Automated Design of Genetic Programming Classification Algorithms

Thambo Nyathi

Supervisor: Prof. Nelishia Pillay

This thesis is submitted in fulfilment of the academic requirements of Doctor of Philosophy in

Computer Science

School of Mathematics , Statistics and Computer Science

University of KwaZulu Natal

Pietermaritzburg

South Africa

December 2018

PREFACE

The research contained in this thesis was completed by the candidate while based in the Discipline of Computer Science, School of Mathematics, Statistics and Computer Science of the College of Agriculture, Engineering and Science, University of KwaZulu-Natal, Pietermaritzburg, South Africa.

The contents of this work have not been submitted in any form to another university and, except where the work of others is acknowledged in the text, the results reported are due to investigations by the candidate.

Date: **04-12-2018**

Signature
Professor Nelishia Pillay

Declaration

PLAGIARISM

I, **Thambo Nyathi**, declare that:

i) this dissertation has not been submitted in full or in part for any degree or examination to any other university;

ii) this dissertation does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons;

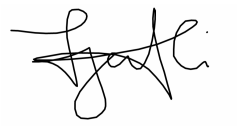
iii) this dissertation does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

   a) their words have been re-written but the general information attributed to them has been referenced;

   b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced;

iv) where I have used material for which publications followed, I have indicated in detail my role in the work;

v) this thesis is primarily a collection of material, prepared by myself, published as journal articles or presented as a poster and oral presentations at conferences. In some cases, additional material has been included;



Signature — Date: **04-12-2018**

Thambo Nyathi
December 2018

Declaration

I can confirm that this work was done under my supervision and it is the candidate's original work. As the candidate's supervisor, I have approved this thesis for submission.

Date: **04-12-2018**

Signature
Professor Nelishia Pillay

Thambo Nyathi
December 2018

Declaration

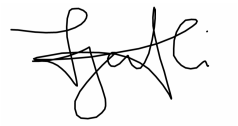
PUBLICATIONS

The following publications are associated with the research presented in this thesis:

1. Nyathi, T., Pillay, N.: *Automated design of genetic programming classification algorithms using a genetic algorithm.* In: European Conference on the Applications of Evolutionary Computation. pp. 224–239. Springer (2017)

2. Nyathi, T., Pillay, N.: *Comparison of a genetic algorithm to grammatical evolution for automated design of genetic programming classification algorithms.* Expert Systems with Applications 104(-), 213–234 (2018)

3. Nyathi, T., Pillay, N.: *Automated design of genetic programming classification algorithms for financial forecasting using evolutionary algorithms* In: International Conference on the Theory and Practice of Natural Computing (TPNC 2018) DOI: 10.1007/978-3-030-04070-3-16



———————————- Date: **04-12-2018**

Signature

Thambo Nyathi
December 2018

Acknowledgements

I would like to thank my supervisor, Professor Nelishia Pillay, who introduced me to the world of genetic programming and without her guidance and patience this journey would have been more rugged. My gratitude also extends to the Centre of High Performance Computing for allowing me access to their distributed computing architecture resources. I would also like to thank the National University of Science and Technology for affording me the time to embark on this journey.

A special thank you to my Bulawayo crew, without you nothing is worth it. To my late mother who always believed in the best of me and always saw the best in me. My Dad, a simple village boy from Kezi, Matopo who has always led by example from the front and the root of my life, who always wanted one of his offsprings to be an M.D, this will have to do.

Abstract

Over the past decades, there has been an increase in the use of evolutionary algorithms (EAs) for data mining and knowledge discovery in a wide range of application domains. Data classification, a real-world application problem is one of the areas EAs have been widely applied. Data classification has been extensively researched resulting in the development of a number of EA based classification algorithms. Genetic programming (GP) in particular has been shown to be one of the most effective EAs at inducing classifiers. It is widely accepted that the effectiveness of a parameterised algorithm like GP depends on its configuration. Currently, the design of GP classification algorithms is predominantly performed manually. Manual design follows an iterative trial and error approach which has been shown to be a menial, non-trivial time-consuming task that has a number of vulnerabilities. The research presented in this thesis is part of a large-scale initiative by the machine learning community to automate the design of machine learning techniques. The study investigates the hypothesis that automating the design of GP classification algorithms for data classification can still lead to the induction of effective classifiers. This research proposes using two evolutionary algorithms, namely, a genetic algorithm (GA) and grammatical evolution (GE) to automate the design of GP classification algorithms. The proof-by-demonstration research methodology is used in the study to achieve the set out objectives. To that end two systems namely, a genetic algorithm system and a grammatical evolution system were implemented for automating the design of GP classification algorithms. The classification performance of the automated designed GP classifiers, i.e., GA designed GP classifiers and GE designed GP classifiers were compared to manually designed GP classifiers on real-world binary class and multiclass classification problems. The evaluation was performed on multiple domain problems obtained from the UCI machine learning repository and on two specific domains, cybersecurity and financial forecasting. The automated designed classifiers were found to outperform the manually designed GP classifiers on all the problems considered in this study. GP classifiers evolved by GE were found to be suitable for classifying binary classification problems while those evolved by a GA were found to be suitable for multiclass classification problems. Furthermore, the automated design time was found to be less than manual design time. Fitness landscape analysis of the design spaces searched by a GA and GE were carried

out on all the class of problems considered in this study. Grammatical evolution found the search to be smoother on binary classification problems while the GA found multiclass problems to be less rugged than binary class problems.

