IMPROVED GENETIC PROGRAMMING TECHNIQUES FOR DATA CLASSIFICATION

A Dissertation
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Nailah Shikri Al-Madi

In Partial Fulfillment of the Requirements
for the Degree of
DOCTOR OF PHILOSOPHY

Major Department:
Computer Science

December 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

IMPROVED GENETIC PROGRAMMING TECHNIQUES FOR DATA CLASSIFICATION

By

Nailah Shikri Al-Madi

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

DOCTOR OF PHILOSOPHY

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. William Perrizo

Dr. Changhui Yan

Dr. Canan Bilen-Green

Approved:

12/16/2013
Date

Brian Slator
Department Chair

ABSTRACT

Evolutionary algorithms are one category of optimization techniques that are inspired by processes of biological evolution. Evolutionary computation is applied to many domains and one of the most important is data mining. Data mining is a relatively broad field that deals with the automatic knowledge discovery from databases and it is one of the most developed fields in the area of artificial intelligence. Classification is a data mining method that assigns items in a collection to target classes with the goal to accurately predict the target class for each item in the data.

Genetic programming (GP) is one of the effective evolutionary computation techniques to solve classification problems. GP solves classification problems as an optimization tasks, where it searches for the best solution with highest accuracy. However, GP suffers from some weaknesses such as long execution time, and the need to tune many parameters for each problem. Furthermore, GP can not obtain high accuracy for multiclass classification problems as opposed to binary problems.

In this dissertation, we address these drawbacks and propose some approaches in order to overcome them. Adaptive GP variants are proposed in order to automatically adapt the parameter settings and shorten the execution time. Moreover, two approaches are proposed to improve the accuracy of GP when applied to multiclass classification problems. In addition, a Segment-based approach is proposed to accelerate the GP execution time for the data classification problem. Furthermore, a parallelization of the GP process using the MapReduce methodology was proposed which aims to

shorten the GP execution time and to provide the ability to use large population sizes leading to a faster convergence.

The proposed approaches are evaluated using different measures, such as accuracy, execution time, sensitivity, specificity, and statistical tests. Comparisons between the proposed approaches with the standard GP, and with other classification techniques were performed, and the results showed that these approaches overcome the drawbacks of standard GP by successfully improving the accuracy and execution time.

ACKNOWLEDGMENTS

First and foremost, I would like to extend my profound thanks and appreciation to professor Simone Ludwig who supervised this research since it was the idea that emerged to its current form. She was a base and reference rich with information and ideas, and her remarks and recommendations were often able to overcome the difficulties encountered by this research. She is an example of a sincere professor who is filled with knowledge, and has been generous in advices and guidance. I offer her the highest appreciation and thanks for all the profitable and enjoyable courses she has taught us, and for her direct to this research. I also want to thank the committee of professors, Prof. William Perrizo, Prof. Changhui Yan, and Prof. Canan Bilen-Green, who have given me the honor of reading and discussing my research, and give me the chance to improve it by their important and useful remarks.

Special and un-describable thanks and grateful to my parents for their support and encouragement that kept working to follow my dream, for the love and guidance they give me every day. Special Thanks to my beloved brothers for their endless support. Infinite thanks to my husband Ibrahim, for his support and inspiration, for the strength he give me, for his believing in me and the love he give me every day, my world is a better place because of him. Also thanks to my daughter Nada for giving me the hope for tomorrow and the motivation to do something better in this world.

Finally I want to thank every person who believed in me and supported me in every form.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

This chapter introduces a high-level description of the work presented in this dissertation. It starts off by elaborating the concepts behind Evolutionary Computation and Data Classification. It then discusses the motivation and the problem statement of this dissertation. Afterwords, a description of the core contributions of this dissertation are presented. Finally, it concludes by giving an overview of the dissertation chapters, supported by publications.

1.1. Background

Genetic Programming (GP) is one of the evolutionary computation approaches that is known as a powerful tool for optimization and problem solving, and has been applied to a wide variety of applications. Data classification is one field that GP has been applied and proved to be a successful and efficient technique. Although GP proved its effectiveness in solving classification problems, but on the other hand it suffers from some drawbacks. In this dissertation we discuss these drawbacks and propose approaches to tackle them. Before discussing our proposed approaches (discussed in next chapters), a general description of the main concepts is presented such as Evolutionary Computation, Genetic Programming, and Data Mining (classification). In addition, the approach of how genetic programming is used to solve a data classification problem is outlined.

1.1.1. Evolutionary Computation

The term optimization describes a process whereby the search for the optimum solution from a set of candidate solutions is sought. Based on specific performance criteria, the optimum solution is searched for. The field of optimization is broad and has found applications in manufacturing, finance, engineering, and logistics to name a few. Before the optimization process can be started, all problems to be optimized should be formulated as a system with its status controlled by a few input

variables and its performance specified by a well-defined objective function. The goal of optimization is to find the best value for each variable in order to achieve satisfactory performance. In practical terms, this means to accomplish a task in the most efficient way or the highest quality or to produce maximum yields given limited resources.

There are many different optimization techniques that exist, such as, Evolutionary algorithms which are optimization techniques that are inspired by processes of biological evolution. Evolutionary algorithms start by creating a population of solutions and apply natural operations to the individuals in order to produce a new population. This iterative process continues until the best solution is found or a predefined number of generations are reached. Natural operations include the selection process which implies the survival of the fittest concept; the best candidates are chosen to undergo recombination and mutation. Given a function to be maximized, the fitness of the individuals is calculated using a fitness measure which is based on the problem to be solved. Selecting two candidates and recombining them, results in one or more new candidates; whereas mutation is only applied to one candidate and results in one new candidate. There are two fundamental components in this evolutionary process that form the basis of evolutionary algorithms [1]:

- Variation by combination and mutation create the necessary diversity and;
- Selection provides the quality.

The combined effect of variation and selection leads to improving fitness values in consecutive populations.

1.1.2. Genetic Programming - GP

GP [1] is one of the evolutionary computation approaches that is known as a powerful tool for optimization and problem solving, and has been applied to a wide variety of applications. GP is an effective technique since it automatically solves

problems without requiring the user to know or specify the form or structure of the solution in advance. Thus, it can be used anywhere when neither the solution nor its structure is known to the user. GP solves the problems by generating programs, and these programs consist of mathematical (+, -, *, /,...etc) and logical functions (if, ifElse, lessThan,...etc.) and terminals (variables (X, Y, Z,...etc.) and constants(2, 5, 8,...etc.). A fitness function is used to evaluate the goodness of the solution provided by each program. The main objective of GP as an optimization process is to maximize or minimize the fitness value based on the target problem.

GP is distinguished from other evolutionary algorithms by the use of a tree representation of variable size rather than of a linear string of fixed length representation as in genetic algorithms. This flexible representation helps to automatically discover the underlying structure of the data. One primary difficulty however, is that the number of solutions may become excessive without any improvement of their generalizability. However, GP has proven to be a very powerful optimization technique in many application areas, such as evolving a computer program that plays a board game, where the parsing of trees is necessary, and thus the GP approach seems to be the best option [1]. An example of a tree representation for a GP program is shown in Figure 1.1, where the program consists of three functions (F1, F2, F3) and three variables (A,B,C) and one constant (2), where the program can be written in LISP language [2] as: (F1 (F2 A 2) (F3 B (F1 B C))).

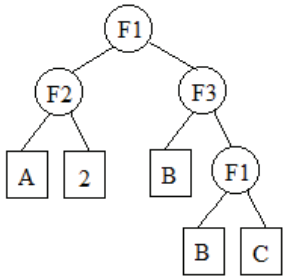


Figure 1.1: GP Program

The GP process has four main steps. It starts with initializing a random population of programs, then a fitness ranking is calculated for each program using the fitness function (for the classification task, the fitness function indicates the classification accuracy). After that, a selection process chooses two programs (parents) in order to generate new programs (children). The new children are generated in the fourth step using the natural operations of crossover and mutation. All these steps are repeated until a new population with the same size than the old population is produced, as shown in Figure 1.2. This process is repeatedly applied until a stopping criterion, such as reaching a predefined number of generations is reached, or finding a solution with the best fitness. Natural operations can be defined as follows:

- Selection: choosing two programs using one of the selection methods such as; Roulette wheel selection [3], Tournament selection [4], Rank selection[5] ...etc
- Crossover: is applied on an individual by simply switching one of its nodes with another node from another individual in the population. With a tree-based representation, replacing a node means replacing the whole branch.
- Mutation: is applied to a single individual by either replacing a function with another function (with condition to take the same number of argument as the tree structure not to be changed) or replacing a terminal with another terminal.

Before starting the GP run many important parameters must be defined, for example; the fitness function to evaluate the program quality, population size, number of generation, functions and constants to be used, and crossover and mutation probabilities.

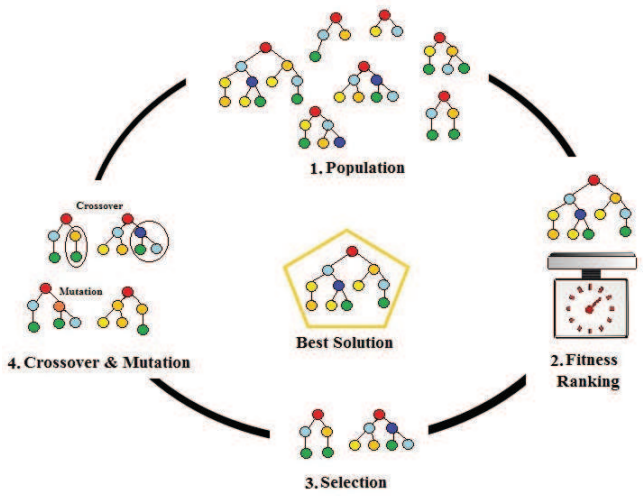


Figure 1.2: GP Process

The actual solution that results from a given GP run is dependent on many factors. Some important factors are the genetic material contained in the initial population, the choices of selection process and crossover type, and the crossover and mutation probabilities. The random choices of these factors imply that when GP is re-applied different outcomes are possible. In order to develop a good quality solution, there are two choices:

1. Perform multiple runs with applicable settings, and take the average and standard deviation of the results to indicate the quality of the GP solution.

2. Use a large population or large number of generations: many more points in the search space will be sampled, and this greatly improves the convergence of the solution.

1.1.3. Data Mining - Classification

Data mining is a relatively broad field that deals with the automatic knowledge discovery from databases and it is one of the most developed fields in the area of artificial intelligence. Given the rapid growth of data collected in various realms

of human activity and their potential usefulness requires efficient tools to extract and make use of the gathered knowledge. One of the important data mining tasks is classification that is applied in many different fields, which is the task of automatically categorizing data into different classes. The main idea behind the classification technique is to build a model (classifier) that distinguishes two (binary classification) or more classes (multiclass classification) with the goal to accurately predict the target class for each item in the data [6]. Classification model effectiveness is measured by calculating the accuracy outlining how accurately the model can predict a new data class.

There are many techniques that can be used to do the classification process such as decision trees, Bayes networks, genetic algorithms, genetic programming and many others. GP was found to be successful for classification problems and has emerged as a powerful technique for classifier evolution [7].

1.1.4. Evolutionary Computation and Data Classification

Evolutionary algorithms are very powerful tools that can be utilized to make use of knowledge hidden in the data collected [8]. For a classification problem, numerous techniques have been applied such as expert systems, artificial neural networks, and evolutionary algorithms. Among these approaches, evolutionary algorithms have emerged as a promising technique in dealing with the increasing challenge of data mining in the medical domain [9]. These challenges include extraction of comprehensible knowledge from the growing volume of data, and solving medical problems like diagnosis, prognosis, imaging and signal processing, planning, and scheduling [10].

GP proved its effectiveness in many classification problems [11, 7, 12]. Solving a classification task using GP indicates that each program presents a classifier model that distinguishes two (binary classification) or more classes (multiclass classification) in order to apply the classifier to new data that determines the class of the data in-

stance. GP has been used for classification problems in many research investigations. For example, GP was used to investigate the prognosis of breast cancer and classify them into two classes resulting in an accuracy of GP that outperforms the linear programming approach [13]. For classification problems, this value resulting from each GP program represents the class label. The goal of GP as an optimization process is to maximize the number of correctly classified records, which is referred to as the accuracy of the classifier. For binary classification, this process is simple since GP can be configured to return a boolean value that represents the class label. For multiclass classification on the other hand, the transformation of the resulting numeric values into the target class labels is not so trivial.

The fitness function for a classification task is the number of correctly classified records, the classifier accuracy, which GP maximizes throughout the generations. In the fitness evaluation process, each program is applied on all dataset records; therefore, the execution time increases corresponding to the dataset size. Moreover, the fitness function takes longer to evaluate the correctly classified records (classifier accuracy).

The main drawback that prevents the use of GP for real time problem solving is its long execution time since it is an iterative process. GP's run time mainly depends on the number of generations, the population size, and the fitness evaluation, whereby the fitness evaluation is the most computational intense portion in GP [14]. Moreover, as mentioned in [15], many typical GP problems do not have large sets of fitness cases for two reasons: first, evaluation has always been considered computationally expensive, and secondly, it is very difficult to evolve solutions for harder problems. For all these reasons, the acceleration of the GP process is needed, which can be achieved by the parallelization of the GP process.

1.2. Motivation and Problem Statement

GP provides many advantages that make it suitable for solving data classification tasks. The advantages gained from GP are:

- GP has a wide search area (specially with the use of crossover or when using a large population), thus it is very likely to cover all the search space and find the optimal solution.

- GP's flexible representation supports its usage for different types of data and different types of problems.

- The resulting program (classifier) has a very short execution time as it is formed from simple functions (mathematical and logical).

- The usage of the fitness function to evaluate the quality of the program (classifier) provides the ability to use different or a combination of constraints over the classifier, such that the accuracy, sensitivity, specificity, the size of the program (classifier), or a combination of them can be used.

However, GP has also some disadvantages, which are:

- Many parameter settings need to be defined and tuned for each problem to be solved before the start of the GP run.

- Very long training times prevent GP from being used for real time applications. This long execution time is a result of its iterative process and specially when using large population sizes and large numbers of generations. Moreover, the execution time increases with increasing dataset sizes.

- GP does not obtain high accuracy results when used for multiclass classification as compared when used for binary classification.

In this dissertation, we want to take advantage of the benefits of GP to solve classification problems and at the same time address the disadvantages GP suffers from. Therefore, in the second chapter we propose three adaptive GP variants that address the GP setting adjustments and the long execution time. In the third chapter, the accuracy for the classification of binary and multiclass datasets is improved. The long execution time of GP is investigated in the fourth chapter proposing a Segment-based GP approach, which accelerates the GP run time. A parallelization version of GP is proposed in Chapter 4, which aims to shorten the GP execution time further, and to provide the ability of using very large population sizes to cover the search space to find the optimal solution.

1.3. Contributions

This dissertation makes many contributions towards improving genetic programming for solving classification problems. The contributions are:

1. GP's long execution time and the need to define many parameter settings before the GP run are investigated. The investigation leads to the proposal of three GP variants which automatically adapt the GP parameters and perform the classification tasks faster. The first variant considers the selection process, which is based on ensuring that the fitness of the next generation is better than the previous one. The second variant concerns the crossover and mutation probabilities that are adaptively changed based on the fitness of the chromosome by protecting the fittest chromosomes from these operations. The third variant is an adaptive function list, whereby the function list is adapted based on functions that performed well during the earlier classification task.

2. GP does not achieve high accuracy for multiclass classification problems as compared to binary classification problems, therefore, two approaches were proposed in order to improve the accuracy results. The first approach uses

the K-means clustering technique in order to transform the produced numeric value of GP program into class labels. The second approach uses a discretization technique to perform the transformation. Both approaches have shown good improvements on the accuracy values compared to the standard GP, and competitive accuracy values compared to the state-of-the-art classifiers.

3. To address the long execution time of GP resulting from the fitness evaluation process but at the same time keeping the good accuracy values of GP, a segment-based GP technique is introduced. Segment-based GP reduces the execution time of GP by partitioning the dataset into segments, and using the segments in the fitness evaluation process.

4. The fourth contribution is a MapReduce GP, which parallelizes GP using the MapReduce methodology in order to speed up the classification process. The proposed approach aims to provide the ability to use large population sizes, thus, finding the best result in fewer numbers of generations.

1.4. Dissertation Overview

This dissertation is in paper-based format, where each chapter has been published or submitted during the PhD investigation.

- **Chapter 2: Adaptive Genetic Programming applied to Classification in Data Mining**

  This chapter proposes an adaptive GP that automatically adapts the parameters of the GP and performs the classification task faster than standard GP. This adaptive GP has three variations, the first considers the selection process, the second concerns the crossover and mutation probabilities, and the third variant is an adaptive function list.

  This chapter is derived from the following publication:

- *Nailah Almadi and Simone A. Ludwig, "Adaptive Genetic Programming applied to Classification in Data Mining", In Proceedings of the Fourth World Congress on Nature and Biologically Inspired Computing (IEEE NaBIC'12), November 2012, Mexico City, Mexico.*

- **Chapter 3: Improving Genetic Programming Classification for Binary and Multiclass Datasets**

  This chapter proposes two approaches in order to improve the GP classification task. One approach uses the K-means clustering technique in order to transform the produced value of GP program into class labels. The second approach uses a discretization technique to perform the transformation.

  This chapter is derived from the following publication:

  - *Nailah Almadi and Simone A. Ludwig, "Improving Genetic Programming Classification For Binary And Multiclass Datasets", In Proceedings of the IEEE Symposium Series on Computational Intelligence - SSCI 2013, April 2013, Singapore.*

- **Chapter 4: Accelerating the Fitness Evaluation of Genetic Programming**

  This chapter proposes a segment-based GP which aims to accelerate the GP fitness evaluation process by reducing the training dataset size but at the same time covering the whole training dataset.

  This chapter is derived from the following publication:

  - *Nailah Almadi and Simone A. Ludwig, "Segment-based Genetic Programming", In Proceeding of the Genetic and Evolutionary Computation Conference (ACM GECCO'13), Amsterdam, the Netherlands, July, 2013, ACM.*

(Short paper)

- *Nailah Almadi and Simone A. Ludwig, "Accelerating the Fitness Evaluation of Genetic Programming", Submitted as a journal paper.*

- **Chapter 5: Scaling Genetic Programming for Data Classification using MapReduce Methodology**

  This chapter proposes a parallelization of the GP algorithm based on the MapReduce methodology in order to speed up the classification process. The proposed approach not only accelerates the execution time of GP, it also provides the ability to use high population sizes, thus, finding the best result in fewer numbers of generations.

  This chapter is derived from the following publication:

  - *Nailah Almadi and Simone A. Ludwig, "Scaling Genetic Programming for Data Classification using MapReduce Methodology", In Proceedings of the Fifth World Congress on Nature and Biologically Inspired Computing (IEEE NaBIC'13), August 2013, Fargo, ND, USA.*

- **Chapter 6: Conclusion and Future Directions**

  This chapter concludes the dissertation by summarizing the contributions of this research and providing insights on future work that could be carried out.

CHAPTER 2. ADAPTIVE GENETIC PROGRAMMING APPLIED TO CLASSIFICATION IN DATA MINING

Genetic programming (GP) is one of the effective evolutionary computation techniques to solve classification problems; however, it suffers from a long run time. In addition, there are many parameters that need to be set before the GP is run. In this chapter we propose an adaptive GP that automatically determines the best parameters of a run, and executes the classification faster than standard GP. This adaptive GP has three variations. The first variant consists of an adaptive selection process ensuring that the produced solutions in the next generation are better than the solutions in the previous generation. The second variant adapts the crossover and mutation rates by modifying the probabilities ensuring that a solution with a high fitness is protected. And the third variant is an adaptive function list that automatically changes the functions used by deleting the functions that do not favorably contribute to the classification. These proposed variations were implemented and compared to the standard GP. The results show that a significant speedup can be achieved by obtaining similar classification accuracies.

This chapter is structured as follows: Section 2.1 describes related work in the area of GP including different GP variants. In Section 2.2, our adaptive GP is described. Section 2.3 presents the experiments as well as the results obtained, and Section 2.4 summarizes this chapter.

2.1. Related Work

Related work in the area of GP is manifold. In particular, several adaptive techniques have been applied to GP in the past. For example, in [16] the authors proposed a self-adaptive selection mechanism for genetic algorithms, which was called offspring selection. Offspring selection is based on the idea whereby the fitness value of the evenly produced offspring is compared to the fitness values of its own parents

in order to decide whether or not to accept the offspring as a member for the next generation. The offspring is accepted as a candidate for the further evolutionary process if and only if the reproduction operator was able to produce an offspring that could outperform the fitness of its own parents; where the fitness of the offspring is compared with the worst fitness of the parents, and only is accepted if it is better. This mechanism was tested on some real valued test functions with a scalable degree of difficulty and found to produce better results.

However in [11], the authors used this type of offspring selection in combination with hybrid formula structures combining logical expressions and classical mathematical functions in GP, and it was applied to three medical benchmark classification problems (Wisconsin, Thyroid, and Melanoma) and compared to other machine learning methods (Linear modeling, k-nearest neighbor, artificial neural network, and standard GP). The results showed that this approach outperforms classical machine learning algorithms frequently used for solving classification problems, namely linear regression, neural networks and neighborhood-based classification.

In [17], a swarm-based improvement of the crossover operator was proposed and tested on symbolic regression. The proposed approach consists of two main parts; the first is function couplings, which constructs a matrix that represents a coupling from every function to every other function or terminal. Once this matrix is defined, crossover can be applied to preserve important couplings, where the modified crossover is applied and the probability of choosing a node as root of the sub-tree for crossover is proportional to the amount of pheromone for the coupling with its parent node. Their approach was tested on three test functions by varying the population size. Constructing a matrix and using the swarm intelligence technique is a very time consuming process, which may improve the quality of the GP process but makes the time for a run even worse, especially given that GP suffers from long execution times.

A GP method called GPTM (GP with Tree Mining) was proposed in [18]. The method first identifies the sub-trees repeatedly appearing in the chromosomes of superior individuals (ones with very high fitness), and then protects them from undesirable crossover operations. To find such sub-trees, GPTM uses a FREQT-like efficient tree-mining algorithm. GPTM was evaluated by three benchmark problems, and the results indicate that GPTM is comparable and finds the optimal individual earlier. The tree mining method is a good idea since all solutions in GP are represented as trees; however, mining every tree is computationally very expensive especially for large population sizes.

In [19], an integrated adaptive genetic algorithm was proposed, containing two adaptive techniques. The first technique dynamically adapts the rates of the crossover and mutation operators, and the second adapts the behavior of these operators whether unary (mutation), binary (crossover), or multiary. The main idea is to record the behavior of different types of these operators by using a "mini" genetic algorithm to modify the rates and the used operators. This algorithm was tested using the royal road function, and the results were compared with the standard genetic algorithm. The results show that this algorithm is more robust and less sensitive to errors with the initial parameter setting.

A tree-based crossover operator that probabilistically crosses branches based on the behavioral similarity between the branches was introduced in [20]. Node behavior is defined as the range of values that it propagates upward while the tree is evaluated; where each node records the minimum and maximum values. A distance between the two parents is calculated using these recorded values, then this distance is normalized and used as a probability of the crossover operation. This proposed technique was compared with the GP method without crossover, random crossover, and a deterministic form of the crossover operator in the symbolic regression domain,

and achieved better results. In GP, usually the population size is large, and each chromosome contains nodes that represent the function list and the terminals and constants, therefore, recording the minimum and maximum values for each node adds an overhead to the whole process.

