9629 1.9537 0.0048	1.9686 1.9570 0.0061	0.0231*	TRD1	1.9908 1.9779 0.0125	1.9908 1.9615 0.0370	0.0084*
CAN2	1.9800 1.9707 0.0042	1.9771 1.9687 0.0064	0.1317	TRD2	1.9925 1.9719 0.0255	1.9931 1.9707 0.0264	0.3681
CAN3	1.9743 1.9609 0.0107	1.9771 1.9631 0.0088	0.0440*	TRD3	1.9917 1.9685 0.0287	1.9964 1.9678 0.0287	0.4488

* Significance level $\alpha = 0.05$

Table 4.4: Accuracy and paired t-test comparisons between **ES1** with *AMPT* and **ES1** without *AMPT*.

The three values in each cell stand for highest accuracy, average accuracy, and standard deviation. The better average accuracy is marked in bold face.

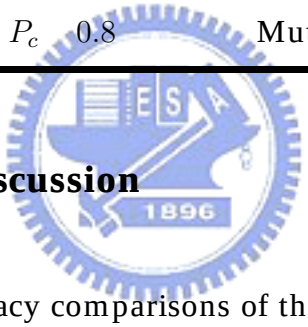
Problem	ES1.w	ES1.w / o	Paired t-test	Problem	ES1.w	ES1.w / o	Paired t-test
HRT1	80.00 **76.80** 3.09	81.33 76.67 2.69	0.4557	DBT1	78.13 **72.50** 2.76	76.56 72.34 3.54	0.4667
HRT2	98.67 **93.33** 3.20	94.67 92.27 2.16	0.2332	DBT2	75.52 **71.25** 2.44	75.00 71.15 1.79	0.4584
HRT3	88.00 **84.40** 2.60	86.67 82.93 2.50	0.0769**	DBT3	77.60 **75.16** 2.53	78.13 74.53 1.46	0.2594
HRS1	71.43 **64.51** 4.61	67.03 62.53 2.28	0.1593	GEN1	89.66 **85.31** 4.37	89.41 85.21 3.10	0.4787
HRS2	67.03 **61.98** 3.78	65.93 60.77 2.64	0.2358	GEN2	90.04 **85.98** 3.95	88.65 83.32 5.97	0.0343*
HRS3	65.93 **61.21** 3.11	60.44 55.27 3.74	0.0010*	GEN3	88.27 **84.59** 3.00	88.15 81.78 5.07	0.0397*
CAN1	98.85 **97.70** 0.72	98.85 97.70 0.77	0.5000	TRD1	97.72 **97.15** 0.47	97.72 95.94 1.74	0.0361*
CAN2	96.55 **94.89** 0.69	94.83 94.08 0.47	0.0022*	TRD2	98.28 **97.33** 1.22	98.67 96.84 1.34	0.0766**
CAN3	97.13 96.32 0.78	97.13 **96.44** 0.45	0.3097	TRD3	98.06 **96.95** 0.92	98.50 96.69 1.38	0.3332

* Significance level $\alpha = 0.05$

** Significance level $\alpha = 0.10$.

Table 4.5: Experimental settings of traditional single population GP and LAGEP.

	Setting name	Γ	$l(i)$	Population size for each	Total number of individuals	IL
Traditional SGP	**ES1**	1	1	2000	2000	511
LAGEP	**ES2**	2	2, 1	666	1998	255
LAGEP	**ES3**	2	3, 1	500	2000	255
LAGEP	**ES4**	2	4, 1	400	2000	255
LAGEP	**ES5**	2	5, 1	333	1998	255
LAGEP	**ES6**	2	6, 1	286	2000	255
Common settings						
Max generation, G		100		Tournament size		6
Crossover probability, P_c		0.8		Mutation probability, P_m		0.2



4.5.2 Analysis and Discussion

TABLE 4.6 shows the accuracy comparisons of the six settings and four different GP structures cited from [75]: G^{3}P for decision trees (G^{3}P D), G^{3}P for fuzzy rule based systems (G^{3}P F), G^{3}P for artificial neural networks (G^{3}P A), and G^{3}P for fuzzy Petri-nets (G^{3}P P). The values in each cell stand for the highest accuracy, the average accuracy and the standard deviation. TABLE 4.7 shows the paired t-test results between **ES1** and other five LAGEP settings.

4.5.2.1 Comparing classification accuracy between SGP, LAGEP and four cited methods

The classification accuracies of the four cited methods are varied. LAGEP has a lower

average accuracy than at least one of the four methods for problems HRS1, HRS3, CAN2, DBT1, and DBT2. However, we found that LAGEP performs amazingly better than the four cited methods for HRT2, HRT3, GEN2, and TRD1. For those problems, the lowest average accuracy of the LAGEP setting is higher than the maximum highest accuracy of any of the four methods.

Comparing SGP, i.e. **ES1**, with the four cited methods, the hypothesis H2: "Using discriminant functions can obtain high classification accuracy." can be accepted since SGP outperforms in many classification problems.

4.5.2.2 Comparing classification accuracy between LAGEP and ES1

First, we compare the classification accuracies between **ES1** and LAGEP settings from TABLE 4.6 and TABLE 4.7. **ES1** has higher average accuracy than at least one LAGEP setting in problems HRT1, HRT3, HRS2, CAN2, CAN3, DBT1, and TRD2. However, the best average accuracy for each problem is obtained by one of the five LAGEP settings. TABLE 4.7 shows that the significant p-value only occurs at the problem TRD3 ($p = 0.0258$) where **ES6** has higher average accuracy. If the significance level α increases to 0.1, four more significance p-values occur at **ES5** with HRS3 ($p = 0.0602$), **ES3** with DBT1 ($p = 0.0621$), **ES5** with GEN2 ($p = 0.0567$), and **ES5** with TRD1 ($p = 0.0552$). By contrast, **ES1** does not significantly outperform any LAGEP setting in the 18 problems.

The classification accuracies obtained by LAGEP and **ES1** are similar. The reason for this situation can be explained by the inadequate generalization of LAGEP and the inconsistent of instances of the training set, the validation set, and the test set. TABLE 4.9 shows the average score values of **ES1** and the population of the second layer of LAGEP settings. The paired t-test between **ES1**'s score values and each LAGEP setting's score values is also conducted as shown in TABLE 4.9. LAGEP achieves significantly better score values than **ES1** except in the case of the TRD problems. **ES1** has higher average score values in TRD1 and TRD3 problems only, but it performs worse than all LAGEP settings in these problems. Furthermore, **ES2**, **ES5**, and **ES6** achieve

significantly better average score values than **ES1** in the CAN2 problem, but TABLE 4.6 shows that **ES2**, **ES5**, and **ES6** have lower average accuracy. In summary, LAGEP has been slightly affected by the **overfitting** problem even though the validation process has already been applied. The overfitting problem is however not serious because we found that some LAGEP settings having highest average score values obtain the highest average accuracies. For instance, considering the CAN1 problem, **ES3** and **ES6** have the highest score values, 1.9627 and 1.9626, respectively, and they both achieve the highest average accuracy, 97.82. If the test set could be more consistent with the training set and the validation set, the enhancement of classification accuracy that LAGEP achieves would be considerable.