Table of contents

List of figures

List of tables

List of Algorithms

Chapter 1

Introduction

1.1 Purpose of the Study

The research presented in this thesis tests the hypothesis that evolutionary algorithms specifically, a **genetic algorithm** and **grammatical evolution** can be used to automatically configure **genetic programming** classification algorithms for data classification. It is hypothesised that at the very least genetic programming classification algorithms automatically designed by a genetic algorithm and grammatical evolution are competitive when compared to human designed genetic programming classification algorithms. The effectiveness of the proposed approach is evaluated on a set of multiple domain problems and problems from specific domains. The research presented in this thesis is part of a large scale initiative by the machine learning community to automate the design of machine learning techniques. The overall aim is to remove reliance on the human expert, providing out of the box software that can also be used by novices.

Classification is considered to be a branch of data mining. Data mining is an area concerned with the extraction of knowledge from real-world data using computational algorithms. Classification may be described as a technique of assigning objects to classes based on a collection (dataset) of features describing those objects [87]. Most real-world problems may be viewed as classification problems. For example, in cybersecurity, intrusion detection systems should be able to classify a connection as malicious or non-malicious [170]. In credit scoring, there is a need to be able to classify a loan applicant as high risk or low risk [126]. Classification is usually performed by classifiers which are models that are induced by classification algorithms.

A significant amount of research has been carried out in the domain of classification resulting in the development of numerous classification algorithms [3, 171]. The application of evolutionary algorithms (EAs) [60] particularly genetic programming (GP) [120] to evolve

classifiers has also gained traction [64, 74]. Using GP as a classification algorithm presents a number of advantages. For example, the representation used by GP allows it to model different types of classifiers such as decision trees [113], discriminant functions [165] and classification rules [26] amongst others. Genetic programming is also capable of performing automatic feature selection as well as controlling the size of evolved classifiers.

It has been shown that like most EAs, the effectiveness of GP depends on its configuration [60, 136, 111]. There is no standard GP configuration for specific problems or problem domains. Different configurations work well for different problems or problem instances, therefore, finding the most effective configuration is a search process. The configuration and design of GP classification algorithms is still predominantly performed manually.

1.1.1 Manual Design

Normally, during the manual design of GP classification algorithms, a formal experimental design strategy is not followed. Manual design is usually performed using an iterative trial and error approach where for each parameter a range of values to be considered are pre-selected. Using a subset of the problem instances trial runs are then conducted adjusting one parameter value at a time with the values of the other parameters staying constant. The parameter value that achieves the best result is chosen as the value for that parameter. This process is done iteratively for all the parameters until all the parameters have values assigned to them. This is then considered to be the best configuration for the algorithm. A number of disadvantages have been identified with this approach. These are listed as follows:

a) *Since parameter values are evaluated iteratively the effect parameters have on each other is not taken into consideration* [100].

b) *The pre-selection of parameter values to consider limits the search space as better values may lie outside the boundary of considered values* [90].

c) *Using a subset of problem instances may give misleading results as different configurations work well with different problem instances* [98].

d) *The large search space of possible parameters leads to human designers making biased design decisions based on intuition and experience* [99].

e) *Because of the large search space, the manual design approach is considered to be a menial time-consuming task* [13].

f) *Design decisions made during manual design are not documented and justified making the reproducibility of experiments difficult* [138].

Although parameter tuning and control methods have been proposed for EAs in general [54, 58], however, none of the proposed methods have been universally adopted [8, 111]. Additionally, there is no research that has been carried out on the automated design of genetic programming classification algorithms.

1.2 Objectives

The major goal of this study is to investigate the feasibility of automating the design of genetic programming classification algorithms using evolutionary algorithms, namely a genetic algorithm and grammatical evolution. To achieve this goal the following objectives have to be met.

1. **To automate the design of genetic programming classification algorithms using a genetic algorithm.**
2. **To automate the design of genetic programming classification algorithms using grammatical evolution.**
3. **To compare the effectiveness of genetic programming classifiers evolved by a genetic algorithm to manually designed genetic programming classifiers.**
4. **To compare the effectiveness of genetic programming classifiers evolved by grammatical evolution to manually designed genetic programming classifiers.**
5. **To compare the performance of genetic programming classifiers evolved by a genetic algorithm to the performance of genetic programming classifiers evolved by grammatical evolution.**
6. **To compare the manual design configurations to the automated design configurations.**
7. **To compare the manual design time to the automated design time.**
8. **To compare the fitness landscape of the design space searched by a genetic algorithm to the fitness landscape searched by grammatical evolution.**

1.3 Scope of the Study

The major objective of the study presented in this thesis is to evaluate the use of evolutionary algorithms to configure genetic programming classification algorithms. The study is scoped as follows:

- Evolutionary algorithms
  The study is restricted to using a genetic algorithm and grammatical evolution to configure GP classification algorithms. The genetic algorithm approach for automating the design of genetic programming classification algorithms is described in **Chapter 5** while **Chapter 6** describes the grammatical evolution approach.

- Problem instances.
  The evaluation of the automatically designed GP classification algorithms is carried out by comparing the performance of automated designed classifiers to manually designed classifiers on the following problem instances:

  - binary and multiclass classification problems obtained from publicly available datasets.
  - binary and multiclass classification problems obtained from the cybersecurity problem domain.
  - financial forecasting problems.

- The evaluation and comparison is carried out with respect to training accuracy, testing accuracy, design time and evolved configurations.