In [21], an adaptive technique for crossover and mutation probabilities in genetic algorithms was proposed. The probabilities are varied depending on the fitness of the solution, whereby higher fitness is protected, while sub-average fitness is totally disrupted. Therefore, the problem of defining the optimal values of these probabilities is solved. After selecting the parents the crossover probability is calculated depending on their fitness and the average probability of the population fitness. If the fitness is high then the resulted probability is low, and if the fitness is low then the resulted probability is high.

This chapter proposes an adaptive GP with three components. The first adaptive component concerns the selection process which is similar to the one proposed in [16, 11], however, the selection is applied not only on the children, but also on the parents. The second component implements the adaptive probabilities of the crossover and mutation processes that were suggested in [21] for genetic algorithms, and test them in GP. And the last adaptive component addresses the function list to be used in GP since it is hard to decide which functions are better to represent the solution. The adaptive function list starts with a complete list of functions, then after some generations the GP updates this list based on the functions that produce the best fitness values. Further details on the proposed adaptive GP approach are discussed in the next section

2.2. Proposed Approach

We propose an adaptive GP approach that consists of three adaptive variations. The first variation can be termed as the adaptive selection process (referred to as SGP from now on) and is based on ensuring that the fitness of the next generation is better than the previous one. This is done by selecting the parents based on their fitness compared to the average of the population's fitness; thereby, only parents whose fitness values are larger than the average of the population's fitness are accepted. Furthermore, the fitness of the children that are produced by the crossover of these parents is calculated for each child and is compared to the average of the old population's fitness. If larger, then this child is selected to be in the next generation. This way we ensure that the next generation's fitness is better than the previous one.

The second adaptive variation involves adaptive probabilities of the crossover and mutation operators. Since these probabilities differ based on the fitness of the solution, higher fitness has a low probability of crossover and mutation to be applied, whereas low fitness has a high probability of these operators to be applied. Keeping in mind that the mutation operator is designed to discover better variants of a single chromosome, whereas, the crossover operator combines useful genetic substructures of multiple chromosomes. These probabilities are calculated, as adopted from [21], as follows:

$$P_{Crossover} = \begin{cases} C_1 * \frac{F_{Max} - F_{PMax}}{F_{Max} - F_{Avg}} & , F_{PMax} \geq F_{Avg} \\ C_1 & , \text{Otherwise} \end{cases} \tag{2.1}$$

$$P_{Mutation} = \begin{cases} C_2 * \frac{F_{Max} - F_{Ind}}{F_{Max} - F_{Avg}} & , F_{Ind} \geq F_{Avg} \\ C_2 & , \text{Otherwise} \end{cases} \tag{2.2}$$

where C_1 and C_2 are constants and are equal to 1 and 0.5 respectively. F_{max} is

the maximum fitness of the population, F_{Avg} is the average of the population's fitness, F_{PMax} is the maximum fitness of the parents, and F_{Ind} is the fitness of the individual chromosome that the mutation operation is applied to. This way, the individuals with high fitness values are protected and copied into the next generation. Using a simple equation to adapt the probabilities that do not need to record data or affect the execution time of the process is an effective way. This variation of GP is called CGP from now on.

The third adaptive variation of the proposed adaptive GP is the idea of an adaptive function list (referred to as FGP from now on). The list of functions that are used in the GP, adapts to the problem and search space. The main goal is to ease the process of selecting the functions that best represent the problem. The idea is based on modifying the function list that is used within the evolutionary process to construct programs, and an update of the function list is performed every 30 generations and after the first 100 generations, in order for the GP to be able to learn the functions that are more suitable for the problem. The number of 30 generations was determined by preliminary experiments since it achieved the best results by evaluating function lists of different generations. The modification process proceeds as follows: first, 10 best fitness programs (Best) and 10 worst fitness programs (Worst) are chosen from the population. Then, the functions used in both lists (Best and Worst) are determined, and the frequency of usage of each function is counted. As an example, the output of this step is shown in Table 2.1.

Then the frequency of each function is compared; if the frequency of Worst $>$ 0 and frequency of best $= 0$, then the function is deleted from the function list, and thus is not be used in the next population. Based on the example above, after this step the functions Add, Subtract, Log and Sine are deleted. At last, an update of the

Table 2.1: Example Of Adaptive Function List

Function	Worst	Best	Function	Worst	Best
Add	2	0	Greater Than	1	1
Subtract	1	0	Exponential	0	0
Multiply	0	2	Log	3	0
Divide	1	2	Sine	1	0
Power	0	0	If Else	0	3

set of functions used in the next generations is performed.

2.3. Experiments and Results

To evaluate the proposed variations and see how effective the different adaptations are to find optimized solutions and/or shorten the run time of GP, experiments were performed using the Java Genetic Algorithms Package (JGAP) [22] on two types of datasets; one with binary classes (true and false), and the other with multiple classes.

2.3.1. Datasets

The experiments are applied on the two types of datasets [23]. The binary datasets are the Wisconsin Diagnostic Breast Cancer (WDBC) dataset that predicts the two breast cancer diagnoses of benign and malignant. The Hepatitis dataset predicts whether a person lives or dies, and the Heart datasets that refers to the presence of heart disease in the patient. The Diabetes dataset is the Pima Indians Diabetes Database and the diagnostic investigated is whether the patient shows signs of diabetes or not. The details for these datasets including the number of features, and the number of records are shown in Table 2.2.

The second type of datasets that were used are multiclass datasets that include the Dermatology dataset, which determines the type of Eryhemato-Squamous disease, the Lymph dataset, which is about the Lymphography disease, the Splice dataset, which is about the Primate splice-junction gene sequences (DNA) with associated

imperfect domain theory, and the CTG dataset, which is about fetal Cardiotocograms (CTGs). This dataset is split into two outcome categories, the first one classifies a morphologic pattern (CTG-Class[1]), and the second is related to a fetal state (CTG-NSP). All details including the number of features and records of these datasets are shown in Table 2.3.

Table 2.2: Binary Datasets

	Dataset	**Classes**	**Features (Filtered)**	**Records**
D1	Breast cancer	2	31 (11)	568
D2	Diabetes	2	8 (4)	768
D3	Heart	2	13 (9)	269
D4	Hepatitis	2	19 (10)	155

Table 2.3: Multiclass Datasets

	Dataset	**Classes**	**Features (Filtered)**	**Records**
D5	Lymph	4	19 (10)	148
D6	Splice	3	61 (22)	3190
D7	Dermatology	6	33 (19)	366
D8	CTG-NSP	3	22 (7)	2126
D9	CTG-Class	3	22 (11)	2126

2.3.2. Experiments

The experiments were conducted as follows. First of all, feature selection was performed on the datasets using WEKA [24], which reduced the number of features as shown in the brackets in Table 2.2 for the binary datasets, and Table 2.3 for the multiclass datasets, respectively. Also, the GP variants were applied on these datasets

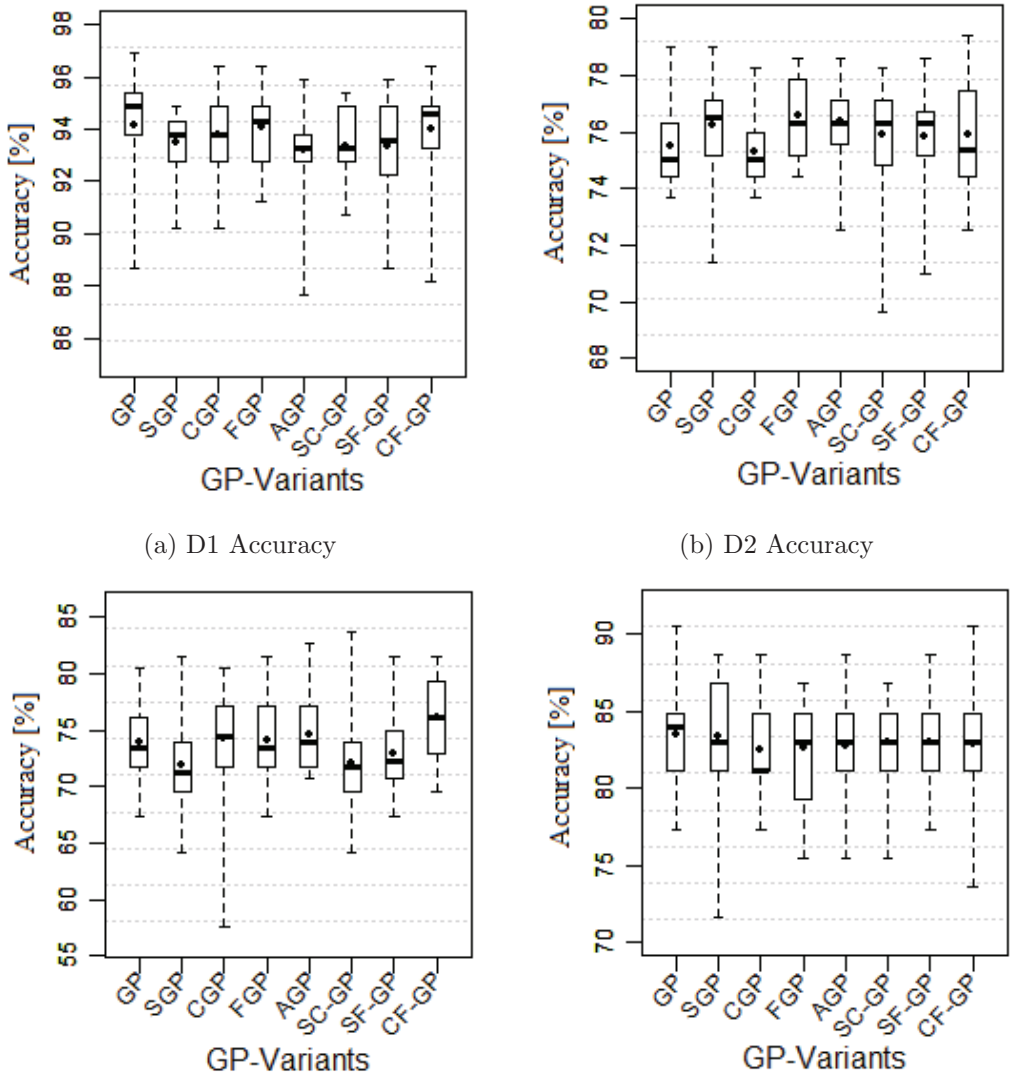
[1]To be used in the next chapters (3 and 4).

using 66% for training, and 34% for testing. After that, our GP algorithm was run using the following settings: population size = 500, number of generations = 1000, crossover probability = 0.5, mutation probability = 0.1, maximum initial depth = 5, maximum crossover depth = 8, function probability = 8, dynamize arity probability = 0.05, new chromosome percent = 0.2.

The experiments applied the standard GP (GP) and the 3 proposed adaptive GP adaptations variations; SGP, CGP, and FGP. Combinations of these adaptive ideas were implemented, such as adaptive selection and adaptive crossover were combined (referred to as SC-GP), and the combination of adaptive selection and adaptive function list (SF-GP), and also the adaptive crossover and adaptive function list (CF-GP). In addition, the combination of all three implementation ideas was done and is referred to as AGP.

2.3.3. Results

To evaluate all adaptive GPs, they are compared with standard GP measuring the accuracy and execution time. The accuracy results for applying the GP and the adaptive GPs on the binary and multiclass datasets are shown in Figure 2, where the solid point represents the average of 30 runs, the solid bar depicts the mean, the lower end of the dashed line represents the minimum accuracy observed by the corresponding GP variant, and the upper end of the dashed line depicts the maximum accuracy value. Figure 2.1(a) shows the results for dataset D1, where GP, FGP and CF-GP (94.14%, 94.11% and 94.02% respectively) all have higher average accuracies compared to other GP variants. Standard GP has the highest maximum accuracy. Regarding the mean value, CF-GP has the closest value compared to the standard GP mean value.



(a) D1 Accuracy

(b) D2 Accuracy

(c) D3 Accuracy

(d) D4 Accuracy

Figure 2.1: Accuracy Results For Binary Datasets

The accuracy results for D2 are shown in Figure 2.1(b), where FGP has the highest average value of 76.6%, whereas standard GP scores 75.5%, however, SGP, AGP, SF-GP and CF-GP all have average accuracies higher than GP with 76.22%, 76.38%, 75.83%, and 75.89%, respectively. For the maximum accuracy values, CF-GP has the highest value. Looking at the mean, GP and CGP both have their means at the same level, but lower than all other GP variants. D3s results shown in Figure

2.1(c) reveal that the GPs average result is 73.94%, whereas CF-GP has the highest average result of 76.05%, and CGP, FGP, and AGP all have accuracies higher than GP (74.27%, 74.02% and 74.60%, respectively). Looking at the maximum values, it is clear that nearly all GP variants have higher values than GP. In addition, the mean values for GP is in the middle compared to the other GP variants, whereby FGP, AGP and CFGP all have higher mean values.

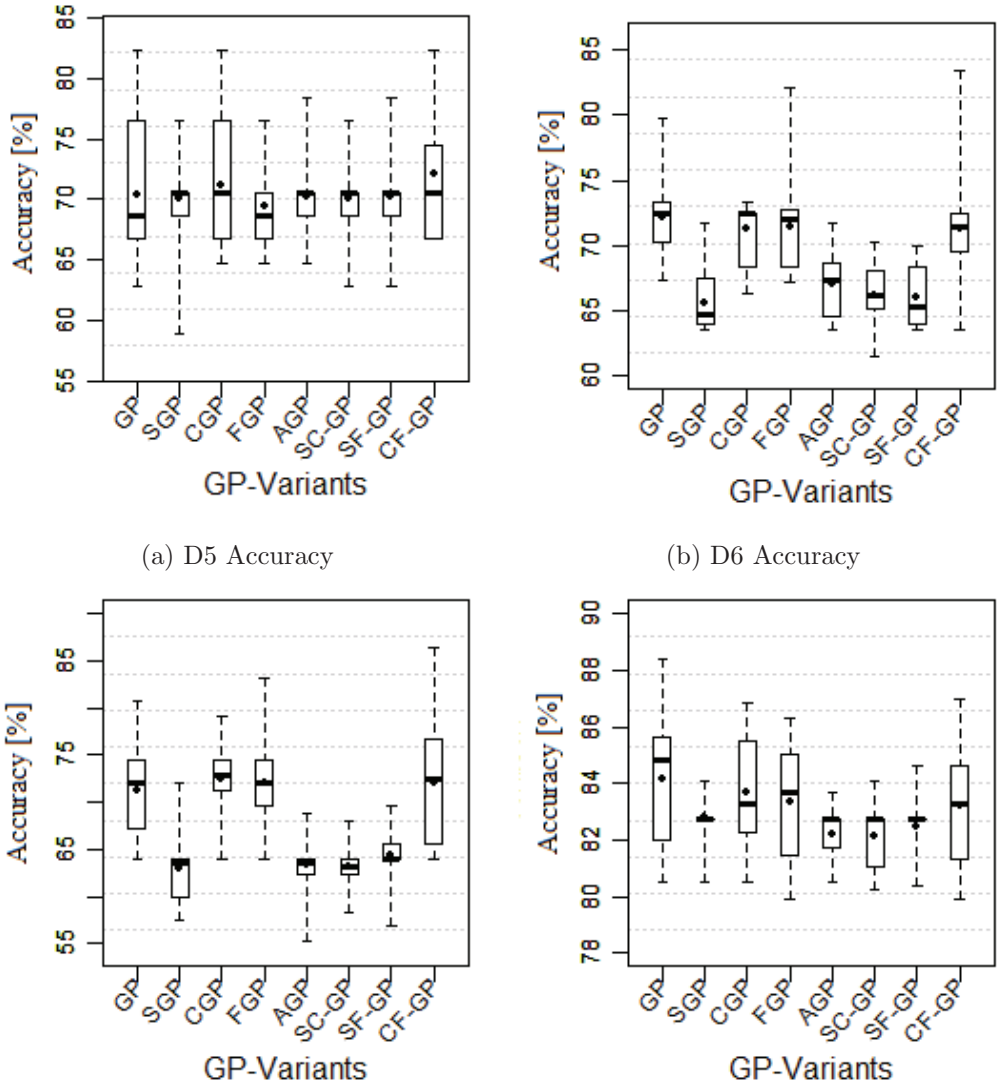
For the last binary dataset, D4, the results are shown in Figure 2.1(d), whereby GP's average result is 83.52%, while SGP has an accuracy of 83.33%, and SC-GP's accuracy is 83.02%. The highest maximum values are achieved by GP and CF-GP. The mean value of GP is noted to be the highest compared to other GP variants.

In summary, it was found from the binary dataset results that for the average accuracy values, FGP and CF-GP both have two results (for D2 and D3) higher than GP with one being very close (D1). SGP, CGP and SF-GP all have one higher value compared to GP (for D2, D3, and D2, respectively). CF-GP scores two highest maximum values for D2 and D4.

For the multiclass datasets, the accuracy results of D5 are shown in Figure 2.2(a), where it is observed that GP's average result is 70.32%, where CGP and CF-GP both have higher accuracies with 71.17% and 72.02%, respectively. Whereas AGP, SC-GP and SF-GP all have close accuracy values (70.13%, 70.06%, and 70.13%, respectively). The highest maximum value is achieved by GP, CGP and CF-GP. The mean value for all GP variants, except FGP, is higher than GP, whereas FGP's mean is at the same level as GP's.

D6's accuracy results are shown in Figure 2.2(b). GP achieves the highest accuracy with 72.13%, however, CGP, FGP and CF-GP all have close values with 71.27%, 71.46%, and 71.34%, respectively. The highest maximum values are found by FGP and CF-GP. The accuracy results for D7 are shown in Figure 2.2(c), where

GP achieved an average accuracy of 71.33%, while CGP, FGP and CF-GP all have higher accuracy values with 72.37%, 71.97% and 72.05%, respectively. The highest maximum values are found by FGP and CF-GP. The mean value of CGP and CF-GP are higher than GP. For D8, as shown in Figure 2.2(d), GP has the highest average accuracy of 84.14%, while CGP, FGP, and CF-GP all have close values (83.68%, 83.32% and 83.18%). In addition, the highest maximum value is also found by GP.



(a) D5 Accuracy

(b) D6 Accuracy

(c) D7 Accuracy

(d) D8 Accuracy

Figure 2.2: Accuracy Results For Multiclass Datasets

To summarize the results of the multiclass datasets, CGP and CF-GP both have two highest average accuracies (for D5 and D7) compared to GP. While FGP has the highest accuracy once (D7) and two close accuracy values (D6 and D8). The highest maximum accuracy is found by CF-GP for three datasets out of the four.

The average execution times for the 30 runs applied to the GP variants on the binary datasets are shown in Figure 2.3. For D1, SGP and CF-GP both have shorter execution times with 67.86 and 69.62 seconds, respectively, compared to GP (75.86 seconds), whereas FGP and SF-GP have close values (79.78 and 76.10 seconds). For D2, GP took on average 94.40 seconds, whereas SGP, AGP, SC-GP and SF-GP all have shorter execution times (71.68, 77.05, 75.13, 76.59 seconds, respectively). The execution time for GP applied on D3, was 40.75 seconds, whereas SGP and SC-GP both have shorter execution times with 36.82 and 40.23 seconds respectively, and AGP and SF-GP both have similar execution times (41.58 and 42.15 seconds). For the last binary dataset D4, GP had the shortest time with 22.84 seconds, followed by FGP with 25.88 seconds and CGP with 29.37 seconds.

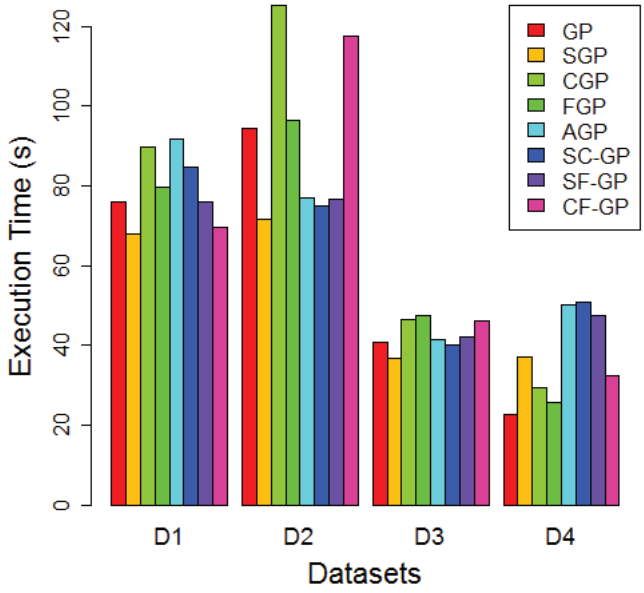


Figure 2.3: Execution Times For All GP Variants - Binary Datasets

For the multiclass datasets the results of the execution time are shown in Figure 2.4 (please note that the y-axis is in logarithmic scale), whereas for D5, GP took on average 19.51 seconds to run, while FGP took 18.36 seconds, and CF-GP took 19.25 seconds. D6s execution time for GP is 555.78 seconds on average, whereas SGP, AGP, SC-GP and SF-GP all have shorter execution times than GP (392.23, 498.73, 513.01 and 446.26 seconds, respectively). For D7, GP has the shortest execution time with 47.14 seconds, closely followed by SGP and CGP with 48.24 and 49.45 seconds, respectively. GPs execution time for D8 is 213.91 seconds, whereby SGP and CGP took 203.75 and 205.14 seconds, respectively, while F-PG, SF-GP and CF-GP all have shorter execution times with 183.06, 179.43 and 178.39 seconds, respectively.

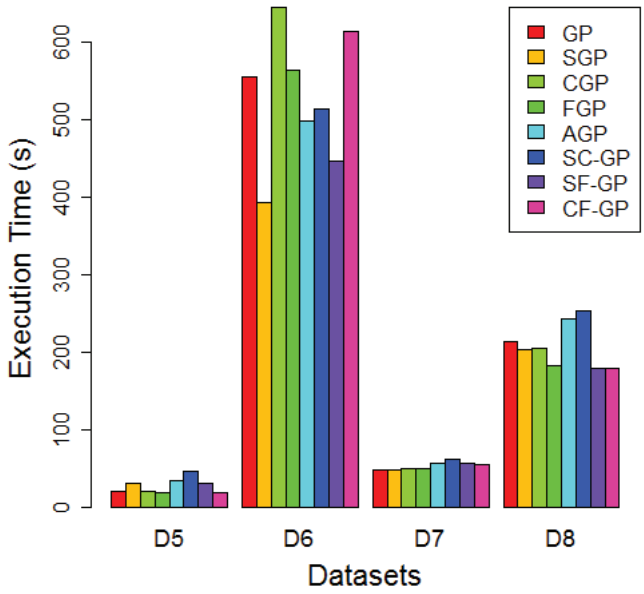


Figure 2.4: Execution Times For All GP Variants - Multiclass Datasets

In summary, the accuracy results show that CF-GP can achieve a higher accuracy compared to GP for both binary and multiclass datasets (4 out of 8 datasets), and can obtain close accuracy values compared to GP for the other 3 datasets, while FGP achieves a higher accuracy in three datasets, and results close to GP for other 3

datasets. Moreover, CGP can achieve a higher accuracy than GP for 3 datasets, and two datasets with a close accuracy compared to GP. While at the same time when focusing on the execution time, CF-GP has a shorter run time compared to GP for four datasets out of eight, and the same goes for FGP, whereby it has two shorter execution times compared to GP. Likewise, SGP has shorter times for 5 datasets; SC-GP and SF-GP have shorter times for 3 datasets. A possible reason for the higher accuracy of CF-GP and FGP is that the function list is being modified in order to only have the functions that suit the problem best. This focuses the process on solving the problem rather than trying many different ways (functions). In addition, for CGP the programs with high fitness are being protected from the crossover and mutation operations, while CF-GP combines these two reasons.

