# 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

# List of Figures

# List of Tables

# 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 $E$.

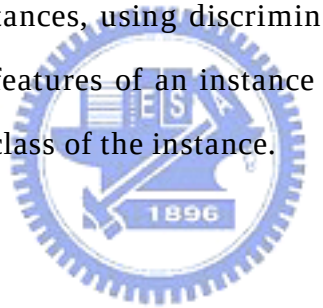
EC can be viewed as a field of machine learning because its individuals adapt to "fit" $E$, where *fit* means gaining high performance by the measure $P$.

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 $\pi$, $e$, 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.

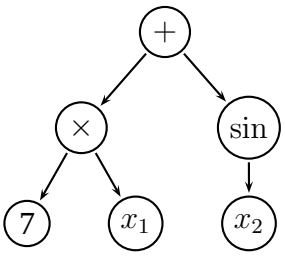



Figure 2.1: An individual representing $7x_1 + \sin(x_2)$

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.

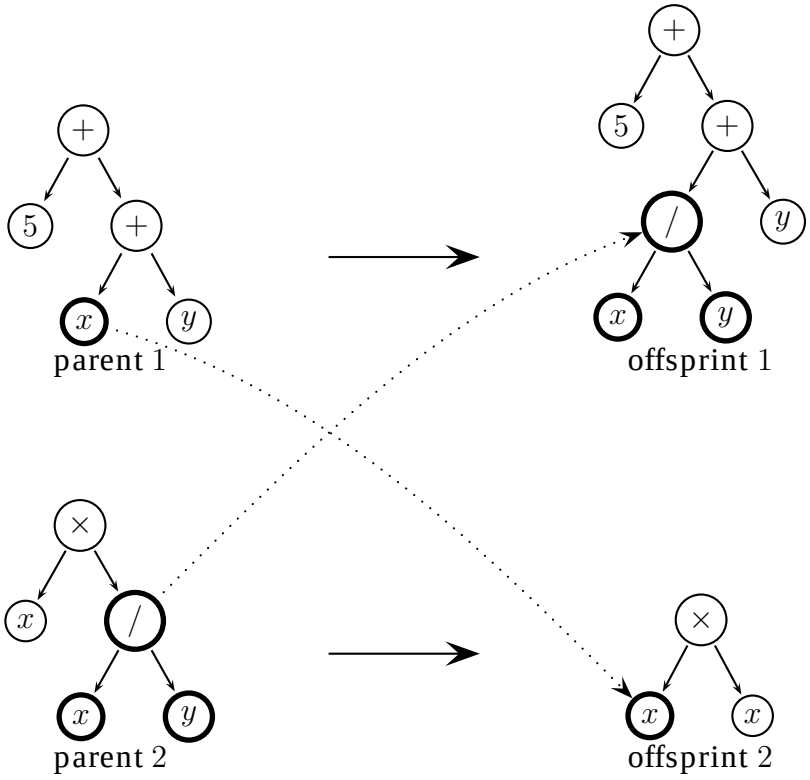



Figure 2.2: An example of the crossover operator. Parents $(5 + (x + y))$ and $(x \times (x/y))$ generate two offspring $(5 + ((x/y) + y))$ and $(x \times x)$

- 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.

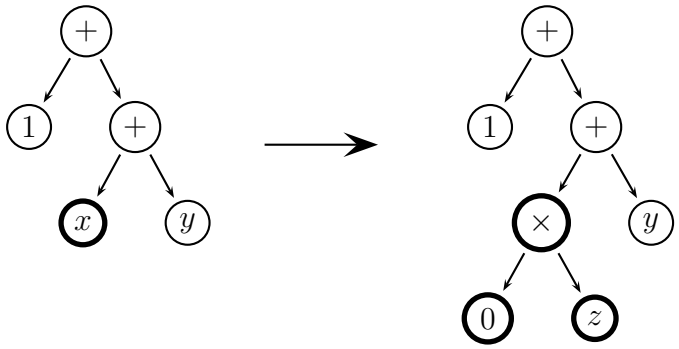



Figure 2.3: An example of the mutation operator. Assume that give variable set is $\{x, y, z\}$, the operation set is $\{+, -, \times, /\}$, and the constant set is $\{0, 1\}$.