- Genetic programming is a highly parameterised algorithm, therefore, there are numerous design decisions that need to be taken during the design of GP classification algorithms. The design decisions considered for automated design in this study are based on a survey of literature and these are outlined in section **5.2** of **Chapter 5**

- Fitness landscape analysis is carried on the design spaces evolved by the automated design approaches as opposed to the solution space. It is not possible to directly compare the fitness landscape of the manual design space as this is conducted manually by a human making selections.

1.4 Contributions

The main contribution of this thesis are two new approaches for the automated design of genetic programming classification algorithms. To the best of the author's knowledge there is currently no method that automates the design of genetic programming classification algorithms. The new approaches presented in **Chapters 5** and **6** make use of a genetic algorithm and grammatical evolution respectively to configure GP classification algorithms.

More specifically:

1. **Genetic programming classifiers evolved by genetic algorithm are shown to be suitable for classifying multiclass problems for the considered problem instances**. In sections **7.2.2** and **7.3** of **Chapter 7** it is shown that GP classifiers evolved by a genetic algorithm outperform those evolved by grammatical evolution and manual design on multiclass problems.

2. **Genetic programming classifiers evolved by grammatical evolution are shown to be suitable for classifying binary class problems for the considered problem instances**. In sections **7.2.1**, **7.3** and **7.4** of **Chapter 7** it is shown that GP classifiers evolved by grammatical evolution outperform those evolved by a genetic algorithm and manual design on binary problems.

3. **Automated design of genetic programming classification algorithms reduces the design time.** In section **7.5** of **Chapter 7** it is shown that the design times achieved by the automated design approaches is less than the manual design times.

4. **Grammatical evolution is shown to search a less rugged design space fitness landscape for binary classification problems while a genetic algorithm searches a less rugged landscape for multiclass classification problems**. A fitness landscape is considered to be rugged if no correlation exists between the distance of solutions and their fitness values. This is outlined in section **7.6** of **Chapter 7**

1.5 Thesis Layout

The rest of this thesis is organised as follows:

1.5.1 Chapter 2 - Literature Review

This chapter mainly provides background information as well as related work. The chapter firstly presents background information relating to the field of classification and the various

classification techniques. Evolutionary algorithms are then briefly introduced in general followed by a detailed description of genetic algorithms, genetic programming and grammatical evolution. An outline of the application of genetic programming as a classification algorithm is also presented. Finally related work using the proposed evolutionary algorithms for automated design is presented.

1.5.2 Chapter 3 - Methodology

This chapter initially provides a brief description of research methods in Computer Science. The chapter then provides details of how the objectives set out in **Chapter 1** will be met using the appropriate research methodology. The details of classification problem instances to be used to evaluate the proposed approach are also presented.

1.5.3 Chapter 4 - Manual Design of Genetic Programming Classification Algorithms

Chapter 4 presents a manual design of the standard genetic programming classification algorithm. A listing of parameter values for each set of problem instances considered in this study is also presented. The performance of the manual GP system presented in this chapter will be compared to the proposed approach.

1.5.4 Chapter 5 - Design of Genetic Programming Classification Algorithms using a Genetic Algorithms

Chapter 5 presents the automated design approach using a genetic algorithm. The design decisions for automated design are also presented in this chapter. The parameter settings for the genetic algorithm for the automated design are also presented in this chapter.

1.5.5 Chapter 6 - Design of Genetic Programming Classification Algorithms using Grammatical Evolution

This chapter presents the design of the automated design approach that uses grammatical evolution to evolve GP classifiers. The chapter also specifies the parameter settings for grammatical evolution used by the automated design system.

1.5.6 Chapter 7 - Results and Discussion

Chapter 7 provides the results of comparing the manual design approach to automated design. The performance is compared on classifier accuracy and design times. An analysis of the evolved configurations is carried out. The ruggedness of the design spaces evolved by the automated design approaches is analysed using autocorrelation analysis and the results are presented.

1.5.7 Chapter 8 - Conclusion and Future Work

Finally **Chapter 8** provides the conclusion and presents future work.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides background information and related work that lays the foundation for the proposed automated design approach presented in this study. The research presented in this thesis is part of a large-scale initiative by the machine learning community to automate the design of machine learning techniques. Furthermore, the desire is to automate certain tasks thereby freeing the human designer to attend to other tasks thus reducing the man-hours spent on the design process.

The first three sections of this chapter present background information specifically the following topics: classification is presented in section **2.2**, evolutionary algorithms in section **2.3** and section **2.4** presents GP and its application as a classification algorithm. Section **2.5** presents a survey of related studies that follow a similar approach to that presented in this thesis. Finally section **2.6** provides background information relating to methods of fitness landscape analysis.

2.2 Classification

Generally, classification is considered to be a supervised machine learning approach [225]. In supervised learning, the algorithm works with labelled data instances. Instance is the term used in machine learning, to refer to observations while the explanatory variables are termed features. The features are normally grouped together resulting in what is known as a feature vector. The possible categories to be predicted are referred to as classes. A labelled data instance is thus one in which a feature vector is given with its class. While the opposite i.e. class not given is referred to as unlabelled. Classification is a two-phase process consisting

of an induction (training/learning) phase and a deduction(testing) phase. During the training phase, a search algorithm is used to induce a model (classifier) from a collection of labelled data instances called a training set. In the testing phase, the induced classifier is evaluated by applying it to a collection of unseen labelled data instances called a test set. A search algorithm used to induce classifiers is commonly referred to as a classification algorithm [225]. Basically, a classification algorithm takes data instances from a training set as input and outputs a classifier. The outputted classifier is a mapping of the relationship between the attributes and classes of the training set. A classifier accepts unseen data instances from a test set as input and outputs a value which represents an evaluation metric of the classification process.