4.5.2.3 Comparing elapsed training time

LAGEP not only improves effectiveness but also reduces training time. TABLE 4.8 shows the elapsed training time of six experimental settings. Obviously, LAGEP settings use much less time than single population GP does. The average elapsed training time of every LAGEP setting is at least 10% less than that of **ES1**. The decrease in time is huge for TRD problems where every LAGEP setting uses at most 64% of time that **ES1** does. This phenomenon could be explained by the number of instances present in TRD problems. The evolutionary process of a population spends most of its time evaluating fitness values for individuals. The individual length directly affects the required computation time. LAGEP settings and **ES1** perform under the same conditions with regard to the number of generations and the number of total individuals. LAGEP requires less training time because it uses shorter individuals. Evaluating function values for a short function is efficient. From observations of the classification accuracy and the elapsed time, one can conclude that LAGEP is more efficient than single population GP. Concluding this section and the previous section, the hypothesis H3: "A LAGEP-based classification requires less training time than a SGP-based classification without loss of classification accuracy." is confirmed.

Table 4.6: Accuracy comparisons of six experiment settings and four cited methods [75].

The three values in each cell stand for highest accuracy, average accuracy, and standard deviation. The better average accuracy is marked in bold face.

Problem	**ES1**	**ES2**	**ES3**	**ES4**	**ES5**	**ES6**	G^3P D	G^3P F	G^3P A	G^3P P
HRT1	80.00 76.80 3.09	80.00 77.47 2.98	82.67 **78.27** 3.45	82.67 76.67 2.90	80.00 76.00 2.18	82.67 77.33 3.27	82.10 78.13 2.12	81.66 77.88 1.93	76.86 76.86 0.00	76.86 74.37 1.44
HRT2	98.67 93.33 3.20	94.67 93.47 1.72	97.33 93.87 1.91	96.00 93.33 1.66	96.00 93.60 1.86	96.00 **94.27** 2.81	82.10 78.32 1.81	79.04 76.32 1.89	80.35 79.66 2.15	78.17 74.68 3.73
HRT3	88.00 84.40 2.60	88.00 **85.20** 2.31	88.00 83.87 1.83	86.67 84.13 1.33	89.33 84.40 2.27	89.33 84.80 2.45	77.30 74.24 1.46	74.68 74.24 0.61	74.68 71.40 1.59	75.11 74.10 1.76
HRS1	71.43 64.51 4.61	73.63 **66.59** 3.28	68.13 64.73 1.90	71.43 65.05 3.39	68.13 65.38 2.65	73.63 65.38 4.22	71.12 66.45 2.78	61.12 58.52 4.49	71.12 69.39 2.05	63.33 59.63 3.90
HRS2	67.03 61.98 3.78	64.84 61.98 2.15	70.33 62.31 3.81	67.03 62.31 2.98	68.13 61.65 3.61	68.13 **63.63** 3.38	63.33 57.73 3.48	62.23 59.73 4.29	61.12 56.50 2.55	64.45 61.12 4.84
HRS3	65.93 61.21 3.11	68.13 62.31 2.69	68.13 62.42 2.73	63.74 61.43 2.95	67.03 **63.52** 3.18	62.64 61.65 0.96	71.11 64.06 3.04	72.23 64.72 5.00	68.89 63.73 2.53	63.33 58.89 4.44
CAN1	98.85 97.70 0.72	98.85 97.70 0.54	99.43 **97.82** 0.85	98.85 97.76 0.79	98.28 97.70 0.77	98.85 **97.82** 0.80	97.71 96.21 1.01	97.71 95.61 1.42	97.13 94.34 1.24	97.13 95.69 0.94
CAN2	96.55 94.89 0.69	95.40 94.60 0.48	95.98 94.89 0.92	95.98 **94.94** 0.59	95.98 94.77 0.74	95.40 94.83 0.47	98.28 95.32 2.18	98.28 95.55 1.23	97.13 91.70 2.16	97.71 95.17 1.19
CAN3	97.13 96.32 0.78	97.13 96.09 0.71	97.13 96.38 0.61	97.13 **96.61** 0.57	97.13 96.32 0.40	97.13 96.03 0.69	97.71 95.61 1.36	96.56 95.10 0.83	98.86 94.72 1.70	97.71 95.58 1.43

Continue

Continued Table 4.6

Problem	**ES1**	**ES2**	**ES3**	**ES4**	**ES5**	**ES6**	G^{3}P D	G^{3}P F	G^{3}P A	G^{3}P P
	78.13	75.00	77.60	76.04	75.52	76.56	73.3	78.02	77.49	76.97
DBT1	72.50	72.71	**73.91**	73.13	72.08	71.98	68.3	73.53	75.46	73.18
	2.76	2.04	2.24	1.98	1.94	2.44	3.24	3.40	1.26	2.56
	75.52	75.00	73.96	73.96	75.00	74.48	74.35	76.44	76.97	76.97
DBT2	71.25	71.46	71.46	71.88	**72.29**	72.08	68.7	75.22	74.59	72.92
	2.44	1.80	1.32	1.15	2.19	1.63	3.48	1.22	1.15	2.65
	77.60	78.65	78.65	78.65	77.08	77.60	80.11	78.01	75.92	75.92
DBT3	75.16	**75.99**	75.36	75.16	75.16	75.47	71.21	75.75	71.24	71.79
	2.53	2.72	1.32	2.13	1.04	1.91	5.11	1.64	1.84	2.16
	89.66	88.65	90.92	89.66	88.27	87.77	77.68	88.03	68.23	87.27
GEN1	85.31	85.41	**86.15**	85.85	85.36	85.49	66.97	62.88	65.26	67.50
	4.37	1.74	2.78	2.34	1.92	1.74	6.70	14.99	4.19	14.93
	90.04	89.79	93.06	91.55	90.29	91.30	70.37	85.63	68.73	79.45
GEN2	85.98	86.61	87.69	86.80	**88.25**	87.93	62.97	62.73	58.52	70.88
	3.95	2.77	2.25	3.33	1.58	3.52	4.71	11.44	4.08	12.12
	88.27	91.43	86.13	88.78	88.40	87.14	73.52	88.03	67.47	75.11
GEN3	84.59	**86.02**	84.82	84.83	85.11	84.97	67.79	62.96	62.17	74.10
	3.00	3.64	1.14	3.00	2.66	1.43	5.06	11.81	7.37	1.76
	97.72	97.89	97.67	98.00	98.17	98.44	97.11	94.72	94.56	94.45
TRD1	97.15	97.19	97.16	97.18	**97.52**	97.19	95.04	93.92	94.50	93.74
	0.47	0.28	0.49	0.55	0.55	0.66	0.81	0.57	0.80	0.69
	98.28	98.28	98.28	98.50	98.78	98.33	97.34	94.56	94.44	94.56
TRD2	97.33	97.24	97.43	97.27	**97.57**	97.54	94.27	94.05	93.92	94.05
	1.22	1.04	0.72	1.18	0.76	0.64	1.25	0.52	0.60	0.60
	98.06	98.22	97.94	98.28	98.33	98.39	98.06	94.56	94.61	95.78
TRD3	96.95	97.06	97.21	97.20	97.04	**97.64**	94.55	93.97	94.28	94.51
	0.92	0.89	0.58	0.82	1.34	0.58	1.37	0.82	0.50	0.84

Table 4.7: Paired t-test results between **ES1** and five other experimental settings.