- 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

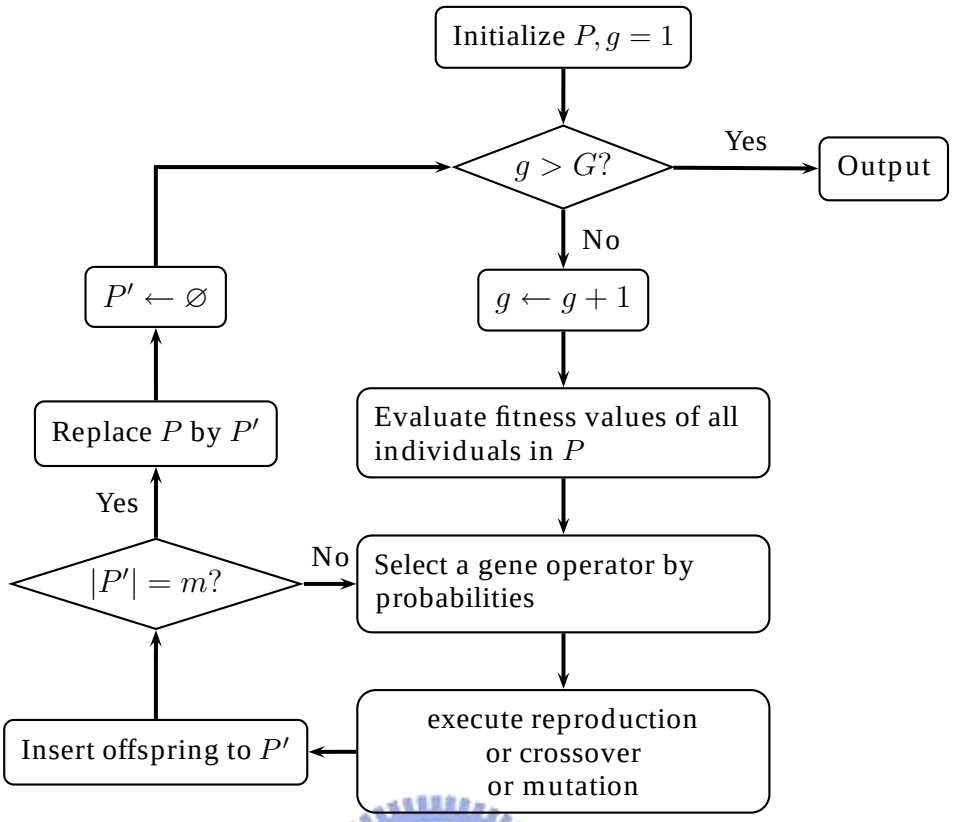



Figure 2.4: The flowchart of GP evolutionary processes. $G$ is the maximum generation, $P$ is the current population, $P'$ is the population of next generation, and $m$ is the population size.

$k = 1$, 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

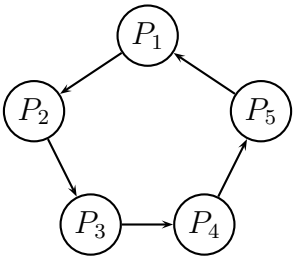



Figure 2.5: An example of a five-population multi-population GP with a circle topology. An arrow indicates the direction of migration between two populations.

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.<br>2. Different populations can use different configurations such as individual length or operators. | 1. Small isolated populations have limited diversity of individuals.<br>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.<br>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<br>(a) Communication topology<br>(b) Frequency of migration<br>(c) The number of migrants<br>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.