Different classification algorithms induce different classifier models [87]. In this thesis, we focus on classification rules (decision rules) and decision trees. Classification rules normally have an antecedent and consequent. The antecedent usually consists of a set of attributes and constants as input and a set of logical operators and/or mathematical functions that for each instance in the data set produce a score which then determines the consequent of the rule. The score can be an actual class or a value which represents a class [225]. A decision tree is a graph like structure (or flowchart) containing multiple interconnected nodes. Each node denotes a test on a feature value, each branching path represents an outcome of the test and tree leaves represent classes. A decision tree may be viewed as a collection of rules [87].

The number of classes contained in a dataset determines the type of classification. If there are only 2 distinct non-overlapping classes then the task is binary classification, however, if the number of classes is greater than two the task is a multiclass classification. Binary classification requires a classifier to predict whether values of an attribute describes one of two classes whereas in multiclass classification the number of possible classes is greater than two. Comparatively, multiclass classification is considered to be a more complex task than binary classification [213]. Generally, from a high level, two approaches are used to achieve multiclass classification. One approach is where the binary classifier with little or no modifications is extended to perform multiclass classification. Classification and regression trees (CART) [31] and C4.5 [179] are examples of classifiers that follow this approach to multiclass classification. The second and most popular approach is to decompose the problem into multiple binary classification problems. In this approach k classes are decomposed into k binary classification problems. A number of different methods have been proposed on how to perform binary decomposition and these are outlined in [9, 11].

Both binary and multiclass classification aim to classify unseen problem instances as accurately as possible. A number of performance metrics are available to measure the effectiveness of the evolved classifiers. These are presented in the next section.

2.2.1 Metrics of Evaluating Performance of Classifiers

The most common metric used for evaluating classifier performance is the predictive accuracy rate i.e. the number of correctly classified instances divided by the total number of considered instances, presented as a percentage $\pm$ the standard deviation. It has been shown that in certain situations this measure on its own may not be efficient and as a result, the following parameters are computed and widely used in coming up with various metrics [171].

a) The number of correctly classified instances (true positives t_p).

b) The number of correctly classified instances that do not belong to a class (True negatives t_n).

c) The number of instances incorrectly assigned to a class (false positive f_p).

d) The number of instances not recognised as belonging to a class (false negative f_n).

These parameters are usually presented in tabular form in what is known as a confusion matrix or contingency table [88] as illustrated in Table 2.1. Metrics formulated from the confusion matrix have been widely adopted for evaluating classifier performance.

Data class	Classified as Positive	Classified as Negative
positive	true positive (t_p)	false negative (f_n)
negative	false positive (f_p)	true negative (t_n)

Table 2.1 Confusion matrix

$$accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{2.1}$$

Equation **2.1** shows the formula for evaluating the commonly used predictive accuracy. As stated a number of researchers have been critical of the predictive accuracy metric and have pointed out certain flaws [225, 19]. A typical situation where accuracy may be flawed is in a case where the classes in a dataset are imbalanced. For example if a dataset with two classes X and Y and whose class distribution is given as 95% X and 5% Y. A classifier that concentrates on class X will have a 95% accuracy but there is no measure on how well it is able to classify Y. Baldi et al. [16] argue that evaluating a classifier using only accuracy

without considering other metrics available from the confusion matrix may lead to a loss of valuable evaluation information about the classifier. In concurrence, Marsland [145] states that accuracy, is a single numerical value that is evaluated using four components (t_p, t_n, f_n, f_p), as a result, there is a loss of information which can be extracted from different combinations of the four components.

$$specificity = \frac{t_n}{t_n + f_p} \tag{2.2}$$

Equation **2.2** defines specificity which is a ratio of instances classified as negative to the total negative instances in the dataset.

$$precision = \frac{t_p}{t_p + f_p} \tag{2.3}$$

Equation **2.3** defines precision which is a ratio of instances classified as positive to the actual positive class instance plus those falsely classified as positive.

$$recall(sensitivity) = \frac{t_p}{t_p + f_n} \tag{2.4}$$

Recall given by equation **2.4** is a measure of the ratio of instances classified as positive to the total positive classes in the dataset. It is also referred to as sensitivity or the true positive rate. A low recall value indicates high false negatives.

$$f_{measure} = 2(\frac{precision * recall}{precision + recall}) \tag{2.5}$$

Precision and recall are considered to be valid metrics individually [88] however, they may be combined to convey more information about a classifier and this is provided by the $f_{measure}$ given by equation **2.5**. Although Marsland further argues that the $f_{measure}$ does not convey any information about a classifier's ability to learn negative (t_n) examples it is still a widely adopted metric.

Each of the metrics outlined by equations **2.1** to **2.5** can be extended for multiclass classification as shown in [201, 88, 145]. A number of researchers have proposed weighted metrics based on the equations from the confusion matrix parameters. Bojarczuk et al. [25] used a product of precision and recall ($f = precision * recall$) as a metric to evaluate the effectiveness of classification rules on medical data while in [128] recall, precision and specificity are used in a weighted metric to evaluate classifiers for financial forecasting. Parameters from the confusion matrix can also be used to construct graphs such as the receiver operating characteristic (ROC) which is a plot of the true positive rate (t_p) against

false positive rate (f_p) [145]. Other metrics such as the root mean square error (RMSE) are also found in the literature although this metric is predominantly used to evaluate regression algorithms.

From the presented analysis there is no best way to evaluate any classifiers, but different metrics may give us different valuable insights into how a classification model performs.