We note that the execution time for the FGP and all combinations have the shortest time due to the smaller function list, which affect the tree size, as mentioned in [25] "A small tree size is a desirable outcome, particularly when an uncomplicated function set is used, as the evolved results may be easier to interpret", therefore, the tree size affects the execution time - the smaller the tree size the shorter the execution time. However, SGP ensures that the accuracy of the parents and their children are higher than the population's average fitness. High fitness values mean that the program best fits to the target problem, thus all programs in the population will fit well to the target problem and the subsequent generations programs will have higher accuracies. Furthermore, most of the generated programs with high accuracy values take less time compared to low accuracy programs, which contain a lot of functions and variables that construct long equations needing longer time to be executed on all records of the dataset.

2.4. Summary

This chapter proposed a modified GP that adapts the parameters of the GP runs automatically and performs the classification task faster than standard GP. This adaptive GP has three variations, the first considers the selection process (SGP), which is based on ensuring that the fitness of the next generation is better than the previous one; this is done by selecting the parents based on its fitness and selecting the produced children also based on its fitness compared to the average of the population's fitness.

The second variant concerns the crossover and mutation probabilities (CGP) that are adaptively changed based on the fitness of the chromosome by protecting the fittest chromosomes from these operations. The third variant is an adaptive function list (FGP), whereby the function list is adapted based on functions that performed well during the earlier classification task.

All these adaptive variations were tested individually, but also combinations were evaluated and compared with standard GP. The measures used were accuracy and execution time. Both, binary datasets and multiclass classification datasets were used. The results revealed that in particular FGP and CF-GP achieve better or similar accuracy values by having shorter execution times. Also CGP and FGP both help in determining the best crossover and mutation probabilities as well as the function list that best suits the problem.

CHAPTER 3. IMPROVING GENETIC PROGRAMMING CLASSIFICATION FOR BINARY AND MULTICLASS DATASETS

GP has shown that good classification accuracy results especially for binary classifications can be achieved, however, for multiclass classification unfortunately GP does not obtain high accuracy results. In this chapter, we propose two approaches in order to improve the GP classification task. One approach (GP-K) uses the K-means clustering technique in order to transform the produced value of GP into class labels. The second approach (GP-D) uses a discretization technique to perform the transformation. A comparison of the original GP, GP-K and GP-D was conducted using binary and multiclass datasets. In addition, a comparison with other state-of-the-art classifiers was performed. The results reveal that GP-K shows good improvement in terms of accuracy compared to the original GP, however, it has a slightly longer execution time. GP-D also achieves higher accuracy values than the original GP as well as GP-K, and the comparison with the state-of-the-art classifiers reveal competitive accuracy values.

This chapter is structured as follows: Section 3.1 describes related work in the area of classification and some proposed improvement techniques, in particular focusing on multiclass problems. In Section 3.2, GP, K-Means clustering, and the discretization techniques are outlined followed by our proposed approaches. Section 3.3 presents the experiments as well as the results obtained, and Section 3.4 summarizes this chapter.

3.1. Related Work

Several techniques have been proposed to tackle classification using GP for both binary and multiclass problems [7]. Some research investigations applied GP in different areas, and others focused on the theoretical side to improve the accuracy of

GP. For binary classification, three main approaches are used. One approach uses zero as a boundary between classes, such that negative values are considered to belong to one class and positive values belong to the other class [26, 27]. The second approach is to make GP execute and produce a boolean value and then use this boolean value as a class label, such as in [28] where logical operators were used to produce "if then else" classification rules. The third approach uses a separate component to determine the class labels for the values of the GP output, while [29] used a linear perceptron. This approach can also be used for multiclass classification problems.

Communal Binary Decomposition (CBD) uses a probabilistic method to model the outputs of programs; where the program's fitness depends on its performance at separating only one pair of classes for a particular binary classification sub-problem. To test CBD, the authors used GP with 5 functions (+, -, *, /, if), and applied it on three image datasets with different class labels (3, 4, and 5 classes). They compared the CBD results with the Program Classification Map (PCM) [30, 31] and Probabilistic Model (PM) [32], and showed that CBD can achieve better accuracy values, however, with a much longer running time. For the multiclass classification process, many approaches were proposed. One approach is to decompose the problem into group of binary sub-problems such as in [12], where the authors first construct a method to decompose a multiclass classification problem into a number of binary sub-problems, solve all these sub-problems in a single GP run, and then combine the binary sub-problems to solve the whole multiclass problem.

Other approaches were done using a separate component, such as in [33], where the authors discuss the texture classification problem, which involves extracting texture features from two or more classes of texture images to train a classifier. They used GP to discover useful texture feature extraction algorithms, where a feature extraction algorithm is considered useful if the feature values have a high classification accuracy.

They integrated the K-Means clustering algorithm into the GP to compute the fitness of the feature extraction algorithm, according to their intuition that "the better the separation, the better the fitness". They did this by computing the overlap between the clusters generated by K-Means, where they computed the average of the feature values for each class given the cluster centroid, then they calculated the midpoint of the two centroids as a cluster boundary, so the points above this boundary belong to one cluster and below this boundary belong to the other cluster. The number of points in the wrong clusters is considered the error. The grey level histograms of different textures were used as input to the evolved programs. The GP function set contains only the + function. They tested their approach using Brodatz as a learning set and the Vistex set for training and testing the classifier. The resulting accuracy was competitive with other feature sets especially when combined with the Haralick feature set [34]. The paper was concluded by stating that the algorithm has captured some texture regularities, but mentioned that it is very difficult to determine what they are. Also, the performance was limited by the fact that there is no spatial information in the histograms.

Static Class Boundary Determination (SCBD) was proposed in [30], in which two or more static pre-defined thresholds/boundaries are applied to the numeric output value of the genetic program, and the ranges/regions between these boundaries are linearly translated into different classes. These regions are set by fixed boundaries at the beginning of the evolution, and remain constant during the run. If there are n classes for a classification task, then these classes are sequentially assigned to n regions along the numeric output value space ranging from negative numbers to positive numbers with n-1 thresholds/boundaries. The first class is assigned to the region with all numbers less than the first boundary; the next class is allocated to all numbers that lie between the first and the second boundaries, and so on.

Slotted Dynamic Class Boundary Determination (SDCBD) and Centered Dynamic Class Boundary Determination (CDCBD) were proposed in [35], where in CDRS the class boundaries are dynamically determined by calculating the center of the program output value for each class. The algorithm starts by initializing the class boundaries as in SCBD, then during the evolution process of each class it calculates the center of the class based on an equation, then it calculates the boundary between every two classes by taking the midpoint of the two adjacent class centers, and then performs the classification based on the new boundaries. Modifying the class boundaries was done to the training examples after every five generations to balance between evolution and class boundary determination. In SDCBD, the output value of a program is split into certain slots. When a large number of slots are used, a large amount of computation is required.

In their experiment, they used 100 slots derived from the range of [-25, 25] with steps of 0.50. Since the input features (terminals) are scaled into [-1, 1], each slot is assigned to a value for each class. First, it evaluates each genetic program based on the SCBD method. Then, it calculates the slot values for each class based on the program output value. After that, it dynamically determines which class each slot belongs to by simply taking the class with the largest value at the slot. However, in case a slot does not hold any positive value, that is, no programs produce any output at that slot for any training examples, then this slot is assigned to the class of the nearest neighboring slot. This method is applied after every 5 generations, similarly to the CDCBD method.

Another multiclass classification technique is presented in [36], where two populations are evolved simultaneously, one contains fuzzy rule sets and the other contains membership functions; both work together to effectively adapt to each other. Moreover, in [25], the authors proposed an Evolved Class Boundary (ECB) approach that

draws a boundary between classes by calculating the mean of the individual output for each class on the training data, and then calculates the midpoint of these two means. They tested their approach using 3 binary datasets, where they balanced the data to have equal classes, by removing some records. They compared their results with the static class boundary [30] and the centered dynamic class boundary approach [36]. The results showed that ECB obtained a good testing accuracy in 2 out of 3 experiments. Also, they compared the ECB accuracy results with other works, where their rank was between 4th and 8th (out of 27).

For the binary classification, configuring the GP to return a boolean value may lead to a loss in quality of the equation created by the GP program since less information is returned. In addition, using zero as a boundary is not very effective since the GP program may produce positive or negative values for both classes. Therefore, a technique is needed to save the GP's program numerical output and transform it into a class label. However, for the multiclass classification problems, related work proposed a technique that finds the boundaries between the classes, also referred to as discretization, by finding specific boundaries for the overall dataset. In our work, we want each program to be an independent classifier that produces different output, and therefore, must have its own boundaries. Moreover, the related approach that decomposes the multiclass classification problem into groups of binary classification problems results in a higher overhead since it has to divide and execute more runs in order for the results to be combined at the end.

Therefore, this chapter proposes two techniques to solve the classification task for both binary and multiclass data by preserving the GP program structure (the equations), and transforming the GP output into class labels for each program (classifier) in order for the GP evolution to be trained to find the best classifier with the best accuracy. In our proposed approaches, K-Means and discretization techniques

are used. In the first approach, K-Means is used in a different way than [33], where the authors used K-Means and find a midpoint between clusters to use as a boundary, however, we used K-Means to create clusters and then use these clusters to calculate the accuracy by applying majority voting.

3.2. Proposed Approach

Given that our proposed approaches are based on GP as well as K-Means clustering and discretization, we first briefly introduce K-Means, and the discretization method used before outlining the details of our algorithms.

3.2.1. K-Means Clustering

K-Means was proposed in [37]. It is one of the efficient clustering algorithms and is used in many applications since it is simple to implement and has shown good performance. K-Means is categorized under the partitioning clustering algorithms, where the goal is to maximize the intra-cluster similarity, and minimize the inter-cluster similarity. K-Means is an efficient method as it runs in linear time. K-Means starts with specifying the number of clusters needed, K, and continues by following these steps (cluster example is shown in Figure 3.1):

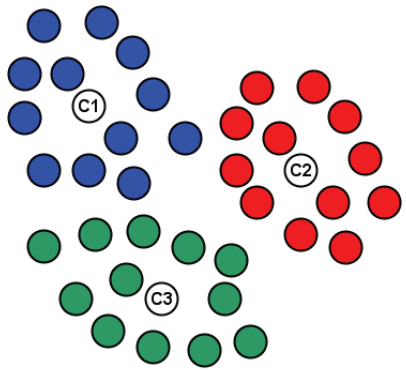


Figure 3.1: K-Means Clustering Example

- Step 1: Choose K initial random centroids, C_1, C_2, C_3...C_K.

- Step 2: Compute the distance from each point to every centroid, and then assign the point to the cluster with the minimum distance (closest one).

- Step 3: Compute the new means (centroids) of the generated clusters, and repeat this step until no changes occur on the clusters, or until a predefined number of iterations is reached. The distance between points and centroids can be computed using the Manhattan distance as follows:

$$Distance(P_i, C_j) = \sum_{v=1}^{D} |P_{iv} - C_{jv}| \tag{3.1}$$

where D is the dimension of point P_i; P_{iv} is the value of dimension v in point P_i; C_{jv} is the value of dimension v in center C_j.

To compute the new centroid the following equation is used:

$$C_j = \frac{1}{\mid n_j \mid} \sum_{i=1}^{n_j} P_i \tag{3.2}$$

where n_j is the number of points in cluster C_j.

3.2.2. Discretization

Discretization [38] is the process that discretizes and converts numeric attributes in the dataset into nominal attributes. There are two types of discretization, supervised and non-supervised. One of the supervised versions takes the class label into account when discretizing the numbers by calculating the entropy on the basis of the class label. It first sorts the attributes, and then finds the split points using the entropy [24]. This technique is used by our approaches. It finds the best split such that the segments are as pure as possible; thus, the majority of the values in a segment correspond to having the same class label. Formally, the goal is to find the split with the maximal information gain (Info (S)), which is obtained when the information value (Info(S, T)) is minimal. Therefore, given a set of samples S, partitioned into

two intervals S1 and S2 using boundary T, the entropy after partitioning is calculated as in Equation 3.3:

$$Info(S,T) = \frac{\mid S_1 \mid}{S} Info(S_1) + \frac{\mid S_2 \mid}{S} Info(S_2) \tag{3.3}$$

where $\mid * \mid$ denotes the cardinality. The boundary T is chosen from the midpoints of the attributes values, and the information content (Info) is calculated as follows:

$$Info(S) = P_1 * log_2(P_1) - P_2 * log_2(P_2) \tag{3.4}$$

where P_1 is the fraction of pairs within the first class, and P_2 is the fraction of pairs within the second class. The best splits are found by examining all possible splits and then selecting the optimal split points. The boundary that minimizes the entropy function over all possible boundaries is selected for the binary discretization. The process is recursively applied to partitions obtained until some stopping criterion is met, such as what is proposed by the Minimum Description Length (MDL):

$$Gain > \frac{log_2(N-1)}{N} + \frac{log_2(3^K - 2) - [KE - K_1E_1 - K_2E_2]}{N} \tag{3.5}$$

where K is the number of classes in S, K_1 is the number of classes in S_1, and K_2 is the number of classes in S_2, N is the number of instances, and E is the entropy of the instances, E_1 entropy of the instances in S_1, and E_2 is entropy of instances in S_2.

An example of the process is shown in Figure 3.2, where there are continuous numbers in the range from 24 to 60, and each number is related to a class label (in the second row). The discretization process uses the information gain by taking the class labels to find split points that have splits with pure class labels. After defining

the splits, majority voting is used to define the class label of the split, thus, in our example, we have class labels C2, C1, C2, C1, C2, C1, C2 respectively, for the splits shown in Figure 3.2.

24	25	38	39	40	51	54	58	60	71	73	77
C2	C2	C1	C1	C1	C2	C2	C1	C2	C1	C1	C2

Figure 3.2: Discretization Example

3.2.3. Proposed GP-K and GP-D

We propose two methods to achieve higher accuracy values for the classification of GP. The first method is based on using K-Means in combination with GP (GP-K), and the other is using GP together with the discretization process (GP-D). These techniques are used to transform the produced numeric output of the GP to a class label by storing the produced values from a program (classifier) in a data vector, and then applying the transformation process on it. In GP-K, the K-Means clustering algorithm is applied on the GP program output, then the accuracy or fitness of this program is computed using the number of records that are in the correct cluster. The process to compute the fitness of each program and to apply the program (equation) on the data records works as follows. The result of GP for each record (a number) is recorded and stored in a data vector. After that, K-Means is applied on this resulting data vector, and the number of K clusters is defined (that is the same as the number of classes in the original dataset). Majority voting is used on the resulting clusters to define the label of the cluster (class label). Then, after labeling the clusters, the records that are incorrectly appearing in the cluster are counted and represent the misclassified records by calculating the accuracy using Equation 3.6.

$$Accuracy = \frac{\#AllRecords - \#MisclassifiedRecords}{\#AllRecords} * 100\% \quad (3.6)$$

For GP-K, we found that the best number of iterations to run K-Means is 500, as it gives good accuracy values and does not affect the execution time too much (more iterations implies a larger execution time). However, K-Means may stop before reaching 500 iterations in case no changes occurred during consecutive iterations. GP-D completes the same process by first applying the GP program on the data records, and then recording the output number by storing it into a data vector. Afterwards, the supervised discretization is applied on this data vector. The discretization was applied using the one implemented by the WEKA data mining software [24], where a vector of the outputs and its class labels are sent to the discretization component. The result from the discretization is a range value instead of a numeric value. After that, all the records in the same range are collected, and then based on the class label for the maximum number of records in the range the class label for that range is defined. The accuracy is also calculated using Equation 3.6.

3.3. Experiments and Results

To evaluate the proposed techniques and see how effective they are in improving the accuracy of the GP, experiments were performed using the Java Genetic Algorithms Package (JGAP) [22] on two types of datasets, one with binary classes (true and false), and the other with multiple classes (multiclass), same shown in Table 2.2 and Table 2.3.

3.3.1. Experimental Settings

The experiments were conducted as follows. First of all, feature selection was performed on the datasets using the WEKA software [24], which reduced the number of features as shown in the brackets in Table 2.2 for the binary datasets, and in Table 2.3 for the multiclass datasets, respectively. The attribute selection method in WEKA

is a supervised attribute filter that allows various search and evaluation methods to be combined [24]. Secondly, our GP algorithm was run using the following settings: population size = 500, number of generations = 1000, crossover probability = 0.5, mutation probability = 0.1, maximum initial depth = 5, maximum crossover depth = 8, function probability = 8, dynamize arity probability = 0.05, new chromosome percentage = 0.2. 37 functions are used, and 14 of them are variables and constants. The experiments are applied on the standard GP (GP), the proposed GP-K and GP-D using 66% of the datasets for training and the remaining 34% for testing.

3.3.2. Results

To evaluate the results of the proposed GP algorithms, they are compared with standard GP measuring the accuracy and execution time. Moreover, a comparison with state-of-the-art classifiers was done to measure how well our algorithms compared to the ones known in the literature. Table 3.1 shows the average with standard deviation for 30 runs, as well as the best accuracy results of the standard GP, and the proposed approaches GP-K and GP-D applied to the binary datasets for the training and testing phases. We can infer from Table III that GP-K and GP-D show improvements compared to the accuracy of GP. For D1 dataset, the GP obtained a testing accuracy of 92.85%, whereas GP-K and GP-D achieved 95.39% and 97.54%, respectively. Similarly, for the D2, D3 and D4 datasets, GP-K showed improvements over the original GP, whereby GP-D scored best.

Looking at the training accuracy, for D1 dataset, GP obtained an accuracy of 95.85%, with the best value (from the 30 runs) of 97.86%. However, GP-K's best value is 98.66%, GP-D's average is 98.66%, and its best value is 99.20%. The same trend is found for the other binary datasets, where GP-K slightly improves GP's accuracy (3%-4%), and GP-D outperforms both GP and GP-K by 5%-8% compared to GP. It can also be seen that both GP-K and GP-D have a smaller standard deviation

compared to GP, which indicates that they are more robust.

Table 3.1: Accuracy For Binary Datasets

Datasets		D1		D2		D3		D4	
		Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
GP	**Avg**	95.85	92.85	76.02	75.84	86.65	73.04	87.81	82.52
	Std	±1.07	±1.90	±0.99	±1.25	±1.91	±4.19	±1.72	±2.88
	Best	97.86	95.88	77.87	78.63	89.27	81.52	91.18	86.79
GP-K	**Avg**	97.76	95.39	77.52	78.34	88.93	76.34	88.20	86.03
	Std	±0.58	±0.86	±0.29	±0.82	±0.51	±3.38	±0.83	±2.35
	Best	98.66	96.91	78.26	80.15	90.40	81.52	90.20	90.57
GP-D	**Avg**	98.66	**97.5**4	78.46	**80.07**	89.94	**80.50**	94.05	**88.55**
	Std	±0.16	±0.48	±0.62	±1.00	±0.43	±2.56	±0.88	±1.10
	Best	99.20	98.45	79.84	82.44	90.96	85.87	96.08	90.57

The accuracy results for the multiclass datasets are shown in Table 3.2. We can see that GP-D achieves the highest accuracy for all datasets, and GP-K improves the accuracy of standard GP. For example, for D6 dataset, GP's accuracy is 70.98%, GP-K's is 74.21% and GP-D's is 86.22%. For D7, GP's accuracy is 72.32%, GP-K's is 79.46% and GP-D's is 86.24%. The maximum improvement that was achieved was for the CTG-class dataset, where GP's accuracy is 44.74%, GP-K's is 61.03% (increased by 16%), and GP-D's is 64.09% (increased by 19% compared to GP's accuracy). Looking at the best results accomplished while testing the datasets, the accuracy values obtained on D5 are 82.35%, 80.39%, 84.31% for GP, GP-K, and GP-D, respectively. However, the difference for D6 is larger where 82.90%, 86.91% and 90.69% were measured. The difference between the best accuracy values of the GP, GP-K and GP-D for D7 and D8 are smaller (close to what is shown for D5), while

for D9, GP's best value is 52.97%, GP-K is 65.98%, GP-D is 73.17%. Moreover, commenting on the standard deviation of the testing process, the values for D5 dataset are ± 4.45, ±2.94, and ±2.15 for GP, GP-D and GP-K, respectively. For the other datasets, the values for GP, GP-K and GP-D are in the same range, besides for the binary datasets, where the values are either smaller or no change is observed.

For the training phase, the results show the same trend as for the testing phase comparing GP, GP-K and GP-D. However, they have higher values compared to the testing phase. GP achieved average accuracy results of 90.72%, 82.90%, 87.14%, 87.24%, 54.10% on the D5, D6, D7, D8, and D9 datasets, respectively. However, GP-K obtained 88.66%, 87.45%, 95.44%, 86.60% and 69.43%. GP-D on the other hand achieved scores of 89.97%, 91.24%, 97.28%, 90.12% and 73.50%. In addition, the best values are much higher, where for the same order of datasets GP-D obtained 91.75%, 92.35%, 99.17%, 91.17%, and 76.41%.

The running time results are shown in Figure 3.3 (drawn using the logarithmic scale to adapt to the large difference between small and large datasets), where we can see that GP-K has a slightly longer running time than GP, while GP-D's running time is much longer compared to GP. For example, for D1 (breast cancer) GP took 55.70 seconds, GP-K took 60.44 seconds, and GP-D took 367.80 seconds. For the binary datasets, and D5 and D7 (Lymph and Dermatology), the running times are similar to D1 (breast cancer), whereas for D6, D8, and D9 (Splice, CTG-NSP and CTG-Class) the difference of GP-D is much larger. The difference in running time increases when the number of records gets larger (compare the number of records of Splice: 3190 with CTG: 2126).

Table 3.2: Accuracy For Multiclass Datasets

Datasets		D5		D6		D7		D8		D9	
		Train	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
GP	**Avg**	83.71	73.07	71.59	70.98	74.43	72.32	84.38	83.53	47.56	44.74
	Std	±3.13	±4.45	±4.10	±4.19	±5.13	±5.29	±0.99	±1.83	±3.40	±4.81
	Best	90.72	82.35	82.30	82.90	87.14	82.40	87.24	87.83	54.10	52.97
GP-K	**Avg**	86.84	76.01	75.60	74.21	91.58	79.46	85.03	83.49	65.59	61.03
	Std	±1.10	±2.94	±3.48	±4.59	±2.11	±4.05	±0.72	±1.19	±1.41	±2.79
	Best	88.66	80.39	87.45	86.91	95.44	86.40	86.60	86.86	69.43	65.98
GP-D	**Avg**	89.97	**78.36**	91.24	**86.22**	97.28	**86.24**	90.12	**88.33**	73.50	**64.09**
	Std	±1.43	±2.15	±0.78	±4.35	±1.05	±4.64	±0.65	±1.05	±1.40	±5.06
	Best	91.75	84.31	92.35	90.69	99.17	96.00	91.17	90.32	76.41	73.17

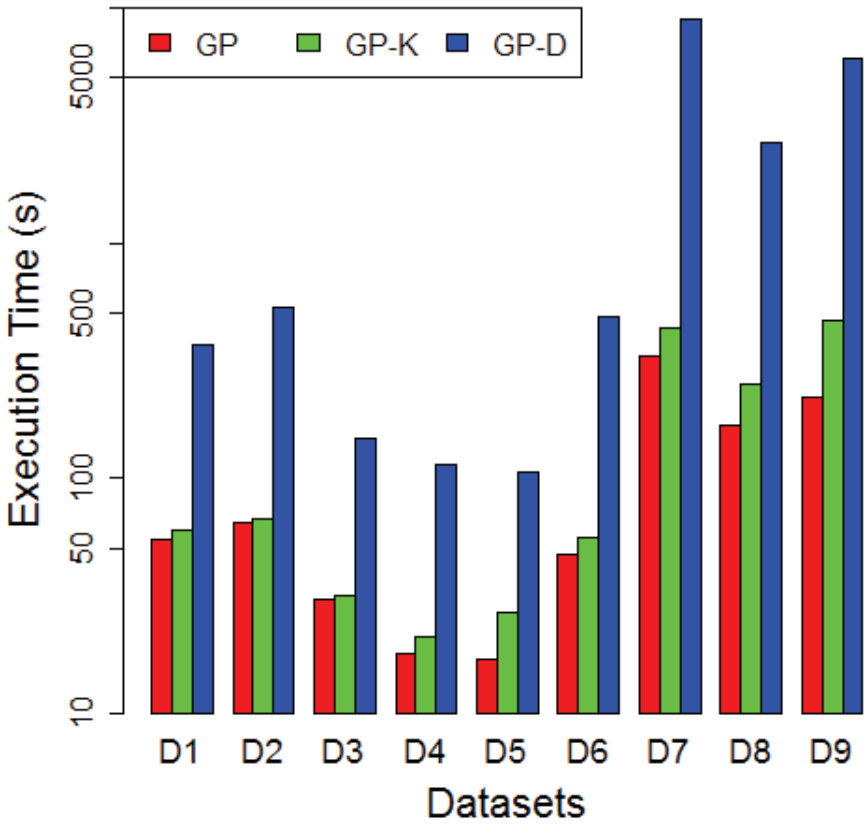


Figure 3.3: Running Time Results For GP, GP-K and GP-D

The reason for this longer running time of GP-D, is that GP-D uses the discretization method as stated in [24] by calculating the information gain for each possible split point. If the dataset has for example 2126 records, then there are 2126 possible split points. In addition, this discretization process is done for every program (classifier) when its fitness is calculated, therefore, for a population size of 500, the discretization portion is executed 500 times per generation. Although GP-D has a longer running time compared to GP and GP-K, however, a substantial improvement in accuracy is achieved, as well as the robustness of the algorithm is increased. However, if time is of essence, GP-K can be used since it has a higher accuracy than standard GP with a comparable running time.