Problem	**ES1** vs. **ES2**	**ES1** vs. **ES3**	**ES1** vs. **ES4**	**ES1** vs. **ES5**	**ES1** vs. **ES6**
HRT1	0.3004	0.1460	0.4576	0.2754	0.3392
HRT2	0.4576	0.3526	0.5000	0.3925	0.2358
HRT3	0.2125	0.3097	0.3902	0.5000	0.3817
HRS1	0.1759	0.4325	0.3819	0.2925	0.3429
HRS2	0.5000	0.4233	0.4254	0.4270	0.1762
HRS3	0.2007	0.2132	0.4427	0.0602**	0.3643
CAN1	0.5000	0.3925	0.4236	0.5000	0.3820
CAN2	0.1494	0.5000	0.4236	0.3849	0.3988
CAN3	0.2955	0.4361	0.1815	0.5000	0.2201
DBT1	0.4267	0.0621**	0.2608	0.3021	0.3021
DBT2	0.4076	0.4064	0.1606	0.1685	0.1966
DBT3	0.2907	0.3908	0.5000	0.5000	0.3888
GEN1	0.4772	0.2882	0.3807	0.4884	0.4549
GEN2	0.3073	0.1218	0.3299	0.0567**	0.1645
GEN3	0.1757	0.4143	0.4311	0.3359	0.3668
TRD1	0.4226	0.4794	0.4426	0.0552**	0.4328
TRD2	0.4445	0.4198	0.4620	0.3298	0.3254
TRD3	0.4075	0.2357	0.2526	0.4344	0.0258*

* Significance level $\alpha = 0.05$

** Significance level $\alpha = 0.10$.

4.5.2.4 Comparing classification accuracy between LAGEP settings

The relation between the number of populations and classification accuracy is not clear. **ES5** achieves best average accuracy in five problems. **ES2**, **ES3** and **ES6** achieve best average accuracy in four problems. From TABLE 4.6, it is not certain that using more populations can obtain higher classification accuracy. Considering the training performance shown in TABLE 4.9, both **ES3** and **ES5** have the best average score values in the five problems. **ES2** has the best average score value in TRD3 only. Based on these observations, the number of populations might be irrelevant to obtaining good

Table 4.8: The average training time in seconds of six experimental settings and 18 problems.

Problem	**ES1**	**ES2**	**ES3**	**ES4**	**ES5**	**ES6**
HRT1	299.12	179.61	169.50	183.20	180.49	164.36
HRT2	334.87	199.01	180.15	178.64	173.70	161.39
HRT3	299.17	200.42	179.19	166.39	170.32	168.22
HRS1	452.25	302.19	305.34	309.23	276.03	286.49
HRS2	429.87	281.04	308.62	290.77	258.99	265.44
HRS3	420.87	325.74	283.43	297.34	266.15	287.89
CAN1	313.05	238.19	235.37	226.36	239.20	239.61
CAN2	349.83	247.53	245.30	244.63	228.43	233.43
CAN3	314.57	206.97	215.00	230.19	222.49	225.88
DBT1	421.83	298.17	294.32	296.38	260.71	248.41
DBT2	344.72	270.59	265.95	282.44	277.93	251.84
DBT3	391.46	274.79	264.93	259.56	249.48	252.10
GEN1	1446.09	1232.41	1285.21	1190.22	1115.65	1157.48
GEN2	1615.26	1287.77	1262.48	1252.35	1118.27	1095.65
GEN3	1585.13	1212.39	1277.77	1157.58	1162.54	1110.59
TRD1	4410.35	2680.82	2789.47	2497.28	2612.90	2522.12
TRD2	5243.94	2774.68	2702.87	2738.57	2725.54	2649.07
TRD3	4451.99	2682.36	2613.43	2495.75	2670.19	2628.05

discriminant functions.

4.5.2.5 The improvement of score values

The performance of the second layer is supposed to be better than the first layer because the population of the second layer uses transformed training instances. We use **ES2** to demonstrate this performance improvement. TABLE 4.10 illustrates the 18 problems and average score values of the three populations of **ES2**, where L_1P_1, L_1P_2, and L_2P_1 implies the first and the second population of the first layer and the population of the second layer, respectively. Both score values obtained by populations of the first layer are lower than the score value of the population of the second layer. TABLE 4.10 also shows the paired t-test results between them. It is obvious that L_2P_1 is significantly better than the two populations of the first layer. This result confirms that the training performance, the accuracy of classifying the training set, and the validation set, can be improved by the proposed layer architecture.

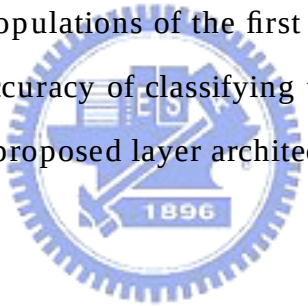


Table 4.9: Comparison of training performance between **ES1** and LAGEP settings.

For each problem, **ES1** column shows the average score value. Each cell containing a LAGEP setting shows the average score of the population of the second layer and paired t-tests result between it and ***E***S1.

Problem	**ES1**	**ES2**	**ES3**	**ES4**	**ES5**	**ES6**
HRT1	1.7898	1.8135 0.0023*	1.8280 0.0015*	1.8257 0.0006*	1.8191 0.0300*	1.8105 0.0275*
HRT2	1.6875	1.7454 0.0001*	1.7438 0.0005*	1.7500 0.0000*	1.7155 0.0161*	1.7158 0.0030*
HRT3	1.7477	1.7697 0.0438*	1.7944 0.0004*	1.7724 0.0183*	1.7727 0.0211*	1.7734 0.0146*
HRS1	1.6762	1.7071 0.0011*	1.7247 0.0001*	1.7255 0.0001*	1.7108 0.0086*	1.7093 0.0004*
HRS2	1.6868	1.7366 0.0007*	1.7385 0.0011*	1.7496 0.0001*	1.7372 0.0001*	1.7225 0.0028*
HRS3	1.6908	1.7275 0.0095*	1.7432 0.0004*	1.7399 0.0008*	1.7357 0.0005*	1.7443 0.0007*
CAN1	1.9537	1.9596 0.0048*	1.9627 0.0017*	1.9577 0.0324*	1.9610 0.0022*	1.9626 0.0001*
CAN2	1.9707	1.9741 0.0127*	1.9743 0.0017*	1.9734 0.0532**	1.9736 0.0005*	1.9749 0.0085*
CAN3	1.9560	1.9579 0.1154**	1.9610 0.0038*	1.9639 0.0001*	1.9649 0.0001*	1.9601 0.0169*

Continue

Continued Table 4.9

Problem	**ES1**	**ES2**	**ES3**	**ES4**	**ES5**	**ES6**
DBT1	1.5901	1.6061 0.0446*	1.6301 0.0002*	1.6206 0.0091*	1.6090 0.0081*	1.6147 0.0071*
DBT2	1.5866	1.6055 0.0763**	1.6206 0.0018*	1.6163 0.0048*	1.6096 0.0036*	1.6174 0.0043*
DBT3	1.5645	1.5956 0.0060*	1.5885 0.0109*	1.5957 0.0043*	1.5980 0.0012*	1.5820 0.0556**
GEN1	1.7915	1.8307 0.0778**	1.8262 0.0761**	1.8376 0.0299*	1.8435 0.0100*	1.8471 0.0310*
GEN2	1.7732	1.8161 0.0089*	1.8392 0.0044*	1.8090 0.0959**	1.8432 0.0018*	1.8338 0.0291*
GEN3	1.7786	1.8285 0.0516**	1.8391 0.0409*	1.8438 0.0176*	1.8406 0.0354*	1.8367 0.0401*
TRD1	1.9779	1.9773 0.3867	1.9769 0.3175	1.9786 0.3699	1.9804 0.2035	1.9801 0.1924
TRD2	1.9719	1.9725 0.4627	1.9746 0.2188	1.9737 0.3581	1.9769 0.0642**	1.9743 0.2621
TRD3	1.9779	1.9773 0.3867	1.9741 0.0909**	1.9726 0.1151	1.9739 0.1328	1.9752 0.1676

* Significance level $\alpha = 0.05$

** Significance level $\alpha = 0.10$

Table 4.10: The average score value of populations of **ES2** and the paired t-test results. L_iP_j indicates the jth population of the ith layer.