2.2.2 Comparing Classification Algorithms

According to Dietterich [52] comparing classification algorithms is not an easy task, but it can be achieved indirectly by comparing the performance of the evolved classifiers. The *no free lunch theorem* [227] also applies in classification, as no one classification algorithm can outperform all other classification algorithms across all problem domains. However, it is still important to have a criteria to compare the performance of classification algorithms. Dietterich argues that if classification algorithms are to be compared the conditions of comparison need to be clearly outlined at the onset and the appropriate statistical tests selected. In agreement Salzberg [190] argues the comparison of classifiers should be performed in a statistical framework preferably using real-world data. According to Dietterich the comparison approach determines the evaluation methods to be used. For example, are two classification algorithms being compared on one dataset or multiple datasets, or are multiple classification algorithms being compared on one dataset or multiple datasets? A number of methods have been proposed and are used for evaluating two algorithms on multiple datasets. Among other, these tests include the z-test [68] and wilcox test [223].

In this thesis, multiple classification algorithms are to be compared on multiple datasets. The most widely used and recommended method for evaluating multiple algorithms on multiple datasets is the non-parametric Friedman test [76]. This test was initially proposed by Demšar [51]. Demšar recommends the Friedman test to evaluate the null hypothesis that all the considered algorithms perform the same. If the null hypothesis is rejected then a post-hoc test for pairwise comparison to establish the significance of the differences in performance is applied. The Friedman test is a non-parametric equivalent of the analysis of variance (ANOVA) test. In the Friedman test, the performance of each algorithm is ranked with the best performing algorithm ranked first the next best performing ranked second and so on. If algorithms tie then the ranks are shared between the tied algorithms. For example, if algorithm A and B tie for first place then positions (ranks) 1 and 2 are shared between them and they are each ranked 1.5. If we assume r_i^j to be the rank of the j^{th} algorithm of k algorithms on the i^{th} of N datasets. The Friedman test compares the average ranks of algorithms, $R_j = \frac{1}{N}\Sigma_i r_i^j$ and the expectation is that for all algorithms the average ranks should be equal, according to the null hypothesis. The Friedman statistic is given by:

$$\chi_F^2 = \frac{12N}{k(k+1)}\Big[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\Big] \tag{2.6}$$

and is distributed according to χ_F^2 with k-1 degrees of freedom. Iman and Davenport [102] showed the Friedman to be conservative and derived a better statistic based on the Friedman statistic as follows:

$$F_f = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{2.7}$$

this statistic is distributed according to the F-distribution k-1 and (k-1)(N-1) degrees of freedom. If the null hypothesis of similar performance is rejected a post-hoc test for pairwise comparison is carried out. If the difference in the average rank between two classifiers is greater than the critical difference then the differences in performance is considered to be statistically significant. The critical difference is evaluated as follows:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{2.8}$$

where critical values of q_α are based on the Studentized range statistic divided by $\sqrt{2}$. Demšar recommends the Nemenyi post-hoc test for general pairwise comparison, however, if there is a control algorithm then the Bonferroni-Dunn test is more suitable.

2.2.3 Datasets

Data from real-world problems is often unstructured and may require processing to enable it to be structured into a format that a classifier can interpret. A data instance is made up of several features each of which is either categorical or continuous. Categorical features include nominal, binary and ordinal types while continuous may refer to integer, inter-scaled or ratio scaled features [30]. A dataset may contain a mix of categorical and continuous features. Some classifiers due to their functionality work well with categorical data in which case numerical data has to be converted to categorical data a process known as discretisation.

2.2.3.1 Discretisation

According to Garcia et al. [77] discretisation is a process that converts data from a quantitative state to a qualitative state. Discretisation transforms a continuous feature into a discrete feature with a fixed number of non-overlapping intervals. Liu et al. [134] argue that most of the classifiers proposed in the domain of classification require discrete features. The general advantages of discretisation highlighted in the literature [77, 56, 115, 116] include:

- improved classification accuracy.
- reduced algorithm execution time.
- possible elimination of noise in the data.
- compact and shorter results.
- easier interpretation and understanding of data by practitioners and users.

However, it is also argued that the process of discretisation results in a loss of information [56]. Therefore, prevention or the minimisation of the loss of information is a major objective of a discretisation method. In a recent survey study, Garcia et al. [77] identified more than eighty discretisation methods and they recommend the following characteristics to be considered when selecting a discretisation method:

a) *number of features* - an effective discretisation method should be able to reduce continuous features to as few as possible discrete levels without loss of information.

b) *inconsistency* - there should no inconsistencies after discretisation such as duplicate instances having different classes.

c) *predictive accuracy* - after discretisation the predictive accuracy obtained from the discrete dataset should not be less than what can be predicted from the numerical dataset. Chmielewski and Grzymala-Busse [39] describe this as the most important feature of a discretisation method.

d) *time requirements* - the process of discretisation should not be time-consuming.

A number of classifications taxonomies have been proposed for discretisation methods but what is generally accepted is that at a high level they fall under *unsupervised* vs *supervised* [115, 134, 121].

Unsupervised discretisation methods discretise a feature based on the distribution of values of that feature. *Equal interval width* (EIW) and *equal frequency interval* (EFI) [225] are examples of unsupervised discretisation methods. EIW is one of the simplest discretisation methods to implement, it divides the range of observed values of a feature into l equal sized bins, where l is user-defined. If V_1 is the least value of a feature and V_2 the maximum value the width of intervals is given by $(V_2\text{-}V_1)/l$. It has been shown that the EIW method has a weakness in that it can distribute instances unevenly as some bins may be empty while others have been filled [225]. EFI is a similar method to EIW. Assuming we have m instances in a feature each of l bins will have m/l instances. Unsupervised discretisation

methods do not take into consideration the class labels during discretisation and this has been identified as a weakness as information loss may occur [225].