A comparison of the accuracy of GP, GP-K and GP-D with other well-known classification methods on the same datasets is shown in Tables V and VI, where the numbers in bold are the highest values, and the values in bold and italic are the second highest values. The following classifiers were used and are described below:

- Bayes Network (BN): learning algorithm using various search algorithms and quality measures.
- Naive Bayes (NB): class for a classifier using estimator classes.
- J48: contains a class for generating a pruned or un-pruned C4.5 decision tree.
- IBK: K-nearest neighbors classifier, which can select an appropriate value of K based on cross-validation and also performs distance weighting.
- MP: multilayer perceptron is a classifier that uses back propagation to classify instances.
- K*: is an instance-based classifier, that is, the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function.
- JRip: implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER).
- SMO: implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.

All these algorithms were applied using the WEKA data mining software [24] with 10 fold cross-validation. The binary dataset results are shown in Table 3.3. GP-D outperforms all other classifiers with an average accuracy of 97.54% for the Breast dataset. However, for the Diabetes dataset, NB has the highest accuracy of

84.38%, whereas standard GP has 75.84%, GP-K and GP-D have average accuracy values of 78.34% and 80.07%, respectively, outperforming the K* and JRip results. For the Heart dataset, GP-D achieves the highest average accuracy of 80.50%, and the best accuracy of 85.87%. GP-K obtains an accuracy of 76.34% and outperforms a number of other classifiers such as J48, BN, IBK, K*, JRip, and MP. For the Hepatitis dataset, GP-D also achieves the highest accuracy of 88.55% with the best value of 90.57%, again outperforming all other classifiers. Moreover, GP-K also outperforms all other classifiers with an accuracy of 86.03% (best value of 90.57%). In summary, GP-D obtains the highest accuracy values for the binary datasets in 3 out of 4 cases, whereby GP-K has the highest average accuracy value on the Hepatitis dataset, as well as outperforming some of the other classifiers.

For the multiclass datasets as shown in Table 3.4, GP-D shows competitive values compared to the other classifiers, such as the Lymph dataset, where GP's best accuracy is 84.31%. Also, for the Dermatology and CTG-NSP datasets, GP-D's best value is 96.00%, which is somewhat close to the best result of 97.81% for Dermatology, and 90.32% (best 92.94%) for CTG-NSP. GP-K did not achieve close scores as GP-D did; however, it definitely improved the standard GP. A statistical two-tailed Z-Test was applied on the average results of the 30 runs for GP-K compared to standard GP as well as for GP-D compared with standard GP using a significance level of 5%. The Z-Test results show that GP-K and GP-D achieve significant accuracy improvements on all datasets except for GP-K when applied on CTG-NSP.

Table 3.3: Comparing Accuracy With Other Classifiers - Binary Dataset

Datasets	**GP**	**GP-K**	**GP-D**	**J48**	**BN**	**NB**	**IBK**	**K***	**JRip**	**SMO**	**MP**
Breast	92.85 (95.88)	95.39 (96.91)	**97.54 (98.45)**	93.48	95.95	94.54	95.24	95.59	95.07	96.65	97.00
Diabetes	75.84 (78.63)	78.34 (80.15)	80.07 (82.44)	81.41	83.27	**84.38**	80.66	78.43	78.81	84.01	81.41
Heart	73.04 (81.52)	76.34 **(81.52)**	**80.50 (85.87)**	74.86	75.52	77.47	68.35	69.92	74.73	76.82	75.52
Hepatitis	82.52 (86.79)	**86.03 (90.57)**	**88.55 (90.57)**	81.93	83.22	85.16	81.29	83.87	78.70	83.22	81.29

Table 3.4: Comparing Accuracy With Other Classifiers - Multiclass Dataset

Datasets	**GP**	**GP-K**	**GP-D**	**J48**	**BN**	**NB**	**IBK**	**K***	**JRip**	**SMO**	**MP**
Lymph	73.07(82.35)	76.01 (80.39)	78.36 **(84.31)**	78.37	79.72	80.40	77.70	81.75	77.02	**81.08**	79.72
Splice	70.98 (82.90)	74.21 (86.91)	86.22 (90.69)	93.54	**96.14**	92.06	68.24	77.86	94.07	85.20	89.78
Dermatology	72.32 (82.40)	79.46 (86.40)	86.24 **(96.00)**	95.35	**97.81**	**97.81**	96.72	97.54	94.26	**97.81**	96.72
CTG-NSP	83.53 (87.83)	83.49 (86.86)	88.33 **(90.32)**	**92.94**	89.46	84.43	92.09	91.90	91.95	88.19	89.51
CTG-Class	44.74 (52.97)	61.03 (65.98)	64.09 (73.17)	**82.83**	78.50	71.49	80.66	81.93	82.07	76.81	79.82

3.4. Summary

This chapter proposed two approaches in order to improve the accuracy of GP for the classification task, in particular for multiclass classification problems. The first proposed approach, GP-K, makes use of the K-Means clustering algorithm to transform the GP output from a numeric value to a class label. The second approach, GP-D, performs the same transformation process by applying the discretization method on every classifier.

A comparison of the standard GP with the proposed approaches was conducted using binary and multiclass datasets measuring the accuracy and running time. Overall, it was shown that both, GP-K and GP-D, improve the accuracy of standard GP, however, this comes at the cost of a longer running time. GP-K only has a slightly longer running time, but GP-D takes a very long time to run. In addition, for the binary datasets, GP-K and GP-D were found to be more robust compared to standard GP as shown by the standard deviation. Comparing the proposed approaches with other known classifiers, it was found that GP-D outperforms 3 of 4 classifiers for the binary datasets, and for the multiclass higher accuracy values than GP were achieved with some competitive values compared to the known classifiers.

Given that both GP-K and GP-D have longer running times than GP, preliminary tests were run keeping the running time for standard GP the same as GP-D in order to see the effect on the accuracy. The tests were applied on the Breast dataset with 30 runs. GP was run for 6,000 iterations instead of 1,000 (running time on average was 373.02 seconds close to 367.8 seconds for GP-D). The average accuracy of GP improved from 92.85% to 94.36%. GP-D has an average accuracy of 97.54%, which is still much better than standard GP. However, more experiments are necessary to confirm the same trend on all other datasets.

CHAPTER 4. ACCELERATING THE FITNESS EVALUATION OF GENETIC PROGRAMMING

GP uses evolutionary operations to search for the best classification model by applying the fitness evaluation. The fitness evaluation process greatly impacts the overall execution time of GP and is therefore the focus of this research study. One of the main factors affecting the fitness evaluation is the dataset size. In this chapter, our goal is to accelerate the fitness evaluation process by decreasing the training dataset size but at the same time covering the whole training dataset. Thus, a segment-based GP (SegGP) technique is proposed that reduces the execution time of GP by partitioning the dataset into segments, and using the segments in the fitness evaluation process. Experiments were done by applying SegGP on 14 datasets to measure whether the same classification accuracy compared to standard GP can be obtained, however, with a shorter execution time. Different sizes of segments were created to measure the effect of the segment size on the accuracy as well as the execution time. The results show that SegGP can obtain higher or similar accuracy results in shorter execution time compared to standard GP.

The rest of this chapter is structured as follows: Section 4.1 describes related work in the area of accelerating the run time of GP. In Section 4.2, our proposed segment-based GP (SegGP) approach is explained. Section 4.3 presents the experiments as well as the results obtained, and Section 4.4 summarizes this chapter.

4.1. Related Work

Several techniques have been proposed to speed up the execution time of GP. Some research investigations applied parallel GP in different ways such as using GPUs [15], OpenCL [39], and even parallel GP on one processor with a demetic approach [40], while others focused on the acceleration of the GP internal steps.

In [40], the authors introduced a variant of linear GP that uses sequences of instructions of an imperative programming language. It is based on eliminating the non-effective code of the programs, which are called introns. The intron removal is done during the fitness evaluation stage before the program is executed, therefore, the program size is smaller and the execution time is faster. An algorithm was used to detect the introns and remove them from each program. A comparison with the artificial neural networks was done and the results show that the accuracy of the classification is not affected by the intron removal, furthermore, a good speedup is achieved. This approach was done on linear GP since it is difficult to be applied on a tree-based GP, especially the intron detection and elimination processes. Also, having a small program size speeds up the process, but the computation overhead of the intron detection and elimination process for each program in the population cannot be ignored.

Regarding the tree representation of GP programs, [18] and [41] proposed approaches to accelerate the GP run time. In [18], a GP method called GPTM (GP with Tree Mining) was proposed. The method first identifies the sub-trees repeatedly appearing in the chromosomes of superior individuals (ones with very high fitness), and then protects them from undesirable crossover operations. To find such sub-trees, GPTM uses a FREQT-like efficient tree-mining algorithm. GPTM was evaluated on three benchmark problems, and the results indicate that GPTM finds optimal individuals earlier. The tree-mining method is a good idea since all solutions in GP are represented as trees; however, mining every tree is computationally very expensive especially for large population sizes. Furthermore, [41] presents two new tree-generation algorithms for genetic programming. These algorithms are fast (running in near-linear time, or in linear time under reasonable constraints), and allow the user to request specific tree sizes, and guarantee probabilities of certain nodes appearing

in trees.

The authors in [42] propose a learning strategy for linear GP referred to as Hierarchical Structure GP (HSGP). HSGP solves a problem by using multiple learning nodes connected in a hierarchical structure using a binary-tree structure consisting of 3 levels. HSGP starts from the bottom-level where each node generates its own initial population, and executes conventional evolutionary processes to evolve the population for a fixed number of generations using different and non-overlapping subsets. Then, each node of the next higher-level receives the evolved population from the connected lower-level nodes and integrates them into a new population, that is then used as the initial population of these nodes. Also, the training subsets of the connected lower-level nodes are integrated together and used as the training subset of the higher-level node. After the learning at the top-level node (with the entire training set) is finished, the best individual obtained by the top-level node is considered the best solution for the problem. HSGP was tested to build an evolutionary system for synthesis of image recognition programs, where it was found that HSGP outperforms the number of program executions of conventional GP. HSGP makes use of the divide and conquer methodology, and provides the ability to use more initial individuals. However, dividing the dataset in the lower level and then recombining it at each node of the upper level, this means that each level will execute the complete dataset, thus, having a binary tree with three levels means that the dataset will be executed three times. Therefore, if it uses a larger dataset, or if the binary tree was larger, then the acceleration results may not be the same. Also, the comparison was based only on the number of program execution, but no execution time was reported to measure the actual GP run time.

The use of a subset of the training data as a part of an object detection system had been proven to be a good way to improve the efficiency and effectiveness of the

GP approach for object detection tasks. In [43], the authors broke the GP search into two phases; the first phase applies GP to a selected subset of the training data using a simplified fitness function. This fitness function maximizes the classification accuracy on the object cutouts, which is simple and easy to evaluate, and finds good genetic programs much more rapidly and effectively than using the full set of training data (like in the second phase). The second phase is then initialized with the resulting individuals from the first phase using the full set of training data with a complete fitness function to construct the final detection programs. It uses a fitness function that maximizes the detection performance on the large images in the training set. In the test procedure, the best refined genetic program is applied to the entire images in the test set to measure the object detection performance. A comparison of this two-phase GP approach with the basic GP approach and the neural network approach was done using the same set of features on three object detection problems of increasing difficulty. The results show that their two-phase GP approach achieved the best detection performance. Multi-phase approaches in GP were also proposed in [44] and [45].

A Dynamic Subset Selection (DSS) method was proposed in [46] to reduce the number of fitness evaluations performed in the GP process. DSS creates a subset from the training dataset to be used in the fitness evaluation step in GP. The subset is randomly selected based on the records' difficulty and age. Records' difficulty is defined as the number of GP programs misclassified it, and records' age is defined as the number of generations the record has not been selected. In this way, DSS ensures that all the training set will be covered, and that the GP program will focus on the difficult records. DSS proved its effectiveness with producing equivalent answers with less number of fitness evaluations. However, the real execution time was not reported, which is an important measure, since DSS needs two passes of the whole training

dataset to create the subset for each generation which takes time. The authors also introduced a Historical Subset Selection (HSS) to compare with. HSS is based on the difficulty of the records based on previous GP runs. The experimental results showed that HSS out-performed the standard GP in terms of number of fitness evaluations, and nearly matched it in terms of quality of results.

The method proposed in this chapter is different from those explained above, in that it accelerates the fitness evaluation step using segments of the training dataset instead of the whole training dataset and at the same time covers the data without the need to scan the whole training dataset at a time. It is based on partitioning the training dataset into ten smaller segments during the pre-processing stage, and then for each fitness evaluation of a program, a subset is chosen and the program fitness is calculated. In the testing phase, no segmentation is done, and the best program found through the training stage is applied on the whole testing dataset to calculate the best accuracy found by GP. Our method is different than DSS in that it covers the whole training dataset at every generation. Also, there is no need to do passes of the training dataset to create the subset, thus, the execution time and the number of fitness evaluations both are accelerating the GP process.

4.2. Proposed Approach

When GP is applied to solve a classification task, the fitness evaluation step refers to the program execution and calculation of its classification accuracy. The classification accuracy is calculated after the program is executed on every record of the dataset counting the number of correctly classified records. Classification accuracy is the percentage of number of correctly classified records to the complete dataset size. Therefore, the number of fitness evaluations (FE) of an independent GP run is related to number of programs, which defines the population size (P), number of records in the dataset (dataset size) (D), and number of generations (G). For each program

the fitness calculation is equal to D since the program is executed for every record in the dataset. Thus, the total number of fitness evaluations for a GP run is $FE = D \times P \times G$.

Our proposed SegGP approach accelerates the GP run time by focusing on the fitness evaluation time, and especially by reducing the dataset size D [47]. It is based on creating segments of the training dataset, during the pre-processing stage, and then feeding these segments into the GP run as shown in Figure 4.1. Thus, whenever GP performs a fitness evaluation for a program it randomly chooses one of these segments and executes the program to obtain the fitness. Each program may choose different segments, and as a population it uses the whole training dataset. The training dataset is partitioned into ten segments. Size of the segments (S) is a percentage of the training dataset, where $S < D$. Using this technique, the number of fitness evaluations is reduced to $FE = S \times P \times G$.

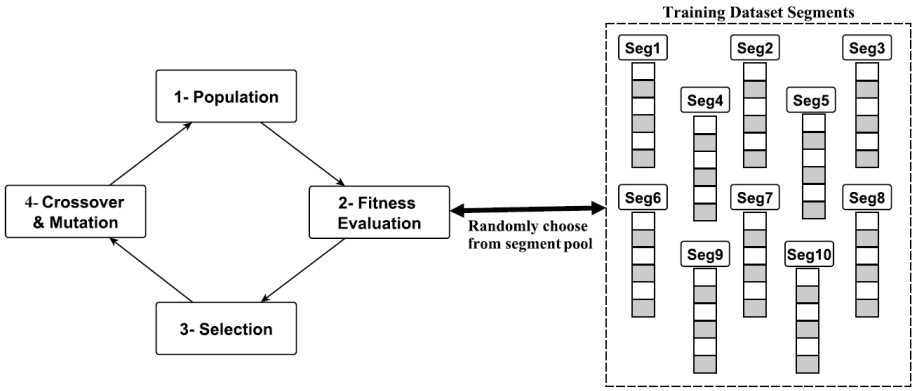


Figure 4.1: Segment-based GP (SegGP) Steps

Creating the data segments is done using the resample supervised instance filtering technique of the Weka data mining software [24]. The resample technique produces a random sub-sample of a dataset using either sampling with replacement or sampling without replacement. For the SegGP approach, we used the resample without replacement technique. Moreover, we specify the size of the segment as

percent of the training dataset, such as for a training dataset consisting of 1,000 records and a 10% segment size, therefore, the segment size consists of 100 records. Figure 4.2 shows an example of how the segments are created. For simplicity, this example shows each segment to be of 100 consecutive records, however, each segment consists of randomly chosen records. For this example, assuming that we have a population size of 500, and the number of generations is equal to 500, the total number of fitness evaluations for SegGP approach is $FE = 500 * 500 * 100 = 25 * 10^6$, while if we use the whole training dataset it is $FE = 500 * 500 * 1000 = 250 * 10^6$.

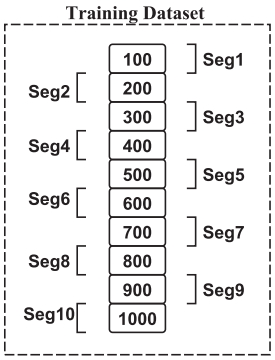


Figure 4.2: Segments Creation Example

The SegGP approach differs from related work that uses only a single subset not covering the whole dataset, as opposed to the SegGP that uses the whole training dataset applying only a random segment of the training dataset for each fitness evaluation. Therefore, the population is likely to cover the whole dataset. In addition, if a program survives for multiple generations, it is likely to cover the whole dataset by using different segments in each generation, and thus, the program is trained on the whole dataset (for both difficult and easy records). There is a high probability that this occurs with segments of large percentage of the whole dataset. For example, if the segment size is 50%, the program could cover the whole dataset in two generations. For the testing phase, which is performed once at the end of the GP run, it is important to note that the testing dataset is used in its entirety without

any partitioning. Hence, the best program (solution) resulting from the GP run is tested using the whole testing dataset.

4.3. Experiments and Results

To evaluate the proposed SegGP approach and to see how effective the acceleration of the GP run is without affecting the classification accuracy, experiments were performed by programming the Java Genetic Algorithms Package (JGAP) [22] to implement SegGP and standard GP.

In order to evaluate SegGP and to find the best segment size to use, we ran SegGP with different segments sizes, starting with a segment size of only 10% of the training dataset, up to using the whole dataset (100%) in steps of 10% increments. To simplify the names of the segments, the segment name is based on the percentage of the whole training dataset. For example, segment "Seg10" consists of 10% of the whole training dataset. The same format is used for all segments (Seg20, Seg30, Seg40 ... Seg90, Seg100), where "Seg100" is the whole training data.

The proposed SegGP approach is compared with standard GP that uses the whole training dataset. The experiments measure the accuracy, execution time, sensitivity, and specificity. In addition, the statistical T-test is applied to test the significance of the results. Moreover, an analysis of the used datasets was done to study the impact of the dataset structure on the classification results of GP.

4.3.1. Experimental Settings

The experiments were conducted as follows; in the pre-processing stage of the datasets, a feature selection process using the WEKA software [24] was performed that reduced the number of features as shown in brackets in Table 5.1. The attribute selection method in WEKA is a supervised attribute filter that allows various search and evaluation methods to be combined [24]. After the preprocessing the GP run was performed on the datasets with the reduced features. However, given the stochastic

nature of GP, 50 independent runs were performed in order to compare our proposed SegGP with standard GP. The GP settings for each run were as follows:

- Population size = 500
- Number of generations = 1000
- Crossover probability = 0.5
- Mutation probability = 0.1
- Maximum initial depth = 5
- Maximum crossover depth = 8
- Function probability = 8
- Dynamize arity probability = 0.05
- New chromosome percentage = 0.2
- Functions: 23 mathematical and logical functions
- Terminals: 14 consisting of variables and constants

We would like to mention that the results for GP are not expected to rank among the top compared to other classification approaches since the parameter settings have not been adjusted for each dataset individually.

The experiments are applied on two types of datasets [23], binary (two classes) and multiclass (3 classes and more), using 66% of the dataset for training, and the remaining 34% for testing.

- Wisconsin Diagnostic Breast Cancer (WDBC) that predicts breast cancer diagnoses of benign and malignant.

- Hepatitis dataset contains data of whether a person lived or died.
- Heart datasets refers to the presence of heart disease in patients.
- Diabetes is the Pima Indians Diabetes Database, which diagnoses whether the patient shows signs of diabetes or not.
- Ionosphere dataset is the classification of radar returns from the ionosphere, the class label is either "Good", which means that radar returns are those showing evidence of some type of structure in the ionosphere, or "Bad", which means the returns are those that do not (their signals pass through the ionosphere).
- Vertebral Column data set (Vertebral-3C and Vertebral-2C) contains values for six biomechanical features used to classify orthopaedic patients into 3 classes (normal, disk hernia or spondilolysthesis), or 2 classes (normal or abnormal).
- Blood Transfusion Service Center: Data taken from the Blood Transfusion Service Center in the Hsin-Chu City of Taiwan. Class labels represent whether or not the person donated blood in March 2007.
- Dermatology dataset, which determines the type of Eryhemato-Squamous disease.
- Lymph dataset, which is about the Lymphography disease.
- Splice dataset contains Primate splice-junction gene sequences (DNA) with associated imperfect domain theory.
- CTG dataset consists of fetal Cardiotocograms (CTG) records. The CTG dataset has two outcome categories, the first classifies a morphologic pattern (CTG-Class), and the second is related to a fetal state (CTG-NSP).

- Balance Scale dataset is generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced.

All details including the number of features and records of these datasets are shown in Table 4.1.

Table 4.1: Binary Datasets (D1-D7) And Multiclass Datasets (D8-D14)

	Dataset	**Classes**	**Features**	**Records**
D1	Breast	2	31 (11)	568
D2	Diabetes	2	8 (4)	768
D3	Heart	2	13 (9)	269
D4	Hepatitis	2	19 (10)	155
D5	Vertebral-2C	2	6 (6)	310
D6	Blood	2	5 (4)	748
D7	Iono	2	34 (14)	351
D8	Lymph	4	19 (10)	148
D9	Splice	3	61 (22)	3190
D10	Dermatology	6	33 (19)	366
D11	CTG_NSP	3	22 (7)	2126
D12	CTG_Class	10	22 (11)	2126
D13	Balance	3	4 (4)	625
D14	Vertebral-3C	3	6 (6)	310

4.3.2. Accuracy Results

The accuracy results for the 14 datasets using different segment sizes are shown using the box plots in Figure 4.3 and Figure 4.4, where the circle represents the average of 50 runs, the solid bar depicts the mean, the lower whisker represents the minimum accuracy observed by the GP for a given segment size, and the upper whisker depicts the maximum accuracy value.