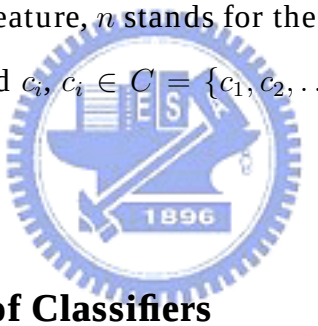
In general, the predictive accuracy is the most important factor. Most classifiers are developed or improved with the goal of achieving high accuracy. There are many accurate evaluation methods. Most of classification models are supervised in that they require a set of training instances. They use a 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 $\xi$ 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 $\zeta$ is the penalty parameter.

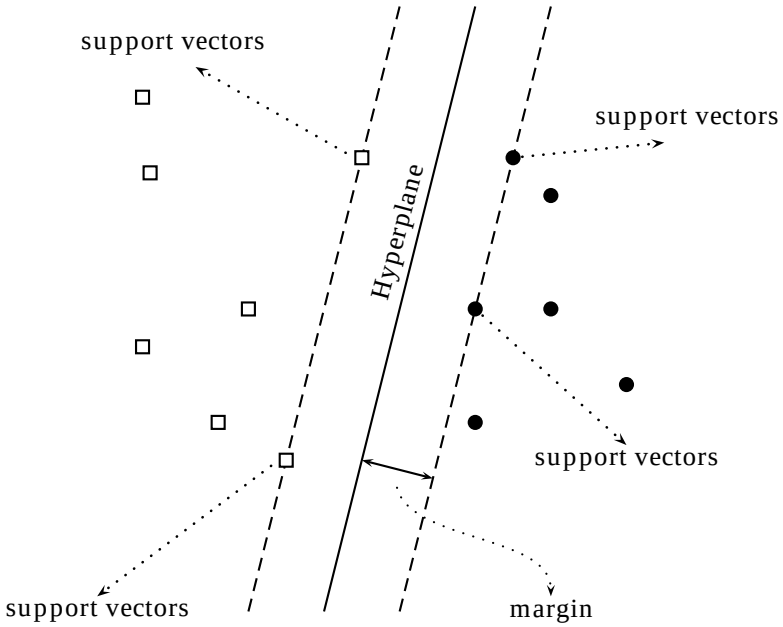



Figure 2.6: An illustration of SVM. Square points and circle points stand for instances of different classes.

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 $\Phi$ 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

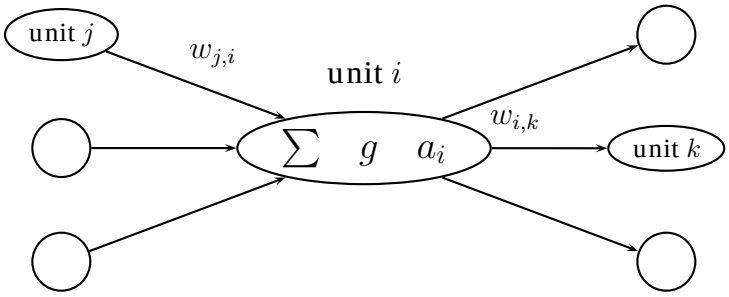



Figure 2.7: An illustration of a NN unit.

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

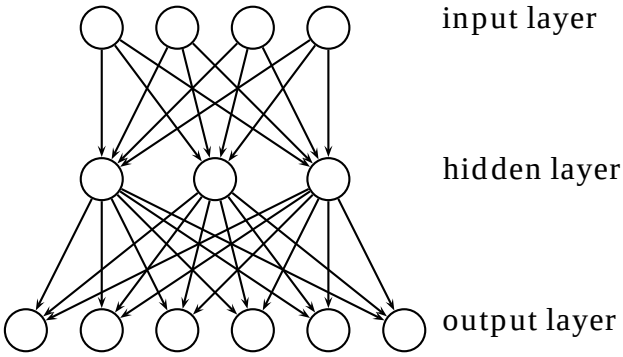



Figure 2.8: An example of multi-layer NN. This example shows one input layer, one hidden layer, and one output layer. The input layer, the hidden layer, and the output layer contain four units, three units, and six units, respectively.