Problem	**ES2** L_1P_1	**ES2** L_1P_2	**ES2** L_2P_1	Paired t-test L_1P_1 vs. L_2P_1	Paired t-test L_1P_2 vs. L_2P_1
HRT1	1.7576	1.7720	1.8135	0.0000*	0.0000*
HRT2	1.6882	1.6737	1.7454	0.0000*	0.0000*
HRT3	1.7329	1.7155	1.7697	0.0000*	0.0000*
HRS1	1.6577	1.6423	1.7071	0.0000*	0.0000*
HRS2	1.6795	1.6885	1.7366	0.0000*	0.0000*
HRS3	1.6700	1.6723	1.7275	0.0000*	0.0000*
CAN1	1.9519	1.9499	1.9596	0.0023*	0.0000*
CAN2	1.9646	1.9664	1.9741	0.0005*	0.0001*
CAN3	1.9511	1.9519	1.9579	0.0000*	0.0004*
DBT1	1.5733	1.5754	1.6061	0.0004*	0.0001*
DBT2	1.5660	1.5728	1.6055	0.0000*	0.0000*
DBT3	1.5505	1.5434	1.5956	0.0000*	0.0000*
GEN1	1.7897	1.7601	1.8307	0.0001*	0.0000*
GEN2	1.7770	1.7585	1.8161	0.0077*	0.0003*
GEN3	1.7824	1.7372	1.8285	0.0119*	0.0001*
TRD1	1.9658	1.9694	1.9773	0.0000*	0.0028*
TRD2	1.9582	1.9660	1.9725	0.0005*	0.0034*
TRD3	1.9658	1.9694	1.9773	0.0000*	0.0028*

* Significance level $\alpha = 0.05$

4.6 LAGEP-FS Experiments

This section discusses the LAGEP-FS experiments and analyzes its performance. LAGEP-FS focuses on the performances of feature selection and feature generation. Therefore, rather than using all the same problems used in the LAGEP experiments, LAGEP-FS experiments use two-class classification only. They are *cancer*, *diabetes*, and *heart*. The conflict resolution method, ZM, does not apply in LAGEP-FS experiments.

The target class is required to be determined before starts training the LAGEP-FS.

For each classification problem, we use the major class of it, i.e., the class with the most training instances. The major classes of problems CAN, DBT, and HRT are C_1,C_2, and C_1, respectively.

4.6.1 Experimental Settings

Since the number of layers and the population size are determined empirically, we design ten different configurations. Experimental settings and common parameters are shown in TABLE 4.11. We perform every dataset with each experiment setting ten times for each problem to evaluate its average performance. A large population contains more individuals and usually has better performance. To eliminate the influence of population size, we decrease the population sizes in 2-layer and 3-layer experiments so that the number of total individuals of all populations approximate to 5000.

4.6.2 Analysis and Discussion

Classification accuracies of LAGEP-FS regarding CAN, DBT, and HRT problems are shown in TABLE 4.12, TABLE 4.13 and TABLE 4.14, respectively. From those tables, one can see that *M1* is found to be better than *M2* in either feature selection or accuracy in most cases. In total 90 average accuracies, *M1* outperforms *M2* 66 times. The best accuracies are derived from the *M1* method.

When *M1* is used, an individual may decrease its fitness value because it has too many features and loses its chance to surpass other individuals in the tournament selection. *M1* attempts to discover individuals that have used fewer features without loss of fitness. On the other side, *M2* aims to reduce the chance of poor features being selected. Weights of features might be converged provided enough generations for *M2*.

In the following, we analyze the experimental results by grouping relevant exper-

Table 4.11: Ten experimental settings of LAGEP-FS.

Setting name	Γ	$l(i)$	Population size for each	Total number of individuals
ES1	1	1	5000	5000
ES2	2	1, 1	2500	5000
ES3		2, 1	1650	4950
ES4		3, 1	1250	5000
ES5		4, 1	1000	5000
ES6	3	1, 1, 1	1650	4950
ES7		2, 2, 1	1000	5000
ES8		3, 3, 1	700	4900
ES9		6, 3, 1	100	5000
ES10		3, 6, 1	100	5000

Common settings

Max generation, G	250	Tournament size	5
Crossover probability, P_c	0.95	Mutation probability, P_m	0.05
IL	255		

imental settings.

4.6.2.1 Analyzing ES1

ES1 is the only experimental setting that uses a single population containing 5000 individuals. It obtains the best accuracies in both the CAN2 and the HRT1. **ES1** is also used to demonstrate the performance of traditional single population GP with the two feature selection methods. *M1* does not always outperform *M2* with respect to classification accuracy, and vice versa.

Table 4.12: Accuracy and average number of used features of ten experimental settings in CAN problems.

Best accuracy and fewest average number of used features are marked in bold face; $>$, $=$, and $<$ indicate the relation between accuracies of using *M1* and *M2*.

Setting name	Feature selection	CAN1		CAN2		CAN3	
ES1	*M1*	$0.9747(6.9)$		$\mathbf{0.9425}(6.7)$		$0.9540(6.5)$	
	M2	$0.9701(7.6)$	$>$	$0.9391(7.0)$	$>$	$0.9546(7.1)$	$<$
ES2	*M1*	$0.9695(\mathbf{4.3})$		$0.9362(4.3)$		$0.9534(4.6)$	
	M2	$0.9598(8.4)$	$>$	$0.9356(8.2)$	$>$	$0.9552(7.7)$	$<$
ES3	*M1*	$0.9741(5.2)$		$0.9351(4.4)$		$0.9569(5.0)$	
	M2	$0.9718(8.0)$	$>$	$0.9333(8.5)$	$>$	$0.9471(8.1)$	$>$
ES4	*M1*	$0.9701(5.2)$		$0.9305(4.6)$		$0.9592(5.6)$	
	M2	$0.9557(8.6)$	$>$	$0.9414(7.3)$	$>$	$0.9529(8.5)$	$>$
ES5	*M1*	$\mathbf{0.9776}(5.2)$		$0.9339(4.5)$		$\mathbf{0.9598}(4.6)$	
	M2	$0.9684(8.9)$	$>$	$0.9379(8.2)$	$<$	$0.9529(8.0)$	$>$
ES6	*M1*	$0.9747(4.8)$		$0.9356(\mathbf{4.0})$		$\mathbf{0.9598}(5.2)$	
	M2	$0.9713(7.5)$	$>$	$0.9017(8.5)$	$>$	$0.9218(7.9)$	$>$
ES7	*M1*	$0.9741(4.8)$		$0.9356(4.3)$		$0.9563(\mathbf{4.4})$	
	M2	$0.9701(8.6)$	$>$	$0.9345(8.1)$	$>$	$0.9460(8.5)$	$>$
ES8	*M1*	$0.9730(5.0)$		$0.9333(4.7)$		$0.9557(4.7)$	
	M2	$0.9621(8.3)$	$>$	$0.9402(7.9)$	$<$	$0.9517(8.2)$	$>$
ES9	*M1*	$0.9672(4.7)$		$0.9402(5.1)$		$0.9575(5.1)$	
	M2	$0.9741(8.7)$	$<$	$0.9397(7.9)$	$>$	$0.9448(8.1)$	$>$
ES10	*M1*	$0.9713(5.1)$		$0.9408(4.8)$		$0.9592(4.8)$	
	M2	$0.9339(8.6)$	$>$	$0.9391(8.2)$	$>$	$0.9517(7.8)$	$>$

Table 4.13: Accuracy and average number of used features of ten experimental settings in DBT problems.