For binary datasets and starting with dataset D1, the average of the accuracy results using the whole training set (Seg100) is 93.41%, whereas using Seg10 the accuracy is 88.75%. Seg60, Seg70, and Seg90 have close accuracy values compared

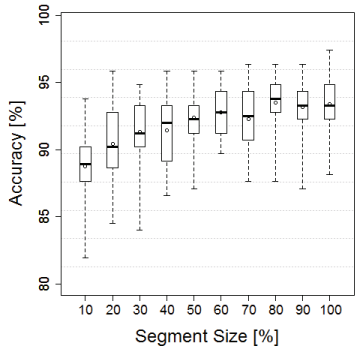
to Seg100 with values of 92.77%, 92.29% and 93.18% respectively. Seg80 has the highest accuracy of all segments, with an accuracy of 93.53%. D2's average accuracy result for Seg10 is 73.45%, while for Seg100 it is 75.50%. The best results are found using Seg60 and Seg80, which are 76.23% and 75.77% respectively. The best average accuracy result for D3 is using Seg100 with an accuracy value of 73.78%; while Seg10, Seg20, and Seg30 all have accuracy values of 66.15%, 68.54% and 68.45%, respectively. However, all other segments from Seg40 to Seg90 have accuracy values close to Seg100 ranging from 72.34% to 72.71%. The accuracy results for D4 show that the average accuracy of Seg10 is 80.64%, while using the whole dataset (Seg100) it is 83.47%. The best accuracy results are obtained for Seg80 and Seg90 with values 84.75% and 84.83%, respectively. The highest accuracy for D5 is using Seg100 with 76.42%, where Seg90 has a close accuracy 75.83%. Seg40 to Seg70 all have accuracy values in the range 74.63% and 74.65%. D6 lowest accuracy is using Seg10 with 71.17%, however the best accuracy is using Seg100 with 74.93%. Seg70, Seg80 and Seg90 all have accuracy values close to the best and in the range between 74.31% and 74.88%. Seg100 accuracy value for D7 is 92.74%, which is the best compared to other segments, while Seg80 and Seg90 accuracy values are 91.72% and 92.42, respectively. The lowest accuracy is using Seg10 with 81.62%.

For the multiclass datasets, the lowest accuracy is also using Seg10. However, D8's best average accuracy result is for Seg90, which is 73.41% that is slightly better than Seg100 (72.9%). For segments Seg10 to Seg50, all accuracy values are in the range between 60.54% and 69.15%. Seg80 and Seg100, have the highest accuracy values for D9 (70.73% and 70.97%), where Seg10 has the lowest value of 67.73%. Moreover, Seg60, Seg70, and Seg90 have very close accuracy values compared to the best with 70.14%, 70.26% and 70.12%, respectively. For D10, Seg100's accuracy is 69.55%, while Seg80 has an accuracy of 70.68%. The worst accuracy value is measured

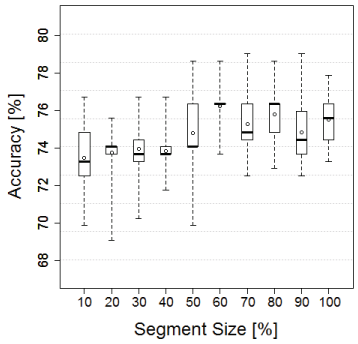
for Seg10 (60.4%). D11 has three high accuracy values of 83.28%, 83.25%, and 83.17% for Seg100, Seg80, and Seg90 respectively, while for Seg10 the accuracy is 79.42%. For D12, Seg100 has an accuracy of 44.66%, where for Seg70 and Seg90 both have close accuracy values of 44.00% and 44.08%, respectively. Seg60 has the highest value with 45.24% and Seg10 has the lowest accuracy with 41.66%. D13 dataset highest accuracy is using Seg100 with 80.55%, while Seg80 and Seg90 both have close accuracy values with 78.48% and 78.02%. Seg10 accuracy is 68.92%, which is the lowest accuracy. The highest accuracy for D14 is using Seg100 72.55%, where Seg90 accuracy is very close 71.32%. And the lowest accuracy is using Seg10 with value 56.37%.

In summary, comparing the accuracy results with the ones using the whole dataset (Seg100), we can conclude that Seg90 has two accuracy values higher than Seg100 for D4 and D8. Seg80 has four accuracy values higher than Seg100 for D1, D2, D4, and D10. Moreover, Seg60 has three values higher than Seg100 for D2, D10 and D12. In addition to the higher values, looking for accuracy values that are similar or slightly lower (in fractions) to the highest average accuracy, Seg90 has 9 close values, for binary datasets (D1, D3, D5, D6 and D7) and for multiclass (D9, D11, D12 and D14).

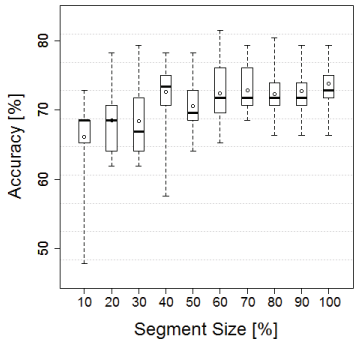
Seg80 has 7 close values, for binary datasets (D2, D4, D6 and D7) and for the multiclass datasets (D9, D11, and D13). However, Seg70 has 6 accuracy values close to the best for D2, D3, D6, D8, D9 and D12. And at last, Seg60 has seven accuracy values to the best, for D1, D4, D6 and D7 in the binary datasets, and D9, D10, and D11 for the multiclass datasets.



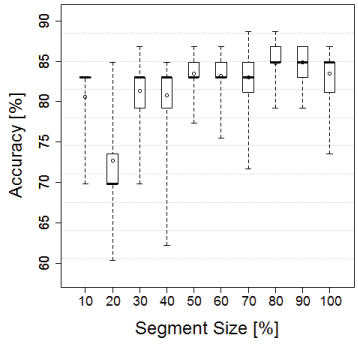
(a) D1



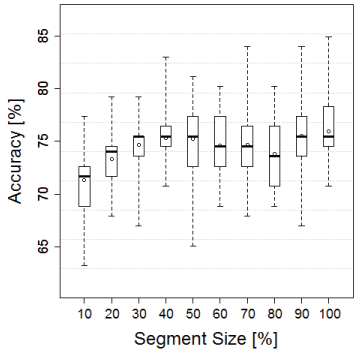
(b) D2



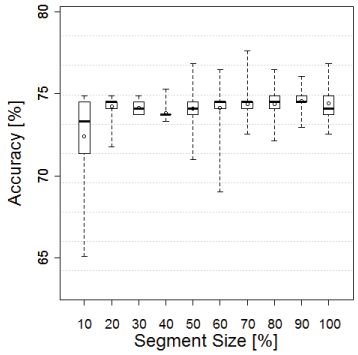
(c) D3



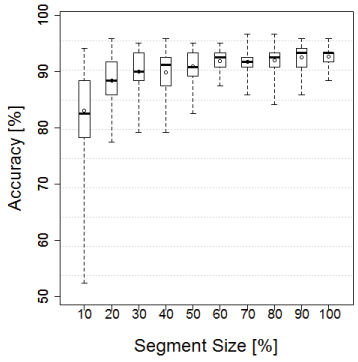
(d) D4



(e) D5

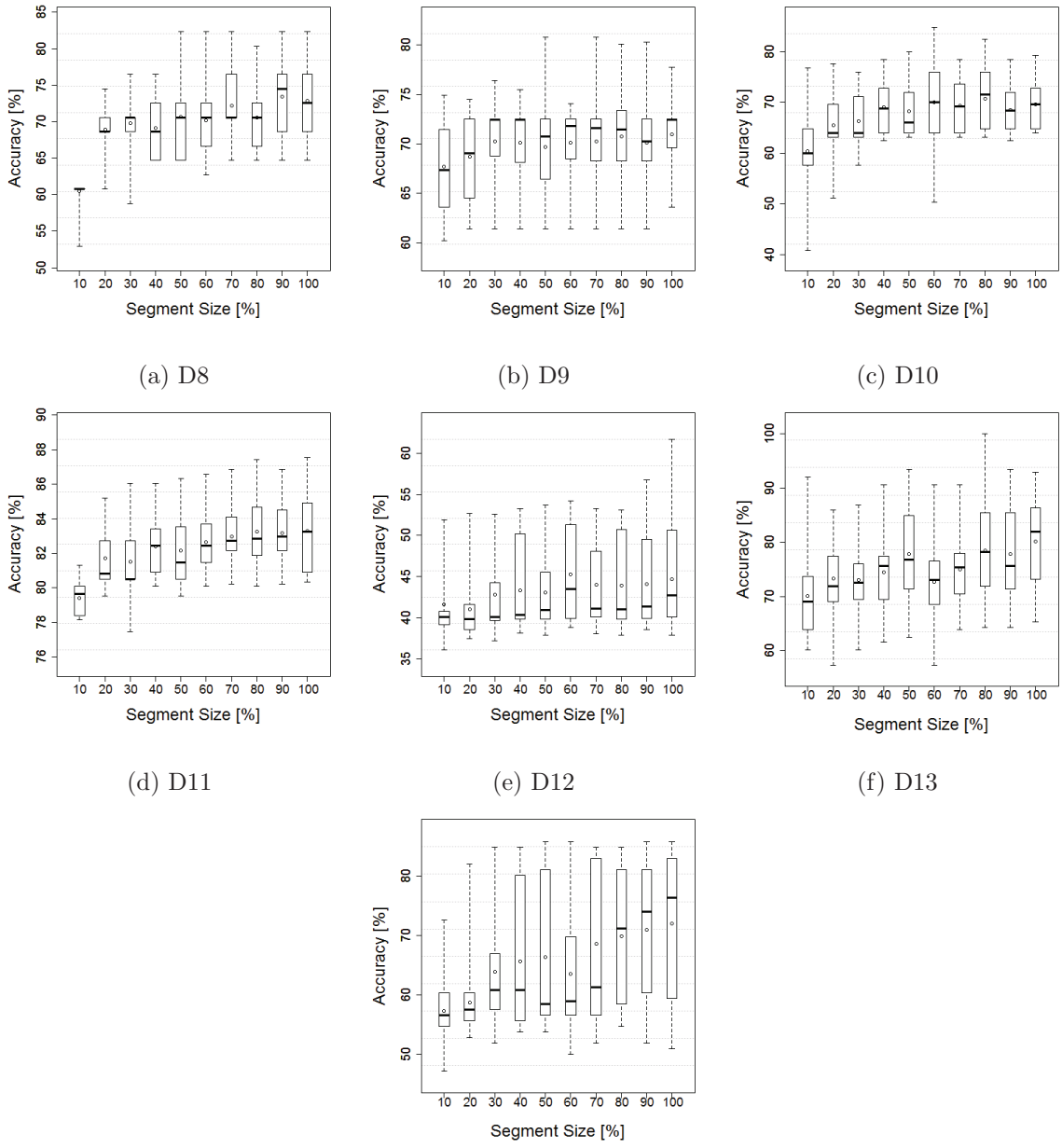


(f) D6



(g) D7

Figure 4.3: Accuracy Results For Binary Datasets



(a) D8 (b) D9 (c) D10

(d) D11 (e) D12 (f) D13

(g) D14

Figure 4.4: Accuracy Results For Multiclass Datasets

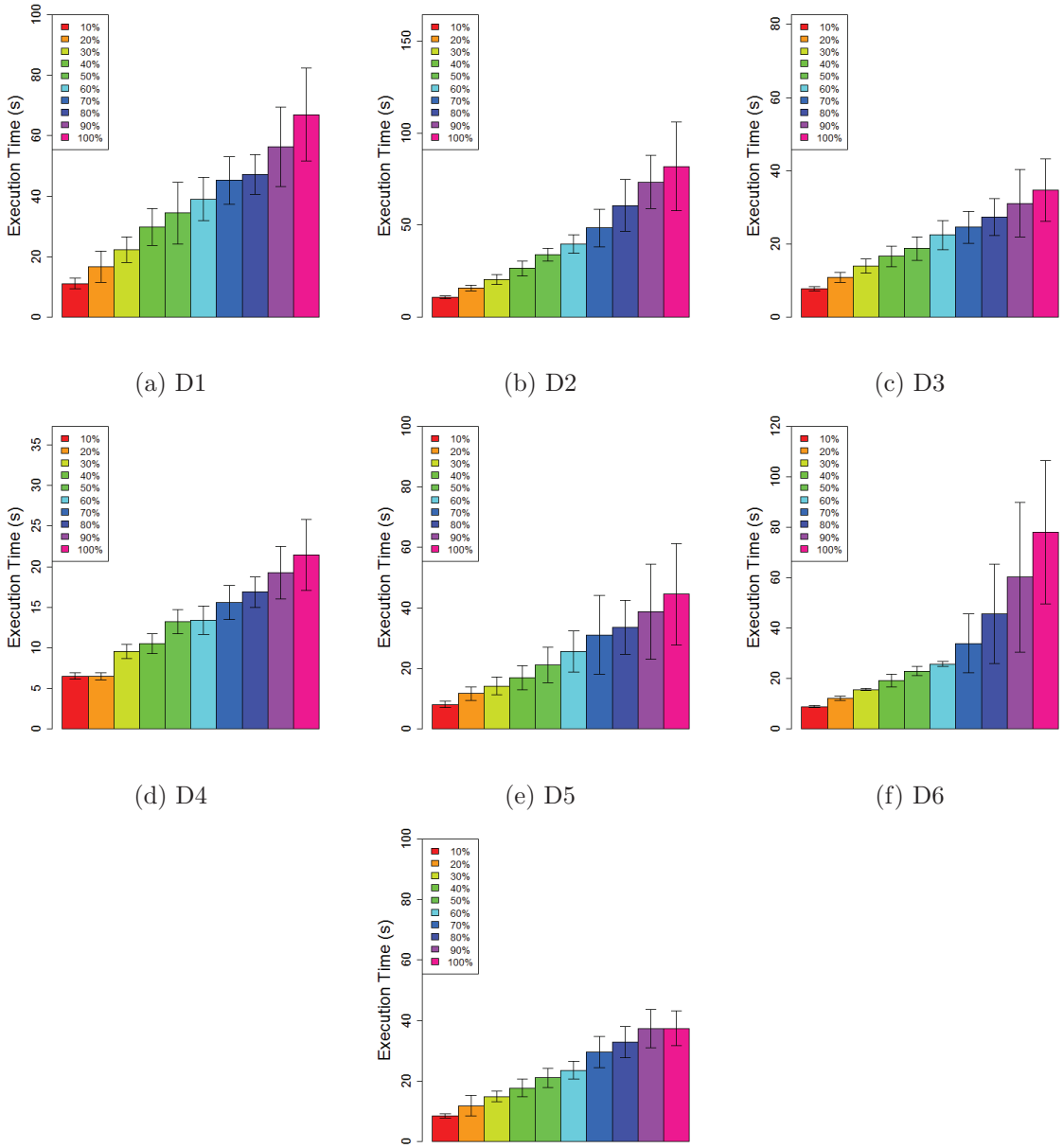
4.3.3. Execution Time Results

The execution time results averaged over 50 runs are shown in Figures 4.5 and 4.6. Figure 4.5 shows the execution time for the binary datasets, and Figure 4.6 for the multiclass datasets. All the results presented in both figures are as

predicted, confirming that the classification of a smaller segment size is faster than for a larger segment size. However, it is observed that Seg90 has no significant difference compared to Seg100, while all other segment sizes have a significant time difference compared to Seg100.

The number of fitness evaluations for each segment size for each of the 14 datasets are shown in Table 4.2, where the values are calculated using the formula stated before ($FE = D \times P \times G$). The values confirm the goal of the SegGP approach that using a segment of the dataset with a smaller size reduces the number of fitness evaluations of GP. It is obvious that when using a segment size of 10% of the whole dataset only 10% of the number of program executions of the whole dataset (100%) is needed. For example, for D1, the segment with 10% performs $18.7 * 10^6$ fitness evaluations, whereas the whole training dataset performs $187 * 10^6$ fitness evaluations. The same observation is valid for other segment sizes on all datasets (D1 to D14).

The speedup of each segment size compared to using the whole dataset is shown in Table 4.3. As can be observed, the smaller the segment size the higher the speedup, for example, using Seg10 saves on average 82.30% of the execution time rather than using the whole data, while using Seg50 saves almost 49.45% of the execution time, Seg70 saves 32.89% of the execution time, Seg80 saves 23.57%, and Seg90 is 11.49 faster than using the whole training dataset.

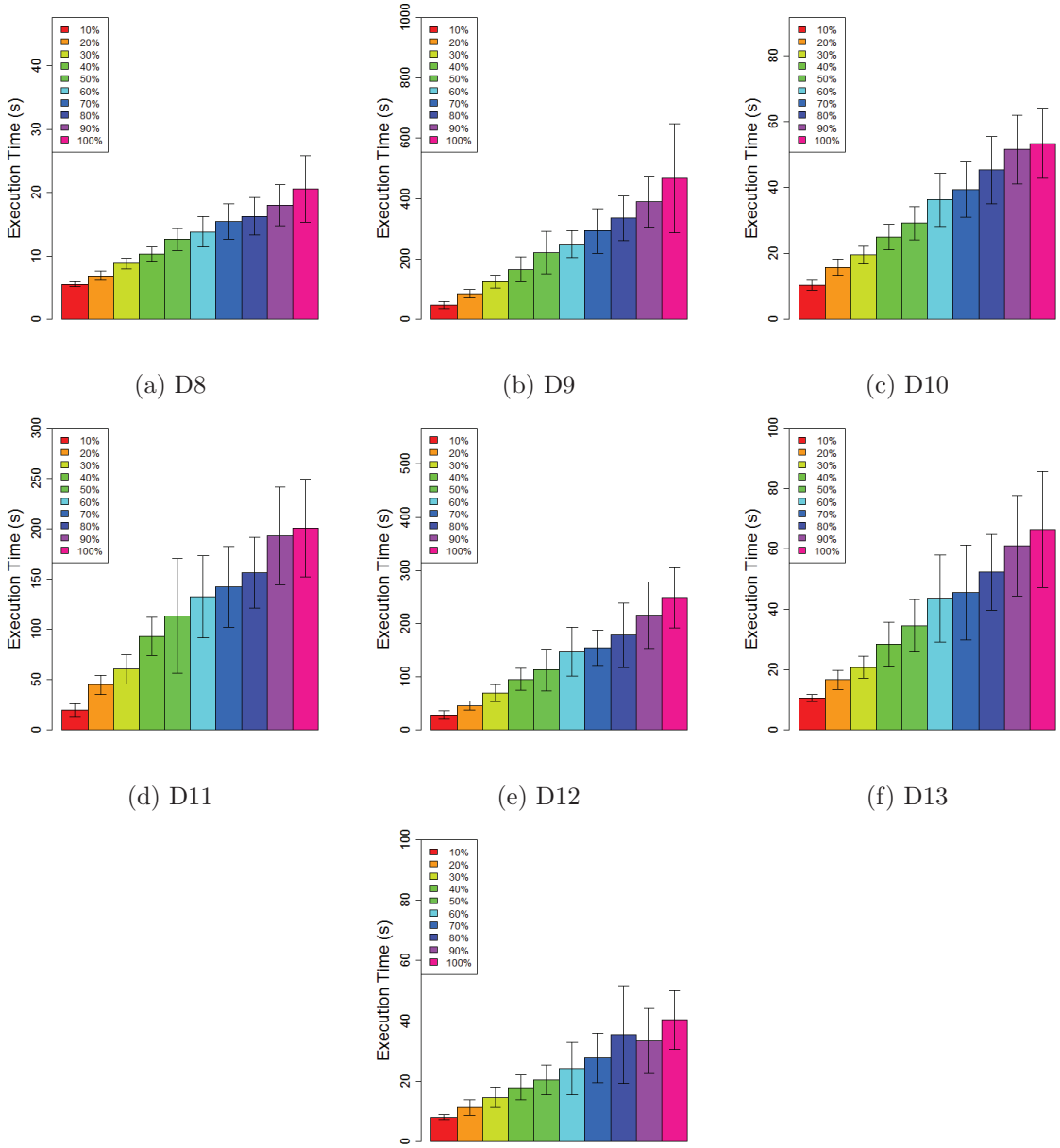


(a) D1 (b) D2 (c) D3

(d) D4 (e) D5 (f) D6

(g) D7

Figure 4.5: Execution Time For Binary Datasets



(a) D8 (b) D9 (c) D10

(d) D11 (e) D12 (f) D13

(g) D14

Figure 4.6: Execution Time For Multiclass Datasets

Table 4.2: FE For Binary And Multiclass Datasets [$*10^6$]

	10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
D1	18.70	37.40	56.10	74.80	93.50	112.20	130.90	149.60	168.30	187.00
D2	25.30	50.60	75.90	101.20	126.50	151.80	177.10	202.40	227.70	253.00
D3	8.85	17.70	26.55	35.40	44.25	53.10	61.95	70.80	79.65	88.50
D4	5.10	10.20	15.30	20.40	25.50	30.60	35.70	40.80	45.90	51.00
D5	10.00	20.00	30.50	40.50	51.00	61.00	71.00	81.50	91.50	102.00
D6	24.50	49.00	73.50	98.50	123.00	147.50	172.50	197.00	221.50	246.50
D7	11.50	23.00	34.50	46.00	57.50	69.00	80.50	92.00	103.50	115.50
D8	4.85	9.70	14.55	19.40	24.25	29.10	33.95	38.80	43.65	48.50
D9	105.25	210.50	315.75	421.00	526.25	631.50	736.75	842.00	947.25	1052.50
D10	12.05	24.10	36.15	48.20	60.25	72.30	84.35	96.40	108.45	120.50
D11	70.15	140.30	210.45	280.60	350.75	420.90	491.05	561.20	631.35	701.50
D12	70.15	140.30	210.45	280.60	350.75	420.90	491.05	561.20	631.35	701.50
D13	20.50	41.00	61.50	82.00	103.00	123.50	144.00	164.50	185.00	206.00
D14	10.00	20.00	30.50	40.50	51.00	61.00	71.00	81.50	91.50	102.00

Table 4.3: Time Speedup For Binary And Multiclass Datasets

	10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
D1	83.34	75.02	66.69	55.59	48.51	41.62	32.42	29.38	15.80
D2	86.56	80.56	74.96	67.60	58.44	51.45	40.88	25.86	10.42
D3	77.67	68.72	59.93	52.19	45.98	35.49	29.24	21.17	10.47
D4	69.64	69.80	55.41	50.93	38.25	37.57	27.17	21.34	10.20
D5	81.74	73.78	68.11	61.95	52.38	42.35	30.21	24.54	12.94
D6	88.63	84.29	79.97	75.26	70.61	67.05	56.58	41.61	22.91
D7	77.45	68.27	60.18	52.65	43.75	36.97	20.91	12.18	0.03
D8	73.08	66.46	57.09	49.87	38.57	32.91	24.75	21.01	12.48
D9	90.82	83.14	75.40	67.27	56.27	50.83	42.24	33.95	22.98
D10	80.59	70.42	63.49	53.26	45.35	31.96	26.26	15.16	3.44
D11	90.19	77.58	69.88	53.24	42.58	34.01	29.17	22.03	0.65
D12	88.60	81.42	71.93	61.65	54.51	40.73	37.91	28.26	13.29
D13	83.91	75.00	68.71	57.23	48.04	34.38	31.50	21.36	8.12
D14	80.00	71.71	63.66	55.41	49.10	39.92	31.24	12.21	17.16
Average	**82.30**	**74.73**	**66.81**	**58.15**	**49.45**	**41.23**	**32.89**	**23.57**	**11.49**

4.3.4. Statistical Test

The statistical T-test [48, 49] was applied on the accuracy results of the GP runs using different segment sizes 10% - 90%(Seg10 - Seg90) comparing SegGP using these segments with standard GP (Seg100) at a significance level of 5%.