Supervised discretisation methods take the class into consideration. The ChiMerge [112] method is an example of one of the most widely used supervised discretisation methods. The ChiMerge method uses the χ^2 statistic to evaluate if the relative class frequencies of adjacent intervals are statistically different. Given a continuous feature, the first step of the ChiMerge method is to sort the numerical values of the continuous feature (and their corresponding classes) into an ascending order. Then a frequency table outlining the number of occurrences of each distinct value of the feature for each possible classification is constructed. The distribution of the values of the feature within the different classes is used to generate a set of intervals. Then using the χ^2 statistic the statistical significance of the differences of the intervals is evaluated. If there is no statistical significance the intervals are merged otherwise the intervals are maintained separately. A detailed outline of the ChiMerge algorithm is provided in [112].

According to Garcia et al. conclusions on which is the best discretisation method cannot be drawn. In a study to evaluate the induction of decision trees using hyper-heuristics Vella et al. [216] use the *equal frequency interval* method the justification of the selection of this method is not provided. Lim and Lee [131] use the *equal interval width* discretisation strategy for a proposed online classification algorithm. Witten et al. [226] propose a classification tool which can perform *equal interval width* pre-processing. In [23] *equal interval width* is used for the pre-processing of datasets used to evaluate classifiers for software error classification. In [225] it is argued that for medium to small datasets the *equal frequency interval* method can be effective. Liu and Setiono [135] propose an approach that extends the ChiMerge method and demonstrate how it improves the predictive accuracy of a classifier. In [56] a study that compares the performance of supervised discretisation to unsupervised discretisation is presented and the results are reported to show a marginal difference in performance. From the studied literature the choice of discretisation method to use seems to depend on the experience and preference of the researcher.

2.2.3.2 Normalisation

Before the data is discretised it is often preferable to have the data normalised. Normalisation is the process of scaling data to within a certain range usually 0-1 [30]. This is usually necessary when a feature exhibits high numeric variation in its values. For example, a feature containing values for distance travelled may have a minimum value of 10 kilometers and a maximum value of 5000 kilometers. To overcome this problem normalisation is performed.

Equation **2.9** outlines the most commonly used formula for normalisation also referred to as *min-max* scaling [30].

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2.9}$$

All the features or some of the features on the dataset can be normalised. In [169] a study is presented to evaluate the effectiveness of normalisation on the classification accuracy. In the study classifiers are applied to datasets before normalisation and after normalisation. The predictive accuracy of the classifiers on datasets after normalisation is found to be higher than before normalisation. Wu et al. [229] compare the performance of three classification algorithms using normalised data from the medical domain specifically ovarian cancer. An argument is put forward that data normalisation has no impact on the final classification results as the effect, if any, of normalisation will affect the three classification algorithms equally. Gunn et al. [85] argue that the necessity of normalisation depends on the variations in the values of the feature(s) and although normalisation may not be necessary for all cases there is no harm in normalising all the features of a dataset.

2.2.3.3 Partitioning

As mentioned a classifier is evolved from a training set and evaluated on unseen instances which constitute a test set. A complete dataset has to be partitioned into a training set and test set. The three commonly used strategies are *holdout validation, k-fold cross-validation* and *N-fold cross-validation* [30].

In the holdout validation strategy, the dataset is randomly split into two subsets, a training set and a test set. There is no standard value for the splitting ratio used to partition a dataset, but the norm is that the training set contains a greater number of examples than the test set. According to Bramer [30] partition ratio values of 90:10(90% training set and 10% test set),80:20,70:30 and 60:40 are acceptable. Barros et al. [19] used a 70% training to 30% testing ratio to partition a dataset containing microarray gene expression data. There are some cases where the test set has a greater or equal number of examples as in the training set. Ye and Keogh [231] describe a decision tree type of classifier which is applied to datasets that contain data from spectography measurements. The proposed multiclass decision tree classifier is evaluated on the following data: a wheat dataset which is split into a 5% training and 95% testing and a coffee dataset which is split to 50% training and 50% testing. The holdout strategy is considered to have an advantage of being simple to implement and since it is performed independently of algorithm execution it does not impose a computational cost on the classification algorithm. However, if the dataset contains imbalances in classes i.e. if

there are few instances of a particular class and they are included in the test set and not in the training set the classifier is bound to perform poorly as it will fail to generalise on instances it has not been trained on. Real-world problems usually contain these imbalances in classes, therefore, partitioning using the holdout strategy has to be performed with care. Engen [62] uses holdout validation to partition a dataset containing intrusion detection data. The dataset is split using an 80% training and 20% test ratio. Due to the imbalances in intrusion classes the selection of data for each partition is performed chronologically ensuring that each class is equally represented in each partition.

In the *k-fold cross-validation* strategy, the dataset is partitioned into k equal parts. If the number of instances of the dataset is not totally divisible by k the k subset will contain the smaller number of instances than the other k-1 subsets. k runs are performed where on each run k-1 subsets are used for training while the k subset is a test set. To evaluate the classification, the number of correctly classified instances in all the runs is divided by the total number of instances. According to [30] typical values of k lie in the range of 5 -10. Although the commonly found k values found in the literature are 5 [2, 229] and 10 [47, 27] a number of studies also use 2-fold cross-validation [10]. De'ath and Fabricius [27] describe an approach which uses classification and regression trees for the analysis of ecological data. In their approach, they use 10-fold cross-validation and argue that this approach eliminates over-fitting. Over-fitting also known as over-training is when a classifier learns exactly how to fit the training set, this usually leads to poor generalisation [225].