Best accuracy and fewest average number of used features are marked in bold face; $>$, $=$, and $<$ indicate the relation between accuracies using *M1* and *M2*.

Setting name	Feature selection	DBT1		DBT2		DBT3	
ES1	*M1*	$0.6885(6.8)$		$0.6969(6.5)$		$0.7052(\mathbf{5.7})$	
	M2	$0.6833(7.2)$	$>$	$0.7063(7.1)$	$<$	$0.7078(7.3)$	$<$
ES2	*M1*	$\mathbf{0.7276}(6.1)$		$0.7000(\mathbf{5.6})$		$0.7276(6.1)$	
	M2	$0.6906(7.8)$	$>$	$0.6813(8.0)$	$>$	$0.6906(7.8)$	$>$
ES3	*M1*	$0.7146(6.8)$		$0.6948(6.8)$		$\mathbf{0.7354}(6.4)$	
	M2	$0.7078(7.9)$	$>$	$0.6917(8.0)$	$>$	$0.7307(8.0)$	$>$
ES4	*M1*	$0.6938(7.3)$		$0.7094(6.1)$		$0.7313(6.5)$	
	M2	$0.6854(8.0)$	$>$	$0.6766(8.0)$	$>$	$0.7115(8.0)$	$>$
ES5	*M1*	$0.7026(7.6)$		$0.7005(6.5)$		$0.7307(7.2)$	
	M2	$0.6818(8.0)$	$>$	$0.6693(8.0)$	$>$	$0.7068(8.0)$	$>$
ES6	*M1*	$0.6854(\mathbf{5.9})$		$0.7078(6.7)$		$0.7292(5.9)$	
	M2	$0.6885(8.0)$	$<$	$0.6536(8.0)$	$>$	$0.6828(8.0)$	$>$
ES7	*M1*	$0.6729(7.0)$		$0.6969(6.8)$		$0.7240(6.9)$	
	M2	$0.6792(8.0)$	$<$	$0.6677(8.0)$	$>$	$0.6885(8.0)$	$>$
ES8	*M1*	$0.6984(7.4)$		$0.6958(7.0)$		$0.7286(6.4)$	
	M2	$0.6807(8.0)$	$>$	$0.6724(8.0)$	$>$	$0.6813(8.0)$	$>$
ES9	*M1*	$0.6958(6.9)$		$\mathbf{0.7104}(7.0)$		$0.7266(5.8)$	
	M2	$0.6750(8.0)$	$>$	$0.6734(8.0)$	$>$	$0.6974(7.7)$	$>$
ES10	*M1*	$0.6984(6.9)$		$0.6938(6.4)$		$0.7057(6.0)$	
	M2	$0.6880(8.0)$	$>$	$0.6479(8.0)$	$>$	$0.6906(8.0)$	$>$

Table 4.14: Accuracy and average number of used features of ten experimental settings in HRT problems.

Best accuracy and fewest average number of used features are marked in bold face; $>$, $=$, and $<$ indicate the relation between accuracies using *M1* and *M2*.

Setting name	Feature selection	HRT1		HRT2		HRT3	
ES1	*M1*	**0.7760(11.0)**		0.9120(17.4)		0.8333(**11.3**)	
	M2	0.7520(14.1)	$>$	0.9133(**12.6**)	$<$	0.8293(13.3)	$>$
ES2	*M1*	0.7387(13.6)		0.8973(16.4)		0.8387(14.4)	
	M2	0.7680(18.1)	$<$	0.8973(19.2)	$=$	0.8333(17.0)	$>$
ES3	*M1*	0.7693(17.4)		0.9067(20.2)		0.8173(16.5)	
	M2	0.7560(24.1)	$>$	0.9000(23.6)	$>$	0.8240(20.0)	$<$
ES4	*M1*	0.7533(18.9)		0.9040(18.2)		0.8120(16.8)	
	M2	0.7560(22.4)	$<$	0.8947(24.7)	$>$	0.8133(21.9)	$<$
ES5	*M1*	0.7600(19.7)		0.8947(21.6)		0.8320(17.5)	
	M2	0.7573(22.0)	$>$	0.8893(25.3)	$>$	0.8187(22.0)	$>$
ES6	*M1*	0.7547(15.9)		0.8987(16.9)		0.8320(15.2)	
	M2	0.7520(18.6)	$>$	0.8960(20.8)	$>$	0.8200(18.1)	$>$
ES7	*M1*	0.7613(17.6)		0.8947(18.7)		0.8240(16.9)	
	M2	0.7467(21.8)	$>$	0.8987(21.9)	$<$	0.8013(21.6)	$>$
ES8	*M1*	0.7467(20.4)		0.8560(22.5)		0.8227(18.2)	
	M2	0.7480(23.9)	$<$	0.9027(25.8)	$<$	0.8000(23.9)	$>$
ES9	*M1*	0.7613(16.2)		0.9080(19.9)		0.8440(13.9)	
	M2	0.7640(24.1)	$<$	0.9080(23.1)	$=$	0.8240(22.4)	$>$
ES10	*M1*	0.7440(17.6)		**0.9200**(16.3)		**0.8453**(12.7)	
	M2	0.7493(21.4)	$<$	0.9133(19.8)	$>$	0.8240(17.9)	$>$

4.6.2.2 Analyzing two-layer LAGEP-FS settings

These settings both use two layers. **ES2** and **ES3** achieve the best accuracies in DBT1 and DBT3, respectively. **ES5** performs excellently in CAN1 and CAN3. It is found that the improvement gained by increasing the number of populations in first layer is not significant. Using a number of smaller populations compensates its lack of population diversity compared to larger population.

4.6.2.3 Analyzing three-layer LAGEP-FS settings

These settings use three layers. **ES6** achieves the best accuracy in CAN3. **ES9** achieves the best accuracy in DBT2. The best accuracies in both HRT2 and HRT3 are obtained by **ES10**. Using more layers does not always infer better performance.

4.6.2.4 Analyzing LAGEP-FS settings that have the same number of populations

ES3 and **ES6** have the same number of populations. **ES5** and **ES7** also have the same number of populations. **ES3** outperforms **ES6** in four problems with *M1* and in all problems with *M2*. This phenomenon raises a hypothesis that using more populations in first layer would be better than separating populations into more layers. Using *M1*, **ES5** outperforms **ES7** in six problems. Using *M2*, **ES5** performs better in seven problems with *M2*. It seems that arranging populations within fewer layers might be a better way.

4.6.2.5 Analyzing ES9 and ES10

The last group consists of **ES9** and **ES10**. **ES9**'s first layer contains six populations, which is the number of populations that **ES10** uses in second layer. **ES10** outperforms **ES9** in six problems with *M1* and three problems with *M2*. Using more populations in

first layer means that the training data for the second layer have more features. Consider the high-dimensional *heart* problems and the eight-dimensional *diabetes* problems. For *heart*, the three populations in the second layer of **ES9** evolve with data having 41 features, including 35 original features and six new ones produced by LAGEP-FS. However, **ES10**'s second layer uses six populations to evolve with data having only 38 features. This explains why **ES9** performs worse than **ES10** in *heart*. However, a different situation occurs in the *diabetes* problems. The three populations of the second layer of **ES9** are sufficient to obtain good results with the given 14 features where six of them are generated by feature extraction. Once the proportion of new features compared to the original ones becomes high enough, using more populations in first layer might be a good configuration.