The T-test results are shown in Table 4.4. The **bold entries** in the table indicate that there is no significant difference in accuracy between the segment size result compared to the whole dataset (values > 0.05 (P-value)), whereas the regular entries indicate the opposite. Considering SegGP's main goal, which is to obtain similar accuracy as GP using the whole dataset, we are interested in the segment sizes that have the most bold entries. Therefore, from the table we can derive which of the segment sizes 10%-90% have close accuracy values compared to 100% (as seen in Figure 4.3 and Figure 4.4). The T-test shows that there is a significant difference of the accuracy results for Seg10-Seg50 compared to Seg100.

On the other hand, the segments with sizes 60% - 90% have accuracy results that are not significantly different than 100% in most cases. Some of these segment sizes have two or three bold entries, where the randomness of the GP program creation, and natural operations enforcement might be a possible reason. Seg90 has only two entries that are significantly different compared to Seg100 (D2 and D4), where Seg80 has 4 entries out of 14, that are significantly different, the same as Seg70. Regarding Seg50 and Seg60, both have 9 and 5 entries respectively. In general we can conclude that accuracy results achieved by Seg60 to Seg90 (only 5 or 4 or 2 entries out of 14) are not significantly different than Seg100.

Table 4.4: T-test Significance Test For Binary And Multiclass Datasets

	10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**
D1	2.20E-16	4.19E-08	2.68E-06	7.06E-06	2.71E-06	**0.081**	0.013	**0.726**	**0.533**
D2	2.89E-09	3.73E-12	2.82E-10	3.36E-12	3.13E-12	6.88E-04	**0.332**	**0.304**	0.024
D3	4.80E-13	5.14E-10	9.49E-09	**0.102**	**0.072**	**0.069**	**0.159**	0.023	**0.121**
D4	8.33E-04	2.20E-16	0.002	4.61E-04	6.51E-04	**0.615**	**0.443**	0.018	0.014
D5	8.34E-12	1.34E-05	0.027	**0.242**	**1.557**	0.034	0.039	6.76E-04	**0.559**
D6	1.48E-06	**0.258**	**0.069**	3.82E-04	4.78E-04	**0.192**	**0.0894**	**0.855**	**0.372**
D7	1.79E-12	1.15E-10	2.72E-05	2.43E-05	2.29E-05	**0.064**	0.021	**0.146**	**0.752**
D8	2.20E-16	2.96E-06	3.19E-04	1.82E-05	1.36E-05	0.0027	**0.467**	0.0082	**0.587**
D9	1.09E-05	0.0017	**0.313**	**0.194**	**0.145**	**0.212**	**0.322**	**0.767**	**0.238**
D10	4.79E-10	6.07E-05	0.0015	**0.632**	**0.564**	**0.677**	**0.807**	**0.274**	**0.277**
D11	2.20E-16	5.71E-05	5.74E-05	0.0178	0.014	**0.104**	**0.393**	**0.943**	**0.780**
D12	0.0044	2.24E-04	**0.0791**	**0.212**	**1.79**	**0.589**	**0.533**	**0.469**	**0.589**
D13	5.02E-10	1.26E-05	2.52E-06	3.57E-04	4.99E-04	1.66E-06	3.73E-04	**0.270**	**0.125**
D14	1.41E-11	3.53E-10	2.40E-04	0.0078	0.005	2.97E-04	**0.177**	**0.375**	**0.642**

4.3.5. Sensitivity and Specificity

In the medical field, a classification model is evaluated using accuracy, sensitivity and specificity measures. As seen before, accuracy measures how many diagnostic tests are identified correctly, whereas sensitivity evaluates how good the test is at detecting a positive disease. Specificity estimates how likely patients without disease can be correctly ruled out [50].

Since we applied our proposed approach on many medical datasets, we measured the sensitivity and specificity in order to compare the standard GP with SegGP.

The sensitivity results for GP for Seg50 to Seg90, and GP for the whole dataset (100%) are represented in Table 4.5. The difference between Seg50 and Seg100 is, in general, very small (0.003-0.031) for almost all datasets except for D8 where the difference is 0.077 since Seg50 has a sensitivity of 0.37, and Seg100 has a sensitivity of 0.45.

The specificity results are illustrated in Table 4.6, where it can be observed that the values of GP using Seg50 and other segment sizes do not differ significantly to using the whole dataset (Seg100). Since the difference is very small and in the range of 0.002-0.019 for all the datasets except for D3, D4, and D14 where the differences are 0.026, 0.029 and 0.043, respectively.

In general, we can conclude that the differences for both sensitivity and specificity are small.

Table 4.5: Sensitivity For Binary And Multiclass Datasets

	50%	**60%**	**70%**	**80%**	**90%**	**100%**
D1	0.916	0.921	0.916	0.930	0.927	0.928
D2	0.698	0.699	0.666	0.690	0.694	0.694
D3	0.720	0.733	0.738	0.736	0.740	0.746
D4	0.613	0.620	0.628	0.653	0.661	0.642
D5	0.722	0.719	0.721	0.707	0.723	0.735
D6	0.510	0.511	0.520	0.517	0.519	0.527
D7	0.889	0.897	0.894	0.900	0.904	0.908
D8	0.376	0.500	0.373	0.364	0.381	0.453
D9	0.660	0.653	0.668	0.666	0.658	0.657
D10	0.530	0.570	0.560	0.571	0.545	0.561
D11	0.537	0.543	0.541	0.558	0.545	0.559
D12	0.218	0.230	0.225	0.218	0.235	0.228
D13	0.590	0.535	0.551	0.582	0.584	0.594
D14	0.587	0.581	0.592	0.623	0.623	0.605

Table 4.6: Specificity For Binary And Multiclass Datasets

	50%	**60%**	**70%**	**80%**	**90%**	**100%**
D1	0.916	0.921	0.916	0.930	0.927	0.928
D2	0.698	0.699	0.666	0.690	0.694	0.694
D3	0.720	0.733	0.738	0.736	0.740	0.746
D4	0.613	0.620	0.628	0.653	0.661	0.642
D5	0.722	0.719	0.721	0.707	0.723	0.735
D6	0.510	0.511	0.520	0.517	0.519	0.527
D7	0.889	0.897	0.894	0.900	0.904	0.908
D8	0.853	0.800	0.851	0.841	0.859	0.856
D9	0.807	0.806	0.813	0.813	0.806	0.809
D10	0.920	0.929	0.930	0.933	0.927	0.928
D11	0.768	0.775	0.779	0.786	0.786	0.787
D12	0.886	0.889	0.890	0.889	0.892	0.891
D13	0.855	0.819	0.834	0.857	0.853	0.870
D14	0.788	0.773	0.807	0.816	0.823	0.831

4.3.6. Data Analysis

To study the impact of the dataset structure on the GP results, an analysis of the datasets used in our experiments is performed. The analysis was first done using the Principal Component Analysis (PCA) to analyze the data which is the simplest of the true eigenvector-based multivariate analysis strategies.

PCA is a useful statistical technique that has been used in most forms of analysis because it is a simple, non-parametric method of extracting relevant information from confusing data sets [51][52]. There are many ways to select the number of components that best represent the dataset, one way is to choose the components that represent

over 90% of the total variation in the data [53].

In our analysis, PCA was applied on our dataset using R software [54]. Figure 4.7 shows the PCA results for the binary data sets, which illustrates the components of each datasets with its amount of variation. It can be inferred that for D1 and D6, 90% of the variation in the datasets is explained by one component, whereas two components are needed for D2, D3, and D4. However, three components explain D5 and ten components are needed for D7. The PCA results of the multiclass datasets are shown in Figure 4.8. The number of components to explain 90% of the D8 dataset is 5, where for D9 it is 19 components taken out of 22 features. Ten components can explain 90% of the D10 dataset, while only 3 components are needed to explain both D11 and D4. However, D12 and D13 datasets both need 4 components.

In order to extract a relation between the data set structure and the GP accuracy, Table 4.7 shows some information about the datasets, such as number of classes, classes distribution which is the number of records for each class, number of numerical features (#NF), number of categorical features (#CF), number of PCA components for which the cumulative variances are above 90% (PCA Comp.), and the average accuracy result when using the whole training dataset.

From this table we can observe that the binary data sets have an average accuracy range between 93.41% and 73.78%, where for the multiclass the average accuracy ranges between 83.28% and 37.9%. Moreover, when the number of classes is 3, the accuracy range between 83.28% - 70.97%, while with 4 classes the accuracy is 72.9%, for 6 classes 69.55%, and for 10 classes it is 37.9%. Thus, we can conclude that GP has higher accuracy values for binary datasets compared to multiclass datasets. Moreover, GP's accuracy decreases when the number of classes increases. Also, we can observe that all the datasets are unbalanced as shown by their classes distributions.

The impact of the number of numerical features on the accuracy, can be sum-

marized by the more number of numerical features the better the accuracy. Since the accuracy of having less than 4 numerical features is in the range 69.55% - 80.16%. However, a dataset with 4 to 7 numerical features, has an accuracy range between 72% - 83.47%. More numerical features (11 or 13) result in an average accuracy of 93.41% and 92.6% (excluding the CTG-Class dataset which has 10 classes that makes the average accuracy low 37.9%).

The number of categorical features has the opposite impact compared to the numerical features, which implies that more categorical features will reduce the average accuracy. From Table 4.7, a dataset with no categorical features or only one feature, has an accuracy between 93.41% - 74.42% (excluding CTG-Class for the same reason above). However, having more categorical features will result in the average accuracy ranging between 80.16% (with 4 features) and 69.55% (with 19 features).

Looking at the number of PCA components that can explain 90% of the data, no impact or relation can be suggested from the datasets and the accuracy results, since a dataset with only one PCA component has an accuracy range 93.41% - 74.42%, and a dataset with 10 or more PCA components has an accuracy range 92.60% - 69.55%. Therefore, no trend or relation can be inferred.

4.3.7. Summary of Results

Overall, comparing the results of SegGP using these segments with the standard GP (Seg100), it was shown that Seg80 achieves higher averaged accuracy values for four datasets (D1, D2, D4 and D10), and obtains close accuracy values for five datasets (D6, D7, D9, D11 and D13). Seg60 has three higher accuracy values (D2, D10, and D12), and six accuracy values close to Seg100 (D1, D4, D6, D7, D9 and D11). Seg90 achieved two higher accuracy values (D4 and D8) than Seg100, and nine accuracy values close to Seg100 (all except D2, D10, and D13).

Looking at the execution time, smaller segments leads to shorter execution time,

Table 4.7: Data Analysis

	Classes	Classes distribution	#NF	#CF	PCA Comp.	Avg Accuracy [%]
D1	2	211 : 357	11	0	1	93.41
D2	2	500 : 268	4	0	1	75.50
D3	2	150 : 119	3	6	1	73.78
D4	2	32 : 123	4	6	2	83.47
D5	2	210 : 100	6	0	1	75.94
D6	2	570 : 178	4	0	1	74.42
D7	2	126 : 225	13	1	6	92.60
D8	4	2 : 81 : 61 : 4	1	9	6	72.90
D9	3	767 : 768 : 1655	0	22	10	70.97
D10	6	112 : 61 : 72 : 49 : 52 : 20	0	19	4	69.55
D11	3	1655 : 295 : 176	7	0	3	83.28
D12	10	384 : 579 : 53 : 81 : 72 : 332 : 252 : 107 : 69 : 197	11	0	5	37.9
D13	3	49 : 288 : 288	0	4	4	80.16
D14	3	60 : 150 : 100	6	0	1	72.00

such as Seg60 has a speedup of 41.23% compared to the execution time of Seg100. Seg80's speedup is 23.57% compared to Seg100's execution time, the speedup of Seg90 is 11.49%. This is because smaller segments contain less records, therefore less number of fitness evaluations are needed.

The sensitivity and specificity results showed very small differences between GP and SegGP. For sensitivity the difference was in the range of (0.003-0.031) except for D8, and for specificity the difference was in the range of (0.002-0.019) except for D3, D4, and D14.

Furthermore, the statistical T-test demonstrated that SegGP using Seg10 to Seg50 has significantly different accuracy values than standard GP, while Seg60 to Seg90 did not show a significant difference compared to Seg100. The results of the statistical T-test suggest that using segments between 10%-and 50% produced an

accuracy results that are significantly different than using the whole training dataset (which are also lower than using Seg100). Therefore, it will not be efficient to use these segment size, however, using segments size 60% and larger produce similar or comparable accuracy results to Seg100.

Data analysis was performed and a relation between the number of classes, number of numerical features, and number of categorical features and the GP accuracy results were found. The GP accuracy increases when the number of numerical features increases, while it decreases with increasing number of classes and categorical features.

4.4. Summary

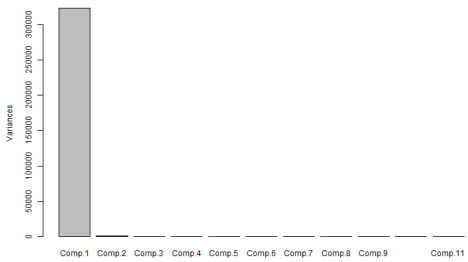
This chapter proposed a segment-based genetic programming (SegGP) approach that accelerates the genetic programming fitness evaluation process. SegGP is based on the idea of reducing the training dataset size but at the same time covering the whole training dataset. The goal is to obtain the same classification accuracy with a shorter execution time than standard GP. The results of SegGP suggest the use of data segments to be beneficial since for each fitness evaluation of the GP program a segment is selected and the program is executed.

A comparison of standard GP (which uses the whole training dataset) and SegGP with different segment sizes was conducted using 14 datasets. Ten segments with different percentage sizes of the whole dataset are used (10%, 20%, 30% ... 90%, and 100%), which are called Seg10, Seg20, Seg30, ..., Seg90, and Seg100, respectively. Measurements included the accuracy, execution time, number of fitness evaluations, sensitivity, specificity, and the statistical T-test.

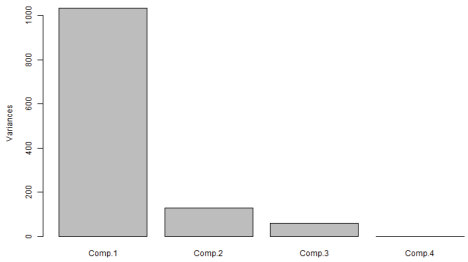
In summary, SegGP obtains the same or in some cases a higher accuracy than standard GP, using Seg80 or Seg60, and a speedup up to 23.57% or 41.23%, respectively is achieved (a smaller speedup is achieved using Seg90 (9.90%)). Moreover, the number of fitness evaluations for these segments is reduced compared to using the

whole dataset, whereby for Seg80 the reduction is 20%, and for Seg60 it is 40%; i.e., the smaller the segment size the higher speedup obtained. Similar sensitivity and specificity results were achieved for Seg50-Seg90 compared to Seg100. These results confirm that SegGP can acquire the same accuracy, sensitivity and specificity results, and that there is no significant difference compared to using the whole dataset, in a shorter execution time.

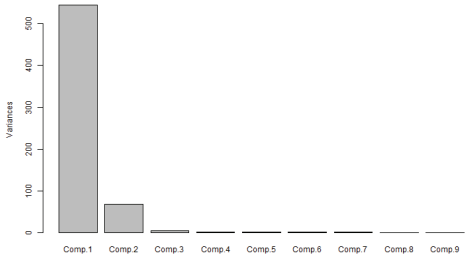
In addition, a comprehensive data analysis was performed to extract a relation between the dataset structure and the GP results, which revealed that increasing numbers of classes has a negative impact on the GP accuracy, since its accuracy is higher for binary datasets than for mutliclass datasets. In addition, the accuracy gets lower for datasets containing more classes and more categorical features. However, a positive impact on the accuracy was observed with the numerical features; the more numerical features the higher the accuracy. Moreover, a Principal Component Analysis (PCA) was applied on the datasets and no relation was found between the number of PCA components and the accuracy of GP.



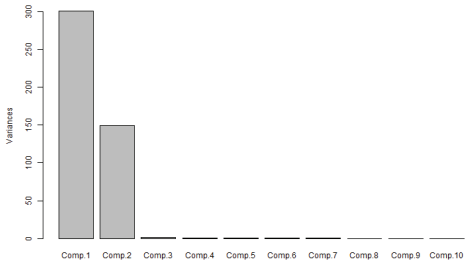
(a) D1 PCA



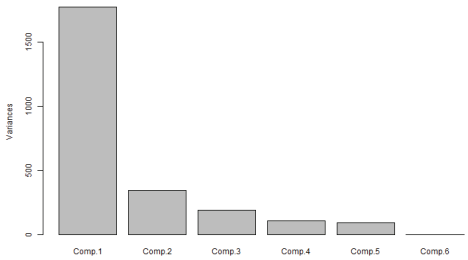
(b) D2 PCA



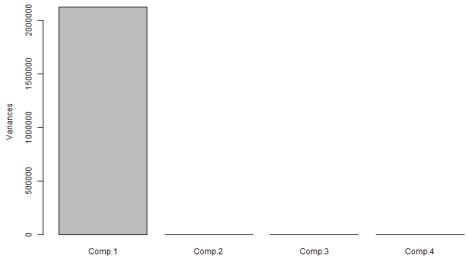
(c) D3 PCA



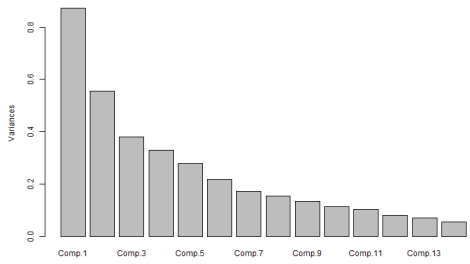
(d) D4 PCA



(e) D5 PCA

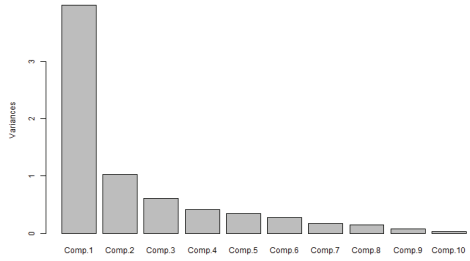


(f) D6 PCA

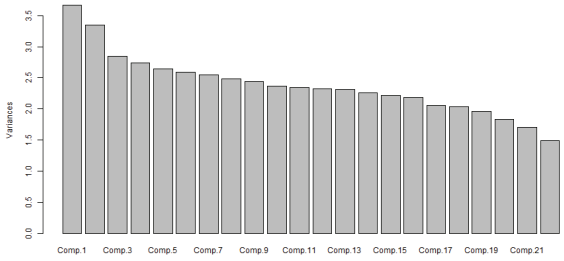


(g) D7 PCA

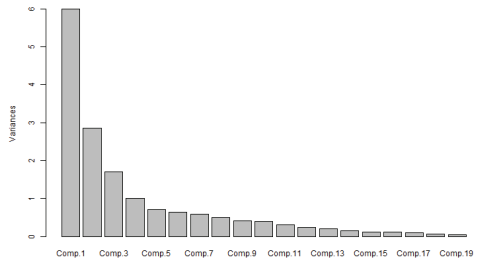
Figure 4.7: PCA Figures For Binary Datasets



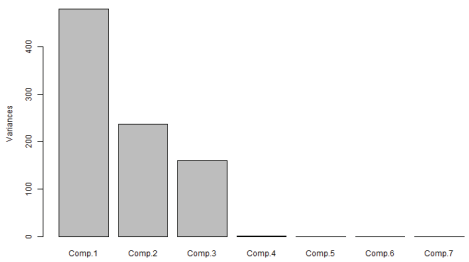
(a) D8 PCA



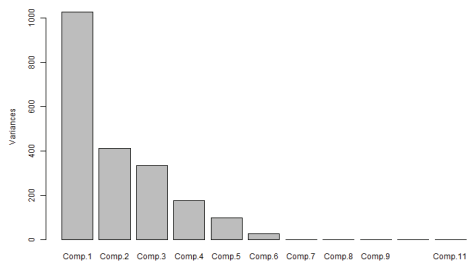
(b) D9 PCA



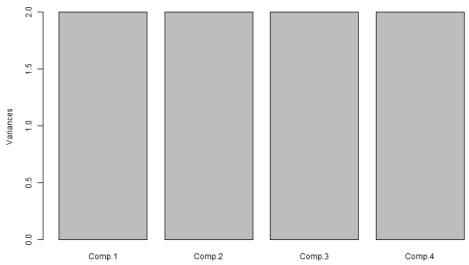
(c) D10 PCA



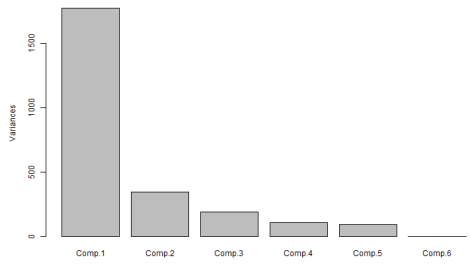
(d) D11 PCA



(e) D12 PCA



(f) D13 PCA



(g) D14 PCA

Figure 4.8: PCA Figures For Multiclass Datasets

CHAPTER 5. SCALING GENETIC PROGRAMMING FOR DATA CLASSIFICATION USING MAPREDUCE METHODOLOGY

GP suffers from the long run time since it is an iterative process (large number of generations are needed to find a good solution), in addition to the grammar used to define chromosome representations, and the consequent large search space. Moreover, the execution time of GP depends on the size of the dataset, large datasets require long execution time. As mentioned in [15], many typical GP problems do not have large sets of fitness cases for two reasons: first, evaluation has always been considered computationally expensive, and secondly, it is very difficult to evolve solutions to harder problems. Therefore, in this chapter a parallelized GP is implemented based on the MapReduce methodology (abbreviated as MRGP). MRGP does not only accelerate the execution time of GP for large datasets, it also provides the ability to use large population sizes, thus finding the best result in fewer numbers of generations. MRGP is evaluated using different population sizes ranging from 1,000 to 100,000 measuring the accuracy, scalability, and speedup.

This chapter is structured as follows: Section 5.1 introduces the main idea of this chapter. Section 5.2 presents some related works. In Section 5.3, we describe the MapReduce model, and introduce the proposed MRGP approach. Then, we report on our experiments, and show the results in Section 5.4. We summarize this chapter in Section 5.5.

5.1. Introduction

All nature-inspired algorithms are intrinsically parallel and distributed. Three main factors allow to easily parallelize EC algorithms [55]: 1) Individual programs are evaluated using multiple independent fitness cases; 2) Populations consist of individuals which could be evaluated on independent hardware in parallel; 3) Independent

runs can be executed simultaneously on different hardware.

Based on these factors three parallelization strategies (levels) can be implemented for EC algorithms:

- Fitness evaluation level: individuals are distributed among other nodes to compute the fitness values of the individuals. Based on the fact that evaluating a fitness function for every individual is the most costly operation of the algorithms.

- Population level: divides the population into several sub-populations then executes each sub-population.

- Individual level: each individual is placed on a grid and all algorithm operations are performed in parallel, evaluating simultaneously the fitness and applying local selection and algorithm-specific operations to a small neighboring group.