N-fold cross-validation functions in a similar manner as *k-fold cross-validation*. The dataset is divided into as many parts as there are instances, meaning at some stage each instance will be a test set, hence this technique is also known as leave one out. Bramer is critical of the computational demands of this method and argues that it is suitable for very small datasets where there is limited data for training. Furthermore, he asserts that the performance of this validation method compared to *k-fold cross-validation* has not been shown to be significantly better. Friedman et al. [75] assert that 5-fold and 10-fold cross-validation are better than the *N-fold cross-validation* technique. De'ath and Fabricius disagree with this assertion, they conducted a study to compare the 5-fold, 10-fold and N-fold validation methods and their results show the methods are comparable as no one method performed significantly better than the others. From the literature considered, it seems to appear that the selection of which dataset partitioning method to use is an arbitrary choice based on the preference of the researcher. There is no clear justification of the selections although in some instances there are references to general advantages or disadvantages of the partitioning methods.

2.3 Evolutionary Algorithms

Evolutionary computation is a sub-field of artificial intelligence which models the Darwinian [43] principle of natural selection as an inspiration for the design of computational methods. Algorithms that follow the principle of natural selection are generally referred to as Evolutionary Algorithms (EAs) [60]. They operate on a collection of candidate solutions for some problem. Each candidate solution is represented as an individual and a collection of individuals is called a population. Each individual of the population is assigned a quality value known as a fitness which is evaluated by a function called a fitness function. The fitness function measures how well a candidate solution is at solving the problem being considered. Evolutionary algorithms are iterative and each iteration is called a generation. Algorithm **1** is pseudo-code of a generic EA.

Algorithm 1 Generic Evolutionary Algorithm

```
1: BEGIN
2:    INITIALISE population with random individuals
3:    EVALUATE each individual;
4:    while termination condition not met DO
5:       SELECT parents
6:       GENETIC MANIPULATION of parents
7:       EVALUATE new individuals
8:       SELECT individuals for the next generation
9:    end while
10: END
```

From Algorithm **1** an EA generally operates as follows: as a first step, a population of individuals is *initialised* before evolution can take place. The initial population is normally randomly generated. The next step is to *evaluate* the fitness of each individual using a fitness function. A predefined termination condition is checked. If the termination condition is met the algorithm outputs the best solution for the problem and terminates. If the termination condition is not met a *selection* method is used to choose individuals from the population to undergo genetic manipulation. The selection method tends to be biased towards fitter individuals. The selected individuals are used to seed the next generation of individuals. This is achieved through the application of genetic operators (crossover and/or mutation) to the selected individuals (parents). The fitness of the resulting individual(s)(offspring) is evaluated. A population replacement strategy is then used to update the population. This process continues iteratively until a stopping criterion is met.

According to [60] variation operators (crossover and mutation) and selection are the basis of EAs. A number of EAs exist that share similarities, however, Evolutionary Strategies(ES) [182], Evolutionary Programming(EP) [70], Genetic Algorithms, Genetic Programming and Grammatical Evolution are considered to be the primary members of the EA family.

In this thesis, the aim is to automatically design GP classification algorithms using a GA and GE, therefore, our focus on EAs will be limited to GA, GE and GP. The following sections review genetic algorithms, genetic programming and grammatical evolution. The aim of the reviews is not to exhaustively enumerate all the available information about the approaches but to provide relevant background information for the research carried out in this thesis.

2.3.1 Genetic Algorithm

Individuals in a GA population represent possible solutions to a problem being solved. An individual is normally encoded as a fixed length linear chromosome where each gene of the chromosome is a binary bit string. Other forms of encoding besides binary bits have been proposed, such as in [40, 93] where real numbers are used. Eshelman and Schaffer [63] present three advantages of using real number encoding over binary. Firstly, they argue that real numbers provide a more accurate and precise representation of real-word problems than binary numbers, secondly, real numbers provide a wider range than the power of two provided by binary encoding. Thirdly, real numbers can represent slight or gradual changes better than the binary number system. Non-numeric encoding is also widely used as shown in [42] where characters from the alphabet are used to encode a GA chromosome which represents an amino-acid protein. Variable length chromosomes have also been proposed and are commonly used as presented in [4].

The first step of a basic genetic algorithm is to randomly create an initial population of individuals. The fitness of each individual in the population is then evaluated. A selection method is used to select individuals from the current population to act as parents and to undergo crossover and mutation to create offspring for the next generation. The population is then updated. This process continues iteratively until a predefined stopping criteria is met and the algorithm terminates. The best solution based on fitness is returned. Algorithm **2** is an outline of the steps of a typical GA algorithm.

Algorithm 2 Genetic Algorithm

```
Create initial population
Calculate fitness of all individuals
    while termination condition not met do
        Select fitter individuals for reproduction
        Recombine individuals
        Mutate individuals
        Evaluate fitness of all individuals
        Generate a new population
    end while
return best individual
```

Since the initial proposal of GAs by Holland, a number of variants have been proposed and are found in the literature [80]. However, according to [150] most implementations of a GA follows the flow of Algorithm **2**. A detailed algorithmic flow is presented as follows.