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 $l$-

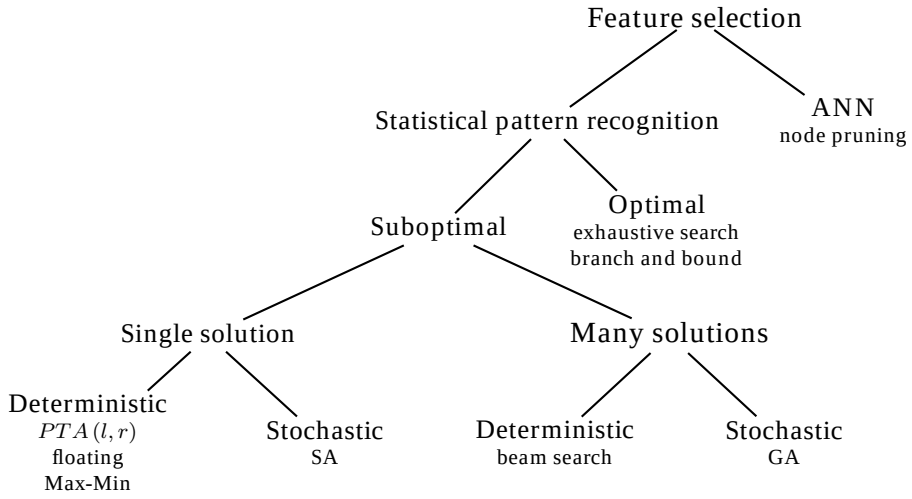



Figure 2.9: Taxonomy of feature selection algorithms (cited from [31]).

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-

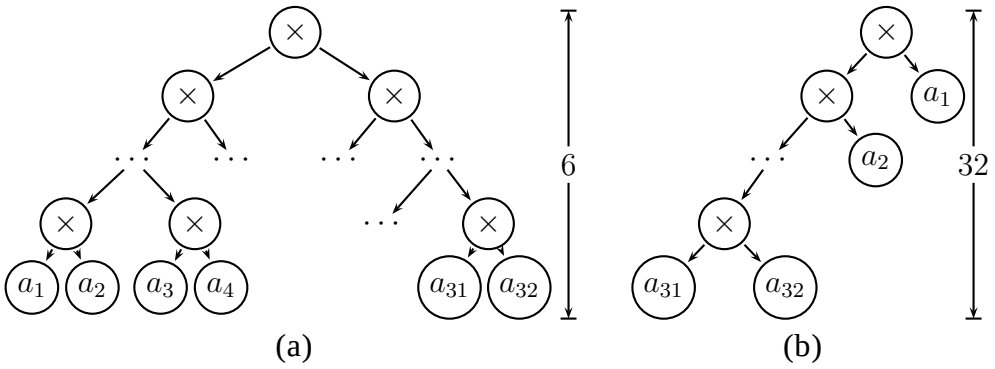



Figure 2.10: Two tree representations of function $F = \prod_{i=1}^{32} x_i$. (a) A balance tree uses 6 of heights. (b) A skew tree uses 32 of heights.