Algorithm parallelism can be done using different methodologies like Message Passing Interface (MPI) [56], or MapReduce [57] and many others. MapReduce is a prominent parallel data processing tool which has been gaining significant interest from both industry and academia. It is a new methodology proposed by Google in 2004, which is a programming model and an associated implementation for processing large data sets [57]. MapReduce enables users to easily develop large-scale distributed applications by supporting fault tolerance, load balancing, and data locality. In MapReduce, the user expresses the computation as two functions: Map and Reduce where the inputs and outputs are represented as a set of key/value pairs. Map takes an input pair and produces a set of intermediate key/value pairs. The MapReduce framework then groups all intermediate values associated with the same key and passes them to the Reduce function. The Reduce function accepts an intermediate key and a set of values for that key, and merges these values together to form a

possibly smaller set of values. The intermediate values are supplied to the user's Reduce function via an iterator. This allows the model to handle lists of values that are too large to fit in main memory.

This MapReduce model is used for many evolutionary computation algorithms such as Genetic Algorithms (GA) [58, 59], Particle Swarm Optimization (PSO) [60], Ant Colony Optimization (ACO) [61] and many others, which emphasize its effectiveness and efficiency to be used for big data calculations. To our knowledge, GP was not implemented using the MapReduce model until now. In this chapter, we parallelize the GP process using MapReduce methodology (MRGP) in order to accelerate the execution time of GP, by using large population sizes, hence, less number of generations may be needed to find good results. The main contributions of this chapter are the following: it demonstrates the transformation of GP into the Map and Reduce primitives, and confirms its ability to use large population sizes in order to find a good solution in shorter execution times (tackling GP's drawback of long execution time), and supports the scalability property to solve large problem sizes such as used for the classification of big data.

5.2. Related Work

GP has been used for many problems, such as classification, regression, and many other optimization problems. All of them were implemented as sequential versions, and based on our knowledge, GP was parallelized only using two approaches: Graphics Processing Unit (GPU) [62], and OpenCL [39]. In [62], the authors used GPUs to accelerate the GP by running the GP program on several GPUs in parallel, thereby demonstrating the benefit of GPUs to accelerate the GP approach. The authors showed that it is possible to get speed increases of several hundred times compared to a typical CPU implementation. In [39] the authors presented a detailed high-performance GP implementation in OpenCL for accelerated tree evaluation on

the CPU and GPU architectures. OpenCl is an open standard [63] for uniform and portable parallel programming across heterogeneous computing platforms. They concluded that GPU is considerably faster and more power efficient than CPU.

Since GA is the closest EC algorithm to GP, therefore, we will review some parallel GA approaches first. Different parallel approaches have been used to parallelize the GA process, and the first attempt of implementing GA with MapReduce was done as described in [59], where the authors presented an extension to the MapReduce model featuring a hierarchical reduction phase (MRPGA: MapReduce for Parallel GAs), which can automatically parallelize GAs. The authors described the design and implementation of the extended MapReduce model on a .NET-based enterprise Grid system, and claim that GAs cannot be directly expressed by MapReduce, therefore, they offer their own implementation to extend the model to MapReduceReduce. Several shortcomings of this MRPGA approach are pointed out in [58], however, showing that GA can be implemented with MapReduce without the need of modifying the structure of the MapReduce concept. In [58], the authors described the algorithm design and implementation of GAs on Hadoop [64] and investigated the convergence and scalability of the implementation on the BitCounting problem, where the results showed that their implementation converged after 220 generations, taking an average of 149 seconds per generation, and scaling well up to problems with 10^5 variables. In [65], the paper showed how GAs can be modeled with the MapReduce model. The authors describe the algorithm design and implementation of simple and compact GAs on Hadoop.

A practical application of GA modeled with the MapReduce was proposed in [66]. The authors implemented GA for the job shop scheduling problems using MapReduce, running experiments with various population sizes (i.e., up to 10^7), and on clusters of various sizes. Moreover, a parallel GA for the automatic generation of

JUnit test suites was proposed in [67]. The proposed solution is based on Hadoop MapReduce since it is well supported on cloud platforms and on graphic cards, thus, being an ideal candidate for high scalable parallelization of GAs. Although related work found in the literature have shown the remarkable power of GPUs in speeding up the execution of GP using different frameworks, so far no one has implemented and evaluated parallel GP using the MapReduce methodology supporting the properties of fault tolerance, load balancing, and data locality.

In this chapter, we are proposing a MapReduce GP approach (MRGP), which transforms the GP implementation into a parallel model using MapReduce methodology. MRGP aims to speed up the execution time of GP by providing the ability to use large population sizes in order to find a good solution in early generations, and to support the ability of GP to solve the classification of large scale data problems.

5.3. Proposed Approach

Given that our proposed approach is based MapReduce methodology, we first briefly introduce MapReduce before outlining the details of our proposed MRGP algorithm.

5.3.1. MapReduce Methodology

MapReduce is a highly scalable model and can be used across many computer nodes, and is mostly applicable for data intensive applications and when there are limitations on multiprocessing and large shared-memory machines.

The surpass of MapReduce moves the processing to the data not vice versa, and processes data sequentially to avoid random access that requires expensive seeks and disk throughput. MapReduce solves the problem by formulating it into two main operations, Map and Reduce. Both Map and Reduce operations take inputs and produce outputs in the form of <key, value>. The Map operation goes over a large number of records and extracts interesting information from each record, and then

all values with the same key are sent to the same Reduce operation. However, the Reduce operation aggregates intermediate results, generated from the Map function that has the same key, then generates the final results.

A well-known and commonly used implementation of MapReduce is Apache Hadoop [64]. It is an open source software framework that supports data-intensive distributed applications licensed under Apache. It enables applications to work with petabytes of data using thousands of independent processors. One of the main components of Hadoop is the storage component, Hadoop Distributed File System (HDFS). HDFS provides high-throughput access to the data and maintains fault tolerance by creating multiple replicas of the target data blocks. HDFS and MapReduce work together to support the ability of moving computation to the data, and not vice versa.

5.3.2. Proposed MRGP Approach

This chapter proposes a MapReduce GP approach (MRGP), which transforms the GP implementation from sequential into a parallel model using MapReduce. The goal is to accelerate the execution time of GP, providing the ability to use larger population sizes in order to find a good solution in early generations, and to give GP the ability to be used in the classification of large scale data.

GP's long execution time for solving classification problems arise from the fitness calculations of each program in the population. Each program is executed on every record of the data then the number of incorrectly classified records is counted. This number is then used as the fitness of that program. Hence, for example, if we have a population size of 100 programs, and a dataset of 1,000 records, then $100 \times 1000 = 10^5$ computations are executed for each generation. Therefore, MRGP distributes and parallelizes these computations to accelerate the fitness ranking process.

MRGP parallelizes the GP process based on the fitness evaluation parallelization

level by transforming the GP process into the Map and Reduce operations. The Map is responsible for the fitness evaluation, while the Reduce is responsible for performing the remaining GP process (selection, crossover, and mutation).

MRGP works as follows: at the beginning, the GP algorithm generates the initial population based on the given settings. Then, this population is written to the HDFS, thus, the mappers can access it. After that, the population is distributed on the mappers based on the number of mappers and the size of the population. Each mapper receives its part of the population and reads the programs from that part in the form of <Key, Value>, where the Key is the program id, and the Value is the program information. Then, it calculates the fitness of that program based on the dataset, and updates the program information to include the fitness, and then emits this program. The program is emitted also in the form of <Key, Value>, where the Key is the population number and the Value is the program information.

Using the population number as the output key of the mapper implies that all programs are going to the same reducer. The reducer recollects the population and then performs all GP steps of selection, crossover and mutation to produce the new population. The Reducer then emits this population in the form of <Key, Value>, where the Key is the program id, and the Value is the program information, which is used in the next mappers. Each Map and Reduce job is considered as a generation of GP. The jobs continue iteratively until the GP reaches its stopping criteria (either the maximum number of generations or a good solution is found). The result of the GP run from all generations is the program that has the best fitness value; therefore, we need to save this program throughout the generations. This process is done by the reducer also, where the best program found so far is saved to a file on the HDFS. The reducer updates this file whenever a better program than the one previously saved is found by collecting the programs from the mappers and comparing the best program

saved in the file with the best program given by the mappers. The detailed process of MRGP is shown in Figure 5.1, where the structure displays that MRGP has several mappers to perform the fitness calculation for the individuals in the population, and one reducer to combine the programs and proceed with the GP iteration process.

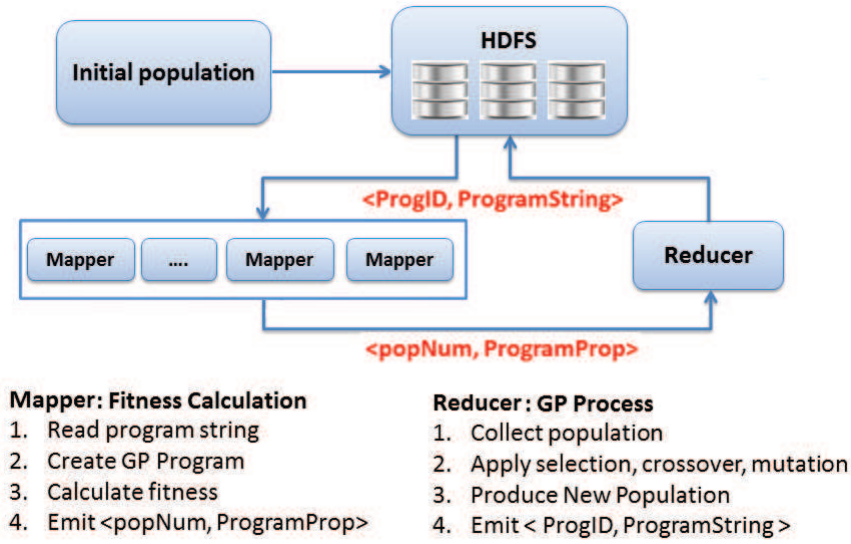


Figure 5.1: MRGP Process Architecture

5.4. Experiments and Results

In this section, we describe the experiments done to evaluate the MRGP algorithm. First, the GP settings, execution environment, and information of the datasets used for the classification problem are given. Then, the experimental results are provided and discussed.

5.4.1. Environment

To evaluate the proposed MRGP approach, we ran the experiments on the NDSU Hadoop cluster. The NDSU Hadoop cluster consists of 18 nodes, containing 6GB of RAM, 4 Intel cores (2.67 GHz each) with HDFS of 2.86 TB aggregated capacity. For the MapReduce framework, Hadoop version 0.20 was used.

Experiments were performed using the Java Genetic Algorithms Package (JGAP) [22] with the following settings:

- Population size = 1,000, 5,000, 10,000, 50,000.
- Number of generations = 100
- Crossover probability = 0.5
- Mutation probability = 0.1
- Maximum initial depth = 8
- Maximum crossover depth = 8
- Function probability = 0.7
- Dynamize arity probability = 0.05
- New chromosome percentage = 0.2

The functions used are $+, -, *, /, Exp, Pow$, and Log. Given the stochastic nature of GP, ten independent runs were performed.

The experiments are applied on six datasets [23], each was partitioned into two parts; 66% of the dataset for training the GP and building the classifier, and the remaining 34% for testing the classifier. All details including the number of features and records of these datasets are shown in Table 5.1.

Table 5.1: Datasets

	Dataset	**Classes**	**Features**	**Records**
D1	Iono	2	34 (14)	351
D2	Vertebral-2C	2	6 (6)	310
D3	Blood	2	5 (4)	748
D4	Balance	3	4 (4)	625
D5	Vertebral-3C	3	6 (6)	310
D6	CTG-NSP	3	22 (7)	2126

A pre-processing stage was performed on the datasets, where a feature selection process using the WEKA software [24] was performed to choose the best features of

the datasets (a supervised attribute filter that allows various search and evaluation methods to be combined [24]). The resulting number of features is shown in the brackets in Table 5.1. The experiments are applied on two types of datasets (binary and multi-class). The benchmark classification problems are:

- Ionosphere: Classification of radar returns from the ionosphere. The class label is either "Good", which means that radar returns are those showing evidence of some type of structure in the ionosphere, or "Bad", which means the returns are those that do not (their signals pass through the ionosphere).

- Vertebral Column: Data set containing values for six biomechanical features used to classify orthopaedic patients into 3 classes (normal, disk hernia or spondilolysthesis), or 2 classes (normal or abnormal).

- Blood Transfusion Service Center: Data taken from the Blood Transfusion Service Center in the Hsin-Chu City of Taiwan. Class labels represent whether or not the person donated blood in March 2007.

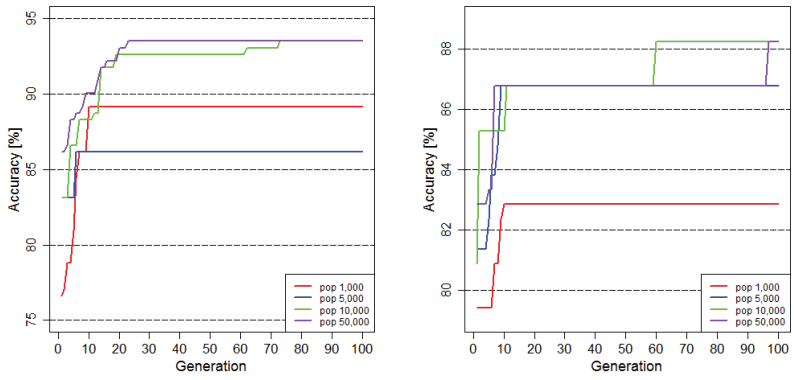
- Balance Scale: generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced.

- Cardiotocography: fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. Classification is categorized with respect to a fetal state (N, S, P).

5.4.2. Results

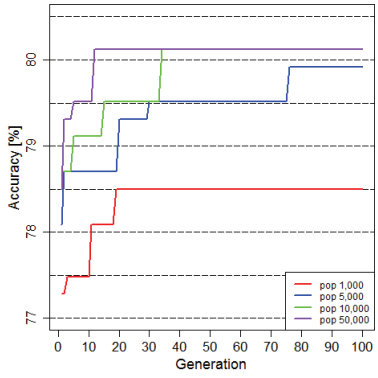
To evaluate MRGP, the experiments measure the speed of convergence and the impact of population sizes, testing accuracy, average time for mapper, speedup and scalability.

The speed of convergence of MRGP per generation, i.e., highest accuracy that can be achieved for each dataset is measured. In addition, a comparison of the impact of the population size on the MRGP convergence is performed. The experimental results for different populations sizes (1,000, 5,000, 10,000 and 50,000) are displayed in Figure 5.2, where it is shown that larger population sizes yield better solutions, which is intuitive since more function evaluations are performed. For example, for D1, the population size 1,000, starts with an accuracy of 76.6%, while population 5,000 starts with 83.12%, and population 50,000 starts with 86.14%. Moreover, at the end of 100 generations, population 1,000's best accuracy is 89.18%, while population 50,000's best accuracy is 93.51%. On the other hand, the accuracy of 89.18% is achieved with population size 50,000 in the 9^{th} generation unlike the 100 generations needed for population size 1,000. In general, the same observation can be obtained with the other datasets. Therefore, larger population sizes achieve better results when the same number of generation is used. This can be explained by having larger population sizes implies that more individuals search the solution space, and therefore, the possibility to find better results is higher than using smaller population sizes.

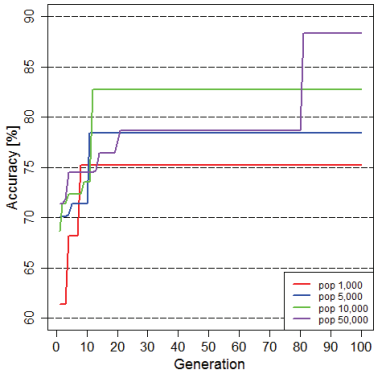


(a) D1 Convergence

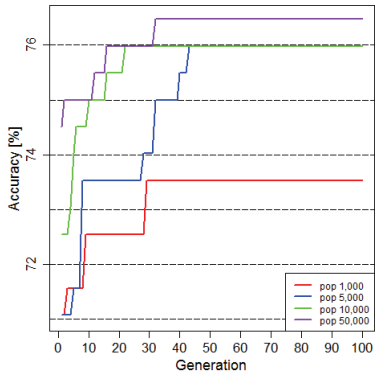
(b) D2 Convergence



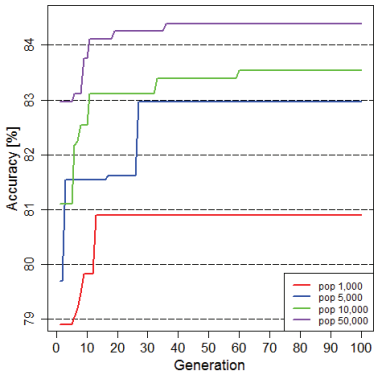
(c) D3 Convergence



(d) D4 Convergence



(e) D5 Convergence



(f) D6 Convergence

Figure 5.2: Convergence Speed With Different Population Sizes

After the GP has trained the classifier by finding the best program using the training dataset, this classifier is then tested using the testing dataset by measuring its quality using the accuracy metric. The accuracy results of the MRGP classifier for

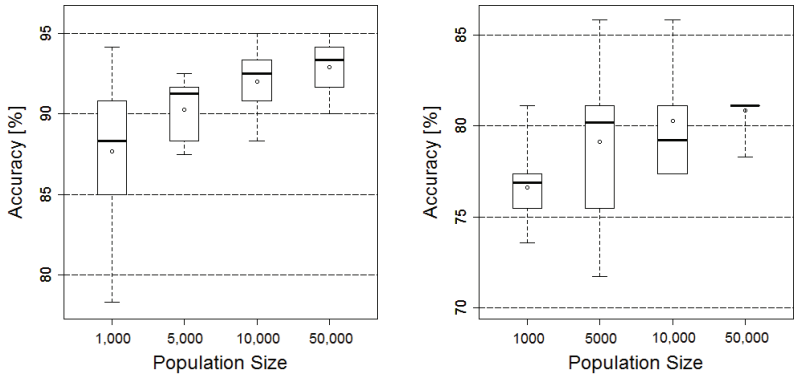
Table 5.2: Accuracy Results - Testing Phase

Dataset	**Pop 1000**	**Pop 5000**	**Pop 10000**	**Pop 50000**
D1	87.66	90.25	91.998	92.91
D2	76.60	79.15	80.28	81.13
D3	74.51	74.82	75.01	75.36
D4	71.22	73.89	75.30	81.59
D5	77.73	83.20	78.87	81.88
D6	81.99	81.66	82.46	82.26

the six datasets and by using different population sizes are shown in Table 5.2. The tables illustrate the average accuracy of ten independent runs for each dataset. From the tables we can infer that for each dataset, the larger the population size the better the accuracy. In general, the difference between the accuracy of using a population size of 1,000 and 5,000 is somehow smaller, while using larger population sizes such as 10,000 or 50,000 increases the difference. For example, for D1, the accuracy for population size 1,000 is 87.66%, while for population size of 5,000 it is 90.25% (with +2.59% difference), nevertheless, the accuracy of population 50,000 is 92.91% (the difference compared to population size 1,000 is +5.25%). The same trend is seen for D4, where pop1000's accuracy is 71.22%, while pop5000's is 73.89% (with difference +2.67%), while pop10,000's accuracy is 75.30 (difference equal to +4.08%), moreover, pop50,000's accuracy is 81.59% (with the highest difference +10.37%).

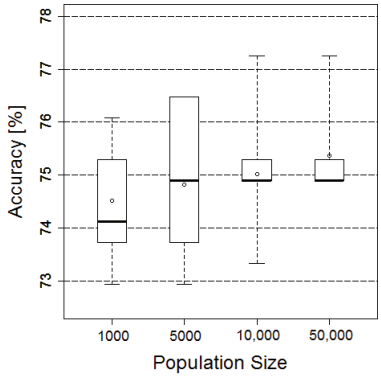
The detailed accuracy results for the six datasets using different population sizes are shown by the box plots in Figure 5.3, where the circle represents the average of 10 runs, the solid bar depicts the median, the lower end of the dashed line represents the minimum accuracy observed by the GP for a given population size, and the upper end of the dashed line depicts the maximum accuracy value. It can be observed from the figure that using a population size of 50,000 leads to better average accuracy. Moreover, it has smaller box sizes (especially for D1, D2, D3, and D4), which means that the accuracy values are within a small range, hence it is more reliable than using

a population size of 50,000.

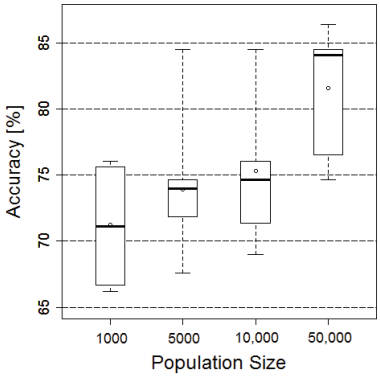


(a) D1 Accuracy

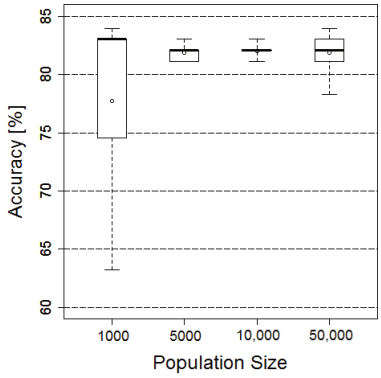
(b) D2 Accuracy



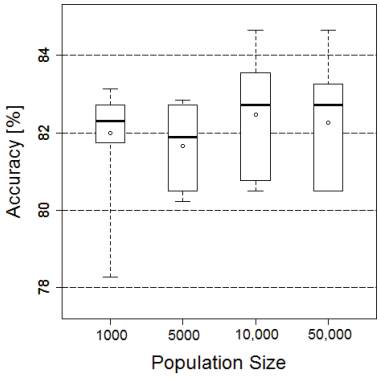
(c) D3 Accuracy



(d) D4 Accuracy



(e) D5 Accuracy



(f) D6 Accuracy

Figure 5.3: Accuracy results - Testing Phase

However, since the scaling of the population size by keeping the number of generations fixed is not a fair comparison, we have investigated MRGP in terms

of numbers of fitness evaluations. In particular, we kept the number of fitness evaluations fixed to 100,000, using different populations sizes of 1,000, 2,500, 5,000, 7,500, and 10,000 and using different number of generations with 100, 40, 20, 13, and 10, respectively. Figure 5.4 shows the accuracy results whereby for all columns belonging to a particular dataset all accuracy results show no significant difference as can be seen by the standard deviation bars. However, what we can clearly see in Figure 5.5 is that the run time for larger population sizes significantly reduces as seen by all datasets. The reason for this, is using larger population sizes need smaller numbers of generations, which in turn means less MapReduce jobs.