4.6.2.6 The effectiveness of new features

This section discusses the effectiveness of the new features. The evolutionary process tends to generate a high score discriminant function. During the evolution, if a constructed new feature impacts on the score, it is supposed to be kept in the output function. Therefore, the effectiveness of new features can be defined as the following:

$$\frac{\text{the number of new features in the final function}}{\text{the number of features in the final function}} \times 100\% \tag{4.1}$$

We evaluate the effectiveness of new features of all generated discriminant functions by the equation. Table 4.15 shows the average effectiveness corresponding to feature selection methods, experimental settings, and the nine problems.

In TABLE 4.15, *M1* seems to use more new features than *M2* does. Above discusses conclude that *M1* has better classification accuracy than *M2* does. We can conclude that constructed new features are effective for obtaining successive discriminant functions.

In this chapter we conduct many experiments to illustrate performance of LAGEP

Table 4.15: Comparison of new feature effectiveness. A higher value stands for a greater effectiveness.

	M1			*M2*		
Problem	CAN1	CAN2	CAN3	CAN1	CAN2	CAN3
ES2	100.00%	100.00%	100.00%	42.34%	38.54%	36.05%
ES3	100.00%	90.91%	100.00%	84.13%	63.93%	46.77%
ES4	100.00%	100.00%	100.00%	37.93%	42.42%	40.74%
ES5	100.00%	100.00%	100.00%	62.00%	53.85%	39.22%
ES6	100.00%	100.00%	100.00%	46.77%	64.52%	68.57%
ES7	73.68%	100.00%	100.00%	68.42%	77.78%	77.78%
ES8	100.00%	100.00%	91.67%	60.00%	61.90%	100.00%
ES9	100.00%	100.00%	100.00%	92.86%	100.00%	100.00%
ES10	100.00%	100.00%	100.00%	70.00%	100.00%	72.73%
Problem	DBT1	DBT2	DBT3	DBT1	DBT2	DBT3
ES2	32.00%	59.09%	63.16%	21.85%	20.79%	22.61%
ES3	64.44%	64.29%	68.18%	40.00%	42.11%	38.01%
ES4	57.69%	50.00%	60.00%	27.49%	30.96%	35.59%
ES5	77.42%	71.43%	62.86%	60.23%	63.27%	50.55%
ES6	56.14%	72.50%	74.36%	61.81%	54.20%	58.04%
ES7	90.32%	84.62%	88.89%	57.21%	54.81%	55.61%
ES8	63.64%	84.62%	100.00%	68.24%	61.95%	64.22%
ES9	100.00%	82.14%	67.86%	62.81%	61.84%	85.93%
ES10	76.19%	100.00%	100.00%	72.22%	66.67%	66.23%
Problem	HRT1	HRT2	HRT3	HRT1	HRT2	HRT3
ES2	26.56%	27.59%	21.36%	38.71%	23.21%	21.05%
ES3	50.00%	45.45%	85.71%	40.74%	33.33%	68.75%
ES4	27.27%	31.43%	47.06%	39.47%	33.85%	47.30%
ES5	54.90%	39.81%	49.30%	47.67%	62.60%	60.49%
ES6	57.14%	66.67%	54.84%	58.57%	52.59%	51.95%
ES7	58.62%	83.33%	59.09%	56.10%	71.43%	52.94%
ES8	79.17%	45.16%	100.00%	69.57%	37.50%	67.69%
ES9	73.68%	80.00%	62.50%	72.00%	88.37%	73.17%
ES10	48.00%	88.89%	82.76%	72.50%	45.65%	66.67%

and LAGEP-FS with regard to different configurations. All hypotheses are confirmed successfully. LAGEP is capable of combining short discriminant functions into a longer one and which has high classification accuracy on test set. The layer architecture makes the populations in the successive layer have higher score value. Moreover, the layer architecture lets completing feature selection and feature generation simultaneously be possible. Two feature selection methods are proposed. Performance of them are shown and compared in above sections.

Chapter 5

Conclusion and Future Work

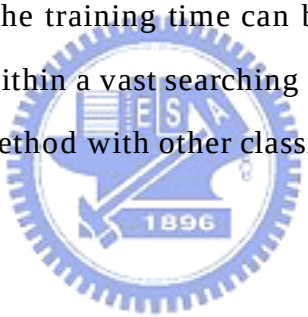
In this paper, we propose two MGP methods, LAGEP and LAGEP-FS, to complete classification, feature selection, and feature generation. Both methods arrange a number of populations into a layer. Every layer evolves its populations to generate a set of discriminant functions. These functions transform the training set into a new training set, which is used for the successive layer. The evolutionary process of every population is efficient because it evolves within short individuals. We also propose a method called *AMPT* to prevent falling into a local optimum.

An experiment comparing a single population with *AMPT* and without *AMPT* shows that *AMPT* is significantly effective. Experimental results show that LAGEP is capable of generating a high accuracy discriminant function efficiently. We found that LAGEP is affected by the overfitting problem slightly; thus we show the average score values of single population GP and LAGEP settings to illustrate that LAGEP settings have significantly higher training performance. We also use one of the LAGEP settings to show that training performance can be improved by layered architecture. Although the classification accuracy is similar, the elapsed training time of LAGEP is much less than single population GP. Experimental results also show that the number of populations could be irrelevant to either training performance or classification accuracy of the test set.

By means of a number of experiments, we show that LAGEP-FS is capable of completing feature selection and feature generation simultaneously. Moreover, the classi-

fication accuracy of LAGEP-FS is comparable to traditional single population genetic programming. We also propose two feature selection methods. Through a series of experiments, the proposed method *M1* shows better performance than *M2*.

In this paper, LAGEP uses only two layers. We intend to investigate the performance of LAGEP with more layers in future. Furthermore, we are interested in an application of LAGEP with different population configurations that is called the *heterogeneous* MGP model. Our further research will focus on discovering the relation between the number of original and the number of new features. Implementation of LAGEP and LAGEP-FS can be achieved by either a parallel distribution computing environment or a serial computing environment. Since the populations are independent of each other, we can perform the evolutionary process of each population on a number of different computers at different times and combine the results to build a layer afterward. In the case that the training time can be greatly reduced, it is possible to find high quality solutions within a vast searching space. Furthermore, it is interesting to combine the LAGEP-FS method with other classification algorithms such as SVM or NN.



Bibliography

[1] A. Ahmad and L. Dey, "A feature selection technique for classificatory analysis," *Pattern Recognition Letters*, vol. 26, pp. 43–56, 2005.

[2] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack, and A. J. Levine, "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays," in *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, pp. 6745–6750, 1999.

[3] David Andre and John R. Koza, "Parallel genetic programming: a scalable implementation using the transputer network architecture," pp. 317–337, 1996.

[4] W. Banzhaf, P. Nordin, R. E.Keller, and F. D. Framcone, *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Application*, San Francisco, CA: Morgan Kaufmann, 1998.

[5] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, "Discovering comprehensible classification rules using genetic programming: a case study in a medical domain," in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-99)*, pp. 953–958, Orlando, FL, USA, 1999.

[6] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, vol. 5, pp. 17–26, February 2001.

[7] Markus Brameier, Frank Hoffmann, Peter Nordin, Wolfgang Banzhaf, and Frank Francone, "Parallel machine code genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference* (Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, eds.), vol. 2, p. 1228, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[8] Kuo-Hsiu Chen, Hong-Ling Chen, and Hahn-Ming Lee, "A multiclass neural network classifier with fuzzy teaching inputs," *Fuzzy Sets and Systems*, vol. 91, pp. 15–35, October 1997.