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

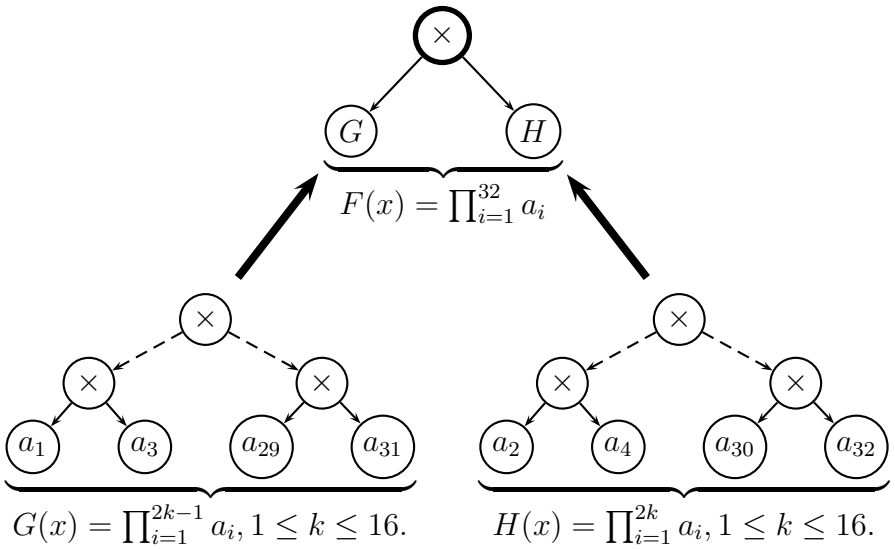



Figure 2.11: Combining functions $G$ and $H$ to form function $F$. $G$ and $H$ are small functions combined by a tree rooted by a $\otimes$ node.

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 $\Phi$ [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