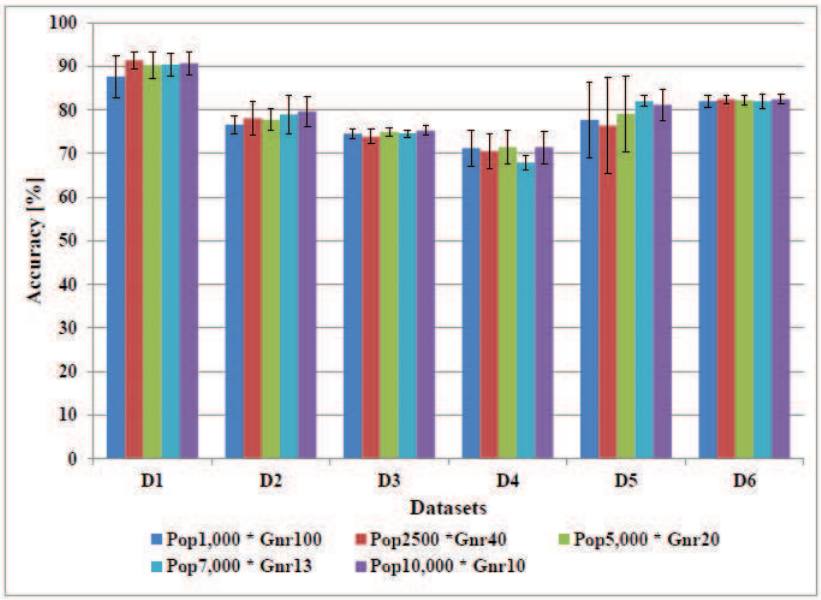


Figure 5.4: Accuracy Results Based On Fitness Evaluations (100,000)

The parallelization of MRGP provides the ability to use large population sizes such as 50,000 and obtain similar high accuracy values. Therefore, to evaluate the parallelization aspect of MRGP, first we calculated the average time needed for different numbers of mappers using the same population size of 100,000 programs and applied it on dataset D6 since it is the largest one. A large population and large dataset were chosen in order to demonstrate higher utilization rates of the

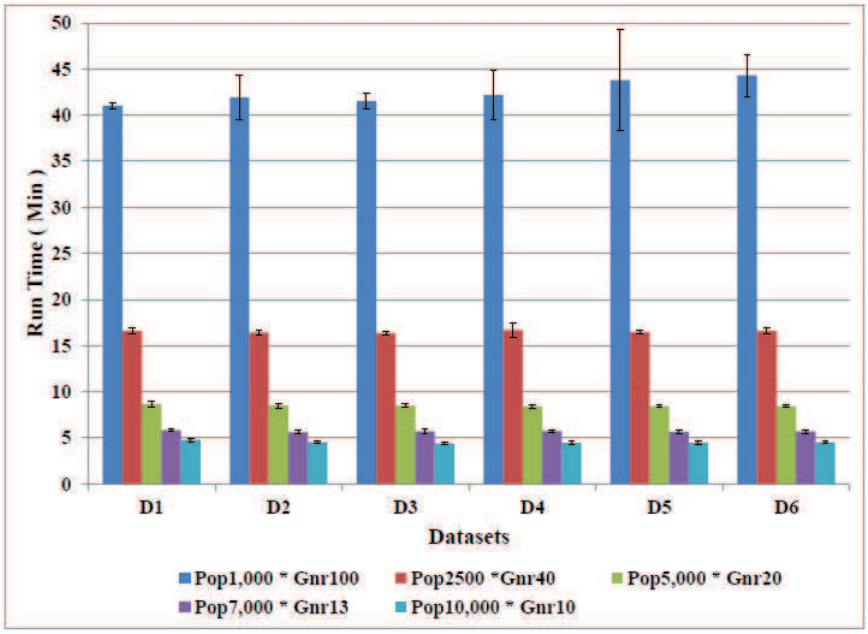


Figure 5.5: Time Results Based On Fitness Evaluations (100,000)

MapReduce (Hadoop) framework, and to reduce the parallelization overhead such as starting MapReduce jobs, starting mappers and reducers operations.

Figure 5.6 displays the average time spent in a mapper, showing that using more mappers reduces the execution time needed in each mapper. This is because the population is divided into smaller parts based on the number of mappers, hence, using more mappers leads to the division of the population into smaller portions, and in turns leads to a faster average execution time in the mapper. However, when the number of mappers increases, the overhead of initializing the mappers also increases, therefore, in Figure 5.6 the average time per mapper does not reduce according to the number of mappers.

The scalability measures the impact of using different population sizes when the same number of mappers is used. Figure 5.7 shows the average time per mapper, for each population size, using the six datasets, and 10 mappers. It can be inferred that MRGP scales well between population sizes 1,000 to 10,000 on all the datasets. However, it increases drastically as the population size reaches 50,000. Therefore, a

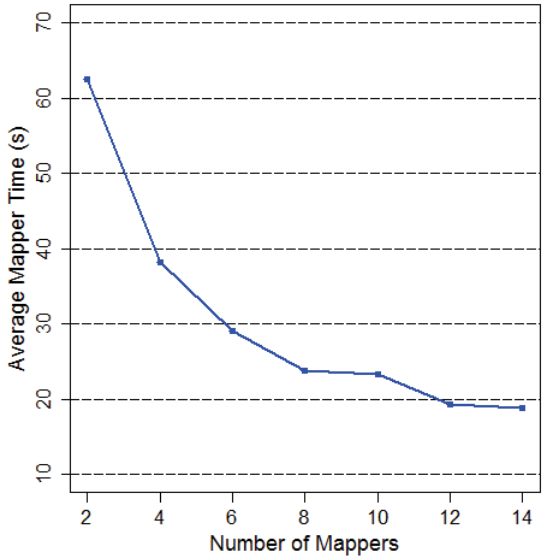


Figure 5.6: Average Time Per Mapper

recommendation that can be made here is that using a population size between 1,000 to 10,000 and adding more mappers would enable MRGP to scale well for larger population sizes.

We note in this figure, that the average mapper running time when using larger population sizes does not increase by the same ratio as the size of population, such as when using a 50 times larger population size (comparing 1,000 to 50,000), the running time increases approximately twice. The reason for this is that the MapReduce processes of copying and sorting of the larger populations causes extra overhead.

Figure 5.8 presents the speedup results of the MRGP algorithm compared to the optimal linear speedup. To measure the speedup, we fixed the population size to 100,000 by increasing the number of mappers by a certain ratio (factor of 2). Then, the speedup measure is calculated as:

$$Speedup = \frac{Time_2}{Time_n} \tag{5.1}$$

where $Time_2$ is the time using 2 mappers, $Time_n$ is the time using n mappers,

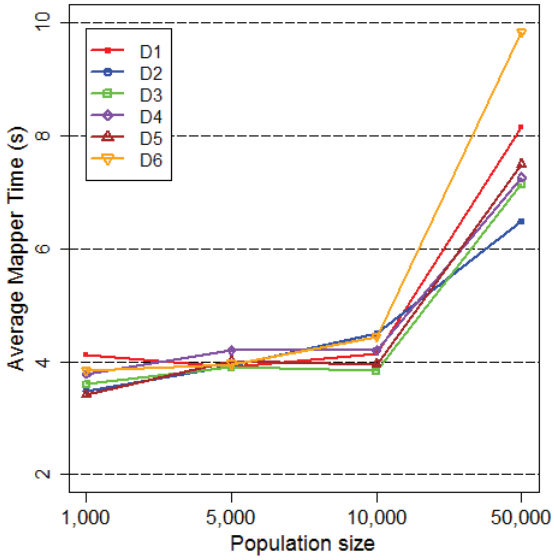


Figure 5.7: Scaleup - Mapper Time

where n= {2, 4, 6, 8, 10, 12, 14}. We started with 2 mappers and used the factor of 2, because running the MRGP using one mapper will not be efficient due to the high overhead of initializing the MapReduce framework compared to running the standard GP.

The figure illustrates that the speedup was very close to the linear speedup (optimal scaling) using 4, 6 mappers, but after that (with more mappers) it diverges from the linear speedup quite dramatically. This is because of the overhead of the MapReduce (Hadoop) framework, which results from starting the mappers and storing the outputs to the distributed file system, in addition to other management tasks. We have to note that the Hadoop framework has proved its efficiency of using very large input/data (which is in our MRGP the population size) and that MapReduce's speedup scales much better with larger population sizes. To proof this we ran MRGP with 200,000 or larger population sizes, however, the JGAP framework did not work due to garbage collection issues which need to be addressed.

In summary, MRGP confirms the ability of implementing GP with the MapReduce framework by providing the ability of using large population sizes, such as 50,000,

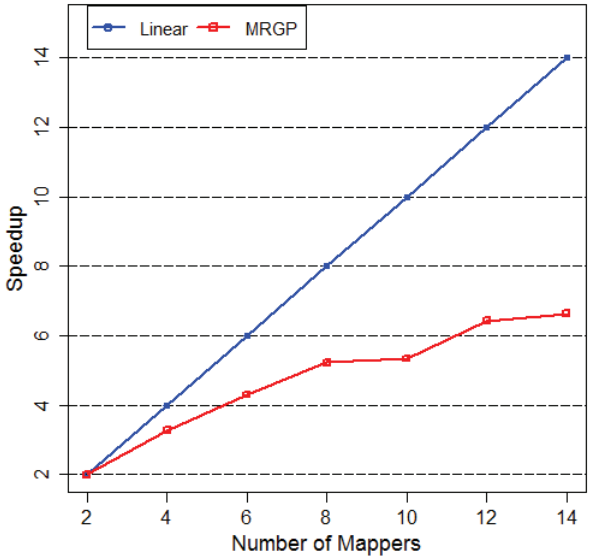


Figure 5.8: MRGP Speedup

which is somehow impossible for the sequential GP. Also, using larger population sizes result in a faster convergence and a higher accuracy. Moreover, the structure of MRGP guarantees its scalability, and supports its speedup based on the provided resources. This supports the ability of applying MRGP to the classification of large scale data, which is not possible with the sequential version.

5.5. Summary

This chapter proposed a parallelized version of the GP algorithm applied to the fitness evaluation level, referred to as MRGP. MRGP is implemented based on the MapReduce methodology which is a new methodology that manages fault tolerance, load balancing, and data locality. MRGP accelerates the execution time of GP by distributing the population to different mappers, which perform the fitness evaluation of the programs, and then the reducer combines these mappers and continues the GP process. MRGP also provides the ability to use large population sizes, thus, finding the best result in fewer numbers of generations. Experiments evaluated MRGP measuring the speed of convergence, accuracy, average time per mapper, scalability and speedup. The results confirm the goals of MRGP by accelerating the execution

time, supporting the usage of large population sizes, and obtaining higher accuracy while maintaining the speedup and scalability properties.

CHAPTER 6. CONCLUSION AND FUTURE DIRECTIONS

In this dissertation, our goal was to improve Genetic Programming (GP) in solving classification problems. The drawbacks that GP suffers from when solving classification problems are the long execution time, the need to tune many parameter settings before the GP run, and the low classification accuracy for multiclass problems compared to the binary classification problems.

In order to tackle these drawbacks, we proposed several approaches. Firstly, adaptive GP variants were proposed in order to automatically adapt the parameter settings and shorten the execution time. The first variant considered the selection step in GP, and applied a criterion on the selected programs fitness to ensure that the new generation is better than the old one, which leads to find the best solution faster. The second variant adapts the crossover and mutation probabilities, where programs with high fitness values will have a low probability for both processes, in such way they will be protected from being changed. The third variant adapts the function list that is used in the GP programs, whereby the function list is adapted based on functions that performed well during the earlier generations and keeping only the functions that best suit the problem. These GP variants could automatically adapt the GP parameters and perform the classification tasks faster.

Secondly, two approaches were proposed to improve the accuracy of GP when applied to multiclass classification problems. The first approach uses the K-means clustering technique in order to transform the produced numeric value of the GP program into class labels. The second approach uses a discretization technique to perform the transformation. The results were good improvements on the accuracy values compared to the standard GP, and competitive accuracy values compared to the state-of-the-art classifiers.

To address the GP long execution time and at the same time keeping the good accuracy of GP, a segment-based approach was proposed to accelerate the GP execution time for the data classification problem. This approach is based on creating segments of the training dataset, and using these segments in the fitness evaluation step of each program. The whole training dataset is used by the population but each program uses only a segment. The results of the proposed technique prove that the use of data segments is beneficial since it shortens the execution time, reduces the number of fitness evaluations, and achieves the same accuracy as the standard GP.

In the last chapter of this dissertation, a parallelization of the GP process using the MapReduce methodology was proposed, which aims to shorten the GP execution time and to provide the ability to use large population sizes leading to a faster convergence. The proposed technique accelerates the execution time of GP by distributing the population to different mappers, which perform the fitness evaluation of the programs, and then the reducer combines the mappers' output and continues the GP process. The proposed technique also provides the ability to use large population sizes, thus, finding the best result in fewer numbers of generations. The results confirm the goals of this approach by accelerating the execution time, supporting the usage of large population sizes, to obtain higher accuracy.

This dissertation discussed GP and its usage in solving classification problems, it also proposed some approaches in order to overcome GP's drawbacks. The experimental results of testing each proposed approach showed that it can achieve its goal to solve GP's drawbacks, thus making GP more applicable to be used to solve additional problems such as real time problems. Therefore, as a recommendation of this study that can be derived by the experiments, the proposed approaches can be used in different situations based on the main objective to be accomplished. Adaptive GP is recommended for unknown classification problems since the parameter settings

of GP are adjusted automatically. However, when high accuracy is the main concern, then GP-D or GP-K are the best variants to be used. When a reasonable execution time is paramount while maintaining good accuracy, Seg-GP and MR- GP are the two variants to be used. Moreover, a complex problem with a large search space can be solved using MR-GP as it provides GP the ability to use large population sizes.

As for future work, our plan includes implementing other adaptation techniques such as the selection based on age, and to combine GP using discretization (GP-D) with the accelerated version of GP (Seg-GP) in order to keep the high accuracy and good robustness and at the same time shorten the running time. Regarding the segment-based GP, future work will also involve testing other balanced and larger datasets, in addition, to test different methods of segment creation such as the oversampling and spread resampling methods. Moreover, for the MapReduce GP, our plan includes solving the issues of the current GP implementation and investigating larger population sizes, parallelizing GP on the population level, and running experiments especially for the classification of large data sets. Also, we intend to apply GP to solve different optimization tasks such as regression and clustering, and use it for specific applications such as job scheduling, or on a specific medical dataset.

REFERENCES

[1] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* SpringerVerlag, 2003.

[2] J. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3(4):184–195, 1960.

[3] J. H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[4] Y. Fang and J. Li. A review of tournament selection in genetic programming. In *Proceedings of the 5th international conference on Advances in computation and intelligence*, ISICA'10, pages 181–192, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] A. Sokolov, D. Whitley, and A. M. S. Barreto. A note on the variance of rank-based selection strategies for genetic algorithms and genetic programming. *Genetic Programming and Evolvable Machines*, 8(3):221–237, 2007.

[6] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[7] H. Jabeen and A. R. Baig. Review of classification using genetic programming. *International Journal of Engineering Science and Technology*, 2(2):94–103, 2010.

[8] L. C. Jain and A. Ghosh. *Evolutionary Computation in Data Mining (Studies in Fuzziness and Soft Computing).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[9] K. Tan, Q. Yu, C.MHeng, and T. Lee. Evolutionary computing for knowledge discovery in medical diagnosis. *Artificial Intelligence in Medicine*, 27(2):129–154, 2003.

[10] C. A. Pena-Reyes and M. Sipper. Evolutionary computation in medicine: an overview. *Artificial Intelligence In Medicine*, 19(1):1–23, 2000.

[11] S. M. Winkler, M. Affenzeller, and S. Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis - an empirical study. In *MedGEC 2006 GECCO Workshop on Medical Applications of Genetic and Evolutionary Computation*, Seattle, WA, USA, 8 July 2006.

[12] W. Smart and M. Zhang. Using genetic programming for multiclass classification by simultaneously solving component binary classification problems. In *Proceedings of the 8th European conference on Genetic Programming*, EuroGP'05, pages 227–239, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] S. A. Ludwig and S. Roos. Prognosis of breast cancer using genetic programming. In *Proceedings of the 14th international conference on Knowledge-based and intelligent information and engineering systems: Part IV*, KES'10, pages 536–545. Springer-Verlag, 2010.

[14] R. Poli, W. B. Langdon, and N. F. McPhee. *A field guide to genetic programming*. Published via `http://lulu.com` and freely available at: `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).

[15] S. L. Harding and W. Banzhaf. Fast genetic programming and artificial developmental systems on GPUs. In *21st International Symposium on High Performance*

Computing Systems and Applications (HPCS'07), page 2, Canada, 2007. IEEE Computer Society.

[16] M. Affenzeller and S. Wagner. Offspring selection: A new self-adaptive selection scheme for genetic algorithms. In *Adaptive and Natural Computing Algorithms*, pages 218–221. Springer Vienna, 2005.

[17] T. White and A. Salehi-Abari. A swarm-based crossover operator for genetic programming. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1345–1346, 2008.

[18] Y. Kameya, J. Kumagai, and Y. Kurata. Accelerating genetic programming by frequent subtree mining. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, pages 1203–1210, New York, NY, USA, 2008. ACM.

[19] H. Luchian and O. Gheorghie. Integrated-adaptive genetic algorithms. In *Advances in Artificial Life*, volume 2801 of *Lecture Notes in Computer Science*, pages 635–642. Springer Berlin Heidelberg, 2003.

[20] J. C. Bongard. A probabilistic functional crossover operator for genetic programming. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 925–932, New York, NY, USA, 2010. ACM.

[21] M. Srinivas and L. Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

[22] K. Meffert et al.: (JGAP) - Java Genetic Algorithms and Genetic Programming package [online]. available: http://jgap.sf.net, January 2012.

[23] A. Frank and A. Asuncion. UCI machine learning repository [online]. available: http://archive.ics.uci.edu/ml, 2010.

[24] I. Witten, E. Frank, and M. Hall. *Data Mining: Practical Machine Learning Tools And Techniques, 3rd Edition*. Morgan Kaufmann, 2011.

[25] J. Fitzgerald and C. Ryan. Drawing boundaries: using individual evolved class boundaries for binary classification problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1347–1354, 2011.

[26] A. Song, T. Loveard, and V. Ciesielski. Towards genetic programming for texture classification. In *Proceedings of the 14th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '01, pages 461–472, London, UK, UK, 2001. Springer-Verlag.

[27] W. A. Tackett. Genetic programming for feature discovery and image discrimination. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 303–311. Morgan Kaufmann Publishers Inc., 1993.

[28] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas. Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain. In *GECCO*, pages 953–958, 1999.

[29] C. Est'ebanez, R. Aler, and J. M. Valls. A method based on genetic programming for improving the quality of datasets in classification problems. *IJCSA*, 4(1):69–80, 2007.

[30] M. Zhang and V. Ciesielski. Genetic programming for multiple class object detection. In *Proceedings of the 12th Australian Joint Conference on Artificial*

Intelligence: Advanced Topics in Artificial Intelligence, AI'99, pages 180–192, London, UK, UK, 1999. Springer-Verlag.

[31] M. Zhang, V. Ciesielski, and P. Andreae. A domain-independent window approach to multiclass object detection using genetic programming. *EURASIP Journal on Applied Signal Processing*, 2003:841–859, 2003.

[32] W. Smart and M. Zhang. Probability based genetic programming for multiclass object classification. In *PRICAI 2004: Trends in Artificial Intelligence*, volume 3157 of *Lecture Notes in Computer Science*, pages 251–261. Springer Berlin Heidelberg, 2004.

[33] B. Lam and V. Ciesielski. Discovery of human-competitive image texture feature extraction programs using genetic programming. In *Genetic and Evolutionary Computation Conference (GECCO 2004)*, volume 3103 of *Lecture Notes in Computer Science*, pages 1114–1125. Springer Berlin Heidelberg, 2004.

[34] R. Haralick, K. Shanmugam, and I. Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-3(6):610–621, 1973.

[35] M. Zhang and W. Smart. Multiclass object classification using genetic programming. In *Applications of Evolutionary Computing*, volume 3005 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 2004.

[36] R. R. F. Mendes, F. de B. Voznika, A. A. Freitas, and J. C. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '01, pages 314–325, London, UK, UK, 2001. Springer-Verlag.

[37] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[38] U. M. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *International Joint Conference on Artificial Intelligence*, pages 1022–1029, 1993.

[39] D. A. Augusto and H. J. C. Barbosa. Accelerated parallel genetic programming tree evaluation with OpenCL. *Journal of Parallel and Distributed Computing*, 73(1):86–100, 2013.

[40] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions On Evolutionary Computation*, 5:17–26, 2000.

[41] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, 2000.

[42] U. Watchareeruetai, T. Matsumoto, N. Ohnishi, H. Kudo, and Y. Takeuchi. Acceleration of genetic programming by hierarchical structure learning: A case study on image recognition program synthesis. *IEICE Transactions*, 92-D(10):2094–2102, 2009.

[43] M. Zhang, U. Bhowan, and B. Ny. Genetic programming for object detection: A two-phase approach with an improved fitness function. *Electronic Letters on Computer Vision and Image Analysis*, 6(1):27–43, 2006.

[44] S. M. Gustafson and W. H. Hsu. Layered learning in genetic programming for a cooperative robot soccer problem. In *Proceedings of the 4th European Conference on Genetic Programming*, EuroGP '01, pages 291–301. Springer-Verlag, 2001.

[45] D. Howard and S. C. Roberts. A staged genetic programming strategy for image analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2 of *GECCO '99*, pages 1047–1052. Morgan Kaufmann, 1999.

[46] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pages 312–321. Springer-Verlag, 9-14 October 1994.

[47] N. AL-Madi and S. A. Ludwig. Segment-based genetic programming. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, GECCO '13 Companion, pages 133–134. ACM, 2013.

[48] J. Derrac, S. Garca, D. Molina, and F. Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[49] A. G. Bluman. *Elementary Statistics A Step by Step Approach*. McGraw-Hill Higher Education, 5th edition, 2004.

[50] W. Zhu, N. Zeng, and N. Wang. Sensitivity, specificity, accuracy associated confidence interval and ROC analysis with practical SAS implementations. In *Proceedings of the NorthEast SAS Users Group Conference NESUG10*, 2010.

[51] L. I. Smith. A tutorial on principal components analysis. Technical report, Cornell University, USA, February 26 2002.

[52] J. Shlens. A tutorial on principal component analysis, derivation, discussion and singular value decomposition. Technical report, 25 March 2003.

[53] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer New York, 2002.

[54] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[55] W. Banzhaf, S. Harding, W. B. Langdon, and G. Wilson. Accelerating genetic programming through graphics processing units. In *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, pages 1–19. Springer US, 2009.

[56] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press Cambridge, MA, USA, 1995.

[57] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, 2004.

[58] A. Verma, X. L. anf D.E. Goldberg, and R. Campbell. Scaling genetic algorithms using mapreduce. In *Ninth International Conference on Intelligent Systems Design and Applications (ISDA'09)*, pages 13–18, 2009.

[59] C. Jin, C. Vecchiola, and R. Buyya. MRPGA: An extension of mapreduce for parallelizing genetic algorithms. In *IEEE Fourth International Conference on eScience, eScience'08*, pages 214–221, 2008.

[60] A. W. McNabb, C. K. Monson, and K. D. Seppi. MRPSO: Mapreduce particle swarm optimization. In *Proceeding of the ninth annual conference companion on Genetic and evolutionary computation conference companion*, GECCO '07, 2007.

[61] B. Wu, G. Wu, and M. Yang. A mapreduce based ant colony optimization approach to combinatorial optimization problems. In *Eighth International Conference on Natural Computation (ICNC)*, pages 728–732, 2012.

[62] S. Harding and W. Banzhaf. Fast genetic programming on GPUs. In *Proceedings of the 10th European conference on Genetic programming*, EuroGP'07, pages 90–101, Berlin, Heidelberg, 2007. Springer-Verlag.

[63] *Khronos OpenCL Working Group. The OpenCL Specification, version 1.2*, 2012.

[64] (2012) Apache Software Foundation, Hadoop MapReduce [Online]. Available: http://hadoop.apache.org/mapreduce.

[65] A. Verma. Scaling simple, compact and extended compact genetic algorithms using mapreduce, thesis for the degree of master, university of illinois at urbana-champaign, 2010.

[66] H. Di-Wei and J. Lin. Scaling populations of a genetic algorithm for job shop scheduling problems using mapreduce. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 780–785, 2010.

[67] L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro. A parallel genetic algorithm based on hadoop mapreduce for the automatic generation of junit test suites. In *Proceedings of the IEEE Fifth International Conference on Software Testing, Verification and Validation*, ICST '12, pages 785–793, Washington, DC, USA, 2012. IEEE Computer Society.