[9] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[11] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Controls, Signals, and Systems*, vol. 2, pp. 303–314, 1989.

[12] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, no. 3, pp. 131–156, 1997.

[13] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. "UCI repository of machine learning databases,". online source: http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[14] P. Domingos and M. Pazzani, "Beyond independence: Conditions for the optimality of the simple bayesian classifier," in *Proceedings of the 13th International Conference on Machine Learning*, pp. 105–112, 1996.

[15] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*, New York, USA: Wiley, 1973.

[16] F. E. B. F. E. B. Otero, M. M. S. Silva, A. A. Freitas, and J. C. Nievola, "Genetic programming for attribute construction in data mining," in *Proceedings of 6th European Conference on Genetic Programming*, pp. 384–393, Essex, UK, 2003.

[17] I. De Falco, A. Della Cioppa, and E. Tarantino, "Discovering interesting classification rules with genetic programming," *Applied Soft Computing*, vol. 23, pp. 1–13, May 2002.

[18] F. Fernández, M. Tomassini, and J. M. Sanchez, "Experimental study of isolated multipopulation genetic programming," in *IEEE International Conference on Industrial Electronics, Control and Instrumentation*, p. 2672 – 2677, Nagoya, Japan, 2000. IEEE Press, Piscataway, NJ.

[19] F. Fernández, M. Tomassini, and L. Vanneschi, "An empirical study of multipopulation genetic programming," *Genetic Programming and Evolvable Machines*, vol. 4, pp. 21–51, May 2003.

[20] David B. Fogel, *Evolutionary computation: toward a new philosophy of machine intelligence*, NJ: IEEE Press, 1995.

[21] A. A. Freitas, "A genetic programming framework for two data mining tasks: classification and generalized rule induction," in *Proceedings of 2nd annual conference on Genetic Programming*, pp. 96–101, Stanford University, CA, USA, July 1997. Morgan Kaufmann.

[22] M. Fuchs, "Large population are not always the best choice in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference GECCO'99* (W. Banzhaf, J. Daida, A. E. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, eds.), pp. 1033–1038, San Francisco, CA, 1999. Morgan Kaufmann.

[23] C. Gathercole and P. Ross, "Small populations over many generations can beat large populations over few generations in genetic programming," in *Genetic Programming 1997: Proceedings of the second annual conference* (J. R. Koza, K. Deb,

M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds.), pp. 111–118, San Francisco, CA, 1997. Morgan Kaufmann.

[24] J. Han and M. Kamber, *Data mining: concepts and techniques*, San Mateo, CA, USA: Morgan Kaufmann, 2001.

[25] Y. Hayashi, M. Sakata, and S. I. Gallant, "Multi-layer versus single-layer neural networks and a application to reading hand-stamped characters," in *Proceedings of International Conference on Neural Networks*, pp. 781–784, Paris, France, 1990.

[26] M. A. Hearst, B. Scholkopf, S. Dumais, E. Osuna, and J. Platt, "Trends and controversies - support vector machines," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 18–28, 1998.

[27] D. Heckerman and M. P. Wellman, "Bayesian networks," *Communications of the ACM*, vol. 38, no. 3, pp. 27–30, 1995.

[28] M. I. Heywood and A. N. Zincir-Heywood, "Dynamic page based crossover in linear genetic programming," *IEEE Transactions on System, Man, and Cybernetics - Part B: Cybernetics*, vol. 32, pp. 380–388, June 2002.

[29] G. Hong, L. B. Jack, and A. K. Nandi, "Feature generation using genetic programming with application to fault classification," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 35, no. 1, pp. 89–99, 2005.

[30] H. Morton I. Aleksander, *An Introduction to Neural Computing*, London, UK: Chapman and Hall, 1990.

[31] A. K. Jain and D. Zongker, "Feature selection: evaluation, application, and small sample performance," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153–158, 1997.

[32] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of 11th International Conference on Machine Learning*, pp. 121–129, New Brunswick, NJ, USA, 1994.

[33] I. T. Jolliffe, *Principal Component Analysis*, New York, USA: Springer, 1986.

[34] V. Kecman, *Learning and soft computing: support vector machines, neural networks, and fuzzy logic models*, Cambridge, MA: MIT Press, 2001.

[35] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 242–258, September 2000.

[36] J. Kittler, "Feature set search algorithms," in *Pattern Recognition and Signal Processing* (C. H. Chen, ed.), pp. 41–60, Netherlands, 1978. Sijthoff and Noordhoff.

[37] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, pp. 273–324, 1997.

[38] A. Konstam, "Group classification using a mix of genetic programming and genetic algorithms," in *Proceedings of the 1998 ACM Symposium of Applied computing*, pp. 308–312, Atlanta, Georgia, USA, February 27 - March 1 1998.

[39] M. Kotani, S. Ozawa, M. Nakai, and K. Akazawa, "Emergence of feature extraction function using genetic programming," in *Proceedings of Third International Conference on Knowledge-Based Intelligent Information Engineering System*, pp. 149–152, 1999.

[40] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA: MIT Press, 1992.

[41] J. R. Koza, *Genetic programming II*, Cambridge, MA: MIT Press, 1994.

[42] J. R. Koza, D. E. Goldberg, and D. B. Fogel, eds., *Genetic programming 1996*, Cambridge, MA: MIT Press, 1996.

[43] J. R. Koza, Forrest H. Bennett III, Forrest H. Bennett, David Andre, and Martin A. Keane, *Genetic programming III: Automatic programming and automatic circuit synthesis*, San Mateo, CA: Morgan Kaufmann, 1999.

[44] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, , and Guido Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[45] M. Kudo and J. Sklansky, "Comparison of algorithms that select features for pattern classifiers," *Pattern Recognition*, vol. 33, pp. 25–41, 2000.

[46] S. R. Kulkarni, G. Lugosi, and S. S. Venkatesh, "Learning pattern classification - a survey," *IEEE Transactions on Information Theory*, vol. 44, pp. 2178–2206, October 1998.

[47] A. Rogers L. Kallel, B. Naudts, ed., *Theoretical aspects of evolutionary computing*, Germany: Springer-Verlag, 2000.

[48] C. Lee and D. A. Landgrebe, "Decision boundary feature extraction for neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 75–83, 1997.

[49] H. M. Lee, "A neural network classifier with disjunctive fuzzy information," *Neural Networks*, vol. 11, pp. 1113–1125, August 1998.

[50] J. Y. Lin. "Lagep project,". online resource: http://www.cis.nctu.edu.tw/~gis91815/lagep/lagep.html, 2007.

[51] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proceedings of the 2001 Congress on Evolutionary Computation*, pp. 1070–1077, May 27-30 2001.

[52] J. Ma, J. Theiler, and S. Perkins, "Two realizations of a general feature extraction framework," *Pattern Recognition*, vol. 37, pp. 875–887, 2004.

[53] J. Mao and A. K. Jain, "Artificial neural networks for feature extraction and multivariate data projection," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 296–317, 1995.

[54] J. Mercer, "Functions of positive and negative type and their connection with the theory of integral equations," *Philosophical Transactions of the Royal Society, London*.

[55] Brad L. Miller and David E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, pp. 193–212, 1995.

[56] M. Mitchell, *An introduction to genetic algorithms*, Cambridge, MA, USA: MIT Press, 1996.

[57] T. M. Mitchell, *Machine learning*, New York, USA: McGraw-Hill, 1997.

[58] D. P. Muni, N. R. Pal, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 183–196, April 2004.