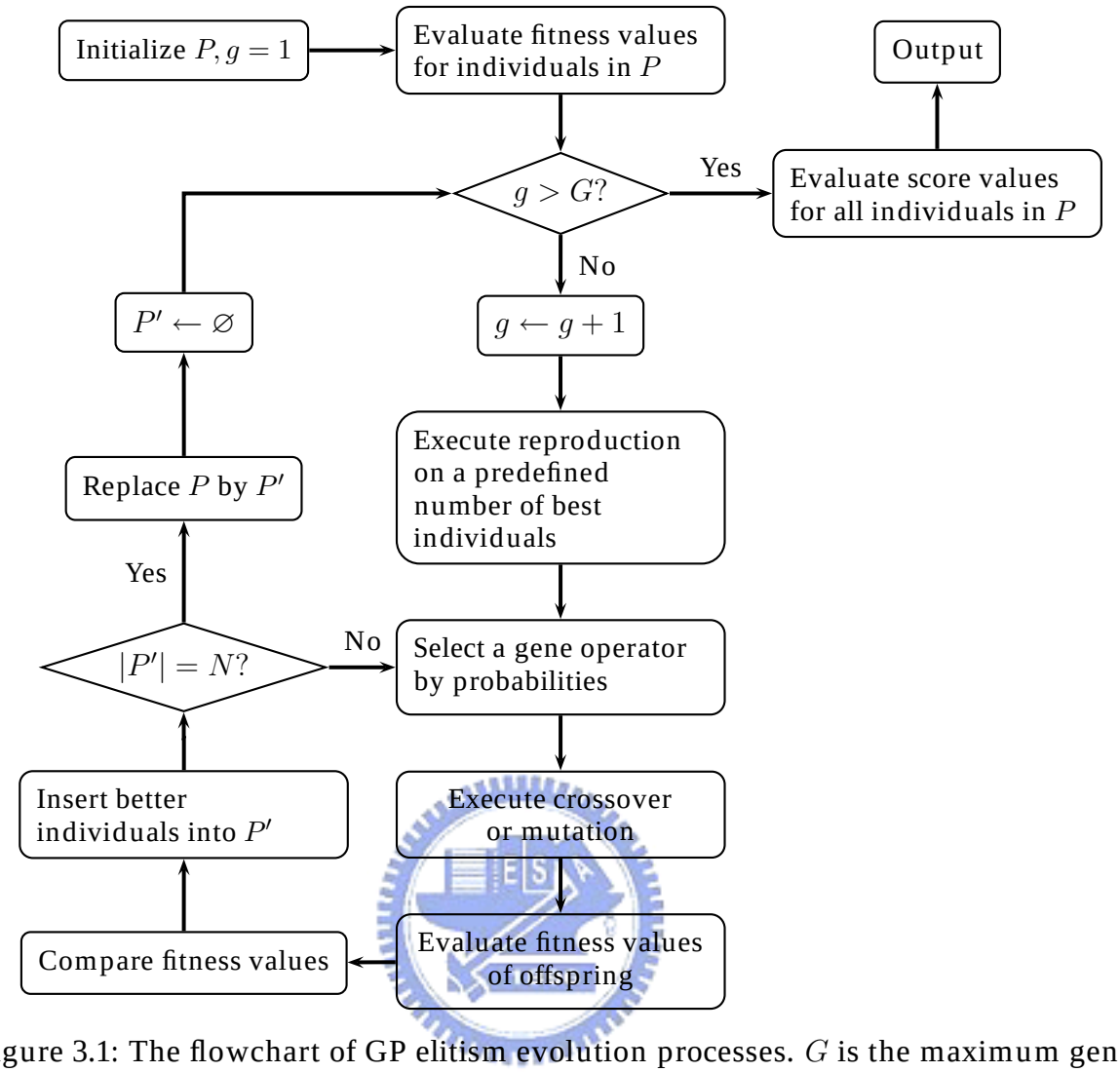



Figure 3.1: The flowchart of GP elitism evolution processes. $G$ is the maximum generation, $P$ is the current population, $P'$ is the population of next generation, and $N$ is the population size.

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, $f_{\text{MAX}}$ is the fitness value of the best individual in generation $g$; and $f_{\text{AVERAGE}}$ 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 $f_{\text{MAX}}$ 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 $100$th generation.

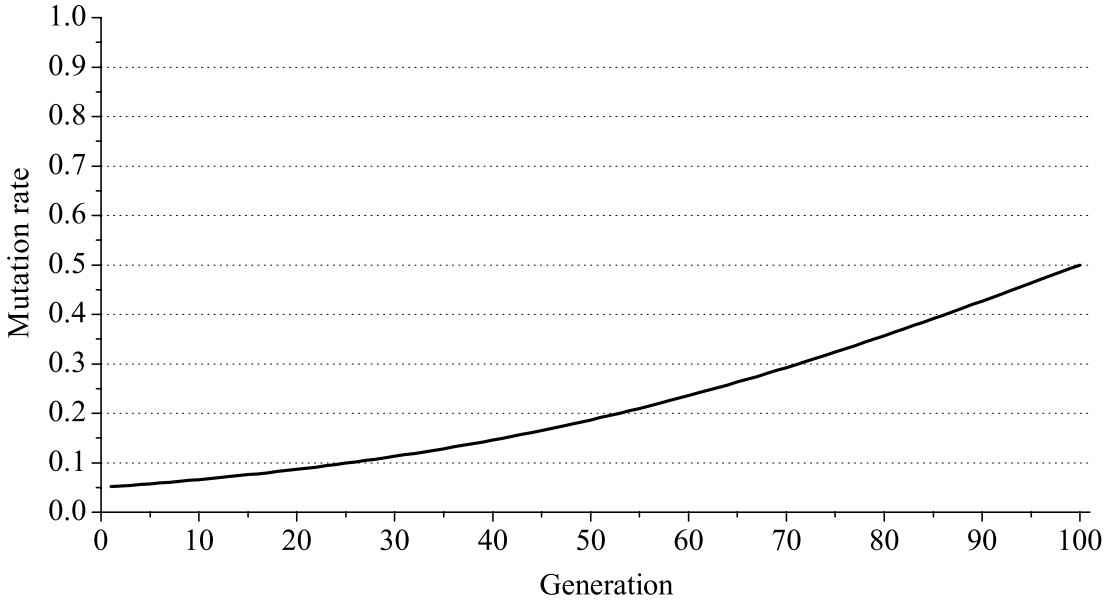



Figure 3.2: The growing curve of $p_m$ using *AMPT*.

## 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:

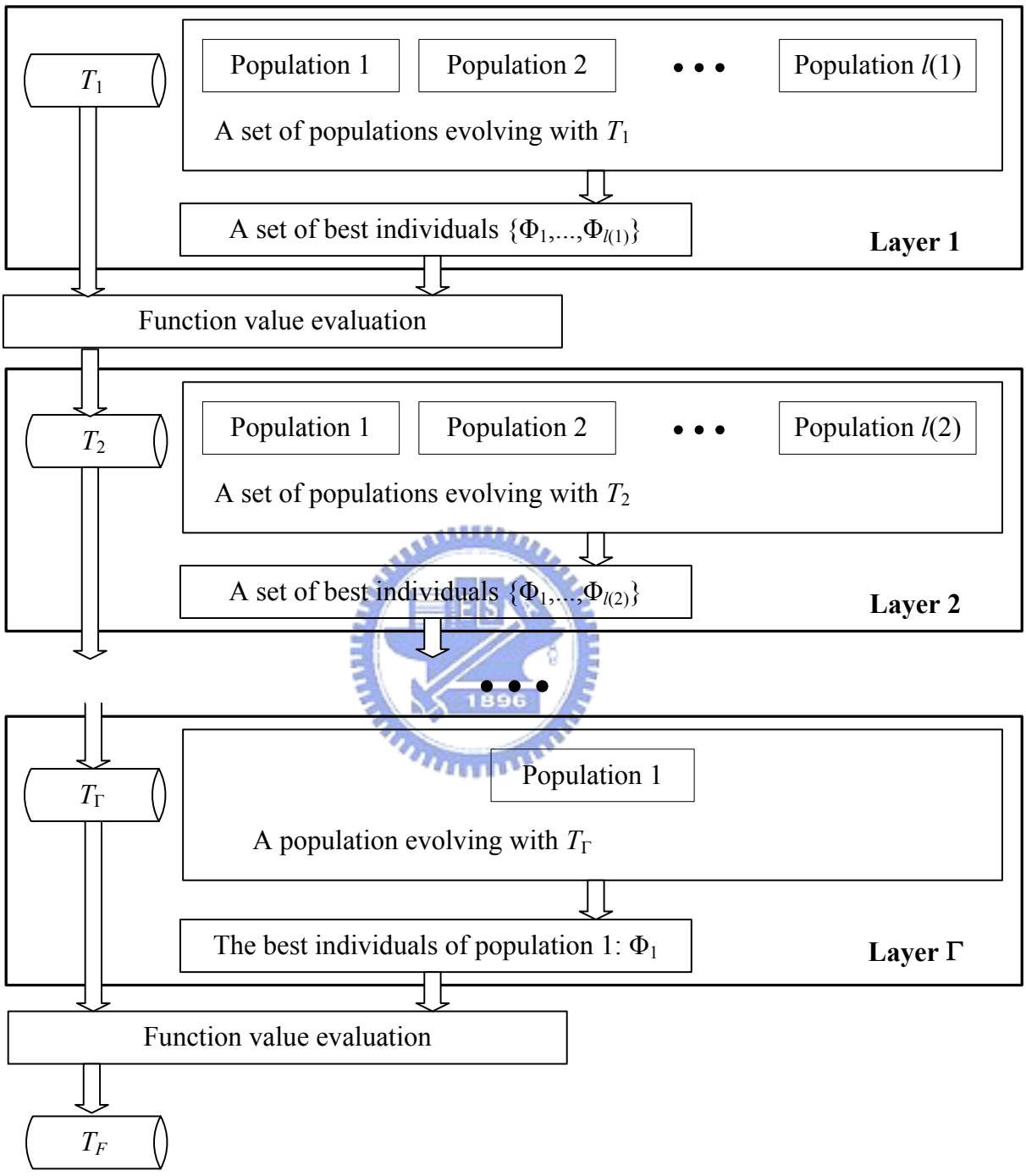



Figure 3.3: Flowchart of LAGEP. $\Gamma$ is the number of layers, $l(i)$ stands for the number of populations in layer $i$.

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

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 $\Phi^*$ 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 $\Phi_{ij}$. 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 $\Gamma$ 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 $\Phi$. 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 $\Phi$ 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