[59] D. P. Muni, N. R. Pal, and J. Das, "Genetic programming for simultaneous feature selection and classifier design," *IEEE Transactions on System, Man, and Cybernetics - Part B: Cybernetics*, vol. 36, no. 1, pp. 106–117, 2006.

[60] D. Nauck and R. Kruse, "A neuro-fuzzy method to learn fuzzy classification rules from data," *Fuzzy Sets and Systems*, vol. 89, no. 3, pp. 277–288, 1997.

[61] I. S. Oh, J. S. Lee, and B. R. Moon, "Hybrid genetic algorithms for feature selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1424–1437, 2004.

[62] Mouloud Oussaidene, Bastien Chopard, and Olivier V. Pictet, "Parallel genetic programming and its application to trading model induction," *Parallel Computing*, vol. 23, no. 8, pp. 1183–1198, 1997.

[63] C. H. Park and H. Park, "Nonlinear feature extraction based on centroids and kernel functions," *Pattern Recognition*, vol. 37, pp. 801–810, 2004.

[64] F. Pernkopf, "Bayesian network classifiers versus selective k-nn classifier," *Pattern Recognition*, vol. 38, pp. 1–10, 2005.

[65] L. Prechelt. "Proben1-a set of neural network benchmark problems and benchmarking rules,". Tech. Rep. 21/94, University at Karlsruhe, Karlsruhe, Germany, 1994.

[66] P. Pudil, J. Novovicova, and J. Kitter, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, pp. 1119–1125, 1994.

[67] Herbrich Ralf, *Learning Kernel Classifiers: Theory and Algorithms*, Cambridge, MA, USA: MIT Press, 2001.

[68] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn, and A. K. Jain, "Dimensionality reduction using genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 4, pp. 164–171, 2000.

[69] M. M. Rizki, M. A. Zmuda, and L. A. Tamburino, "Evolving pattern recognition systems," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 6, pp. 594–609, 2002.

[70] B. Schölkopf and A. J. Smola, *Learninig with Kernels*, Cambridge, MA, USA: MIT Press, 2002.

[71] J. Sherrah, R. E. Bogner, and A. Bouzerdoum, "Automatic selection of features for classification using genetic programming," in *Proceedings of Australian and New Zealand conference on Intelligent information systems*, pp. 284–287, Adelaide, SA, Australia, November 18-20 1996.

[72] W. Siedlecki and J. Sklansky, "A note on genetic algorithms for large-scale feature selection," *Pattern Recognition Letters*, vol. 10, pp. 335–347, 1989.

[73] P. K. Simpson, "Fuzzy min-max neural networks-part 1: Classification," *IEEE Transaction on Neural Networks*, vol. 3, pp. 776–786, October 1992.

[74] M. G. Smith and L. Bull, "Using genetic programming for feature creation with a genetic algorithm feature selector," in *Proceedings of 8th International Conference on Parallel Problem Solving from Nature*, pp. 1163–1171, Birmingham, UK, 2004.

[75] A. Tsakonas, "A comparison of classification accuracy of four genetic programming-evolved intelligent structures," *Information Sciences*, vol. 176, pp. 691–724, 2006.

[76] V. Vapnik, *The nature of statistical learning theory*, New York, NY, USA: Springer-Verlag, 1995.

[77] X. Wang and K. K. Paliwal, "Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition," *Pattern Recognition*, vol. 36, pp. 2429–2439, 2003.

[78] G. Wilson and M. I. Heywood, "Crossover context in page-based linear genetic programming," in *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 809–814. IEEE Press, May 12-15 2002.

[79] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transaction on Systems, Man, And Cybernetics-Part C: Applications and Reviews*, vol. 30, no. 4, pp. 451–462, 2000.

Vita

Jung Yi Lin was born in November, 1978 in Taitung, Taiwan. He received his B.S. in Applied Mathematics, and M.E. in Computer Science and Information Engineering from I-Shou University (ISU), Kaohsiung, Taiwan, in 2000 and 2002, respectively. In 2007, Jung Yi Lin received his Ph.D. degree in Computer Science from National Chiao Tung University (NCTU), HsinChu, Taiwan. Jung Yi Lin was a visiting Ph.D student in AI Lab, Department of Management Information System, University of Arizona, Tucson, Arizona, USA, from May 2006 to May 2007. His current research interests include evolutionary computation, machine learning, data mining, and artificial intelligence.

Jung Yi Lin is reachable by email: jylin@ieee.org

Publication List

SELECTED JOURNAL PAPERS

1. B. C Chien, **J. Y Lin**, T. P Hong, “Learning Discriminant Functions with Fuzzy Attributes for Classification Using Genetic Programming”, Expert Systems with Applications, Volume: 23, Issue: 1, July, 2002. (SCI)

2. B. C. Chien, **J. Y. Lin**, W. P. Yang, “Learning Effective Classifiers with Z-value Measure Based on Genetic Programming”, Pattern Recognition, Vol. 37, No. 10, 2004, pp. 1957-1972. (SCI)

3. L. Chuang, J. Hwang, B. C. Chien, **J. Y. Lin**, C. Chang, C. Yu, and F. Chang, “Predicting Fetal Birth Weight by Ultrasound with the use of Genetic Programming”, Ultrasound in Medicine & Biology, Vol. 29, Number 5, Supplement 1, (2003), S163-S163. (SCI)

4. B. C. Chien, **J. Y. Lin**, W. P. Yang, “A Classification Tree Based on Discriminant Functions”, Journal of Information Science and Engineering, Vol. 22, No. 3, 2006, pp. 573-594. (SCI)

5. **J. Y. Lin**, H. R. Ke, B. C. Chien, W. P. Yang, “Designing a Classifier by a Layered Multi-Population Genetic Programming Approach”, Pattern Recognition, Vol. 40, Issue 8, Aug. 2007, pp. 2211-2225 (SCI)

6. **J. Y. Lin**, H. R. Ke, B. C. Chien, W. P. Yang, “Classifier Design with Feature Selection and Feature Extraction Using Layered Genetic Programming”, Accepted by Expert Systems with Applications at 2007, will be appeared at 35(2), 2008 (SCI).

SELECTED CONFERENCE PAPERS

1. **J. Y. Lin**, B. C. Chien, Shih-Jing Lin, Yi-Chung Hsieh, Yung-Feng Hwang, “An Automatic Image Classification System Using Genetic Programming”, National Computer Symposium (NCS), Vol. B, Taipei, Taiwan, 2001, pp. 31-38.

2. **J. Y. Lin**, B. C. Chien, Tzung-Pei Hong, “A Function-Based Classifier Learning Scheme Using Genetic Programming”, Proceedings of 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Taipei, Taiwan, LNAI 2336, Springer-Verlag, pp. 92-103, 2002.

3. B. C. Chien, **J. y. Lin**, “A Classifier with the Function-based Decision Tree”, Sixth International Conference on Knowledge-based Intelligent Information Engineering Systems and Allied Technologies (KES), 2002.

4. L. Chuang, J. Y. Hwang, B. C. Chien, **J. Y. Lin**, C. H. Chang, C. H. Yu, F. M. Chang, “Predicting fetal birth weight by ultrasound with the use of genetic programming”, 10th Congress of the World Federation for Ultrasound in Medicine and Biology, hosted by the American Institute of Ultrasound in Medicine (AIUM), Palais de Congress, Montreal, Canada, June 1-4, 2003,

5. **J. Y. Lin**, H. R. Ke, B. C. Chien, and W. P. Yang, “Medical problem classification using layered genetic programming with feature generation,” Workshop on The Sciences of The Artificial (WSA), Dec 7-8, Hualien, Taiwan, 2005. pp. 182-198