2.3.1.1 Initial Population Generation

At the initialisation step, individuals are randomly created. The value of each gene is randomly selected from a range of possible values for that gene determined by the encoding scheme. The number of chromosomes created (population) is determined by a population size parameter. The encoding used depends on the problem being solved as each individual in a GA represents a candidate solution. Figure **2.1** shows three examples of typical GA

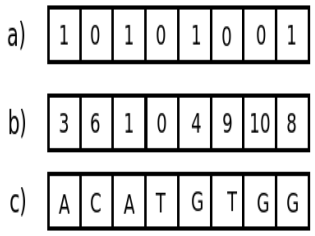


Fig. 2.1 Examples of GA individuals

individuals. Each individual is an example of a specific encoding scheme. The example denoted a) illustrates the binary encoding scheme where each gene can be a value of either be 1 or 0. Example b) is for the real number encoding scheme and c) shows encoding an individual using characters from the alphabet. The range of possible values for each gene is specified before initial population generation to enable each randomly created individual to be a valid candidate solution of the problem being solved. For example in the Travelling

Salesman Problem (TSP)[183] a chromosome can represent a route and each gene can be a city. A chromosome can also be encoded in such a way that each gene uniquely represents a different element of the problem. For example in [204] where a GA is used to design a metaheuristic to solve a TSP problem, each gene of the chromosome is a specific design component of the metaheuristic.

2.3.1.2 Fitness Evaluation

Fitness is a measure of the quality of each individual's ability to solve the problem at hand. A fitness function is used to evaluate the fitness of an individual. The fitness function is normally predefined. It can be a mathematical function or some measure of how well a GA individual is at solving the specified problem. DeJong and Spears [48] describe a GA approach for data classification that uses predictive accuracy as the fitness function. Miller et al. [147] propose to use a GA to design an artificial neural network (ANN) [86]. The performance measure obtained from evaluating the ANN is assigned as the fitness of the GA individual which specified the design of that ANN.

2.3.1.3 Selection

A selection method is used to select individuals from a population. According to Goldberg [79] the two commonly used selection methods for a GA are fitness proportionate and tournament selection.

Fitness proportionate selects individuals with a probability that is directly proportional to their fitness values [191]. The selection of an individual proceeds as follows:

i) evaluate the probability, p_i of selecting each individual in the population:

$$p_i = \frac{f_i}{\sum_{k=1}^{n} f_k} \tag{2.10}$$

where n is the population size and f_i is the fitness of an individual.

ii) calculate the cumulative probability, q_i for each individual using the following equation:

$$q_i = \sum_{k=1}^{i} p_k \tag{2.11}$$

iii) choose a uniform random number *rand* between 0 and 1.

iv) if *rand* $<q_1$ the first individual is selected or else the individual x_i such that q_{i-1} <*rand* <= q_i is selected.

v) Steps iii) and iv) are repeated n times to create n candidates in the mating pool.

An individual is randomly selected from the mating pool.

Tournament selection follows a simpler approach. A fixed number of individuals t are randomly selected from the population and the individual with the best fitness from the t individuals is returned.

According to Whitley et al. [222] fitness proportionate selection may lead to a GA experiencing premature convergence if the population of the GA contains individuals with a very high fitness. The authors point out that these *super-fit* individuals will dominate the mating pool and after a number of generations the population will constitute mainly of genetic material from those individuals. Tournament selection appears to be favoured due to its operational simplicity and it is quicker to execute than fitness proportionate [58].

2.3.1.4 Crossover

Crossover is a convergence operator with an objective of directing the population to a global maxima/minima. Crossover is considered as the primary genetic operator for a GA [1]. In crossover usually, two individuals are selected using a selection method to act as parents which exchange elements of their chromosomes before and after a randomly selected crossover point(s) to create one or two offspring. The likelihood of crossover occurring is determined by a crossover probability p_c. A random number r is chosen between [0,1] and if r is greater than p_c then crossover takes place. The objective of a GA is to achieve convergence, therefore, crossover is usually applied at higher rates. A number of crossover approaches have been proposed as outlined in [61, 158, 205]. In this thesis, we consider the three widely used crossover operators namely, one-point crossover, two-point crossover and uniform crossover.

one-point crossover is also referred to as single point crossover. In this crossover operator a point is randomly selected along the chromosomes (parents) and the tails are exchanged to form two offspring as illustrated in figure 2.2.

two-point crossover is an extension of one-point crossover where two crossover points are randomly selected. Each parent chromosome is broken into three segments of contiguous

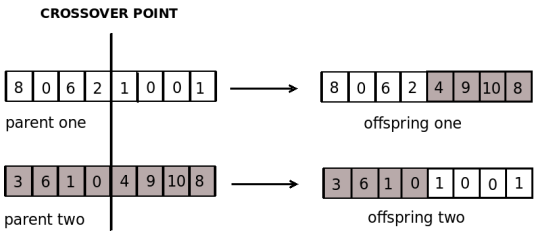


Fig. 2.2 One point crossover

genes and the offspring are created by taking alternative segments from the parents. This crossover method is illustrated in Figure **2.3**.

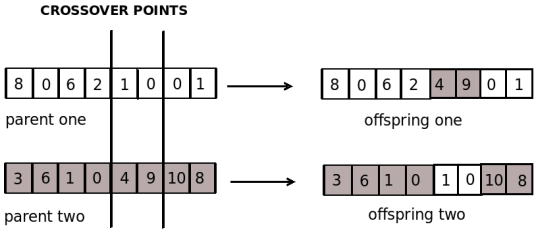


Fig. 2.3 Two-point crossover

Uniform crossover considers genes individually based on a probability known as a swapping probability [208]. Given two individuals, parent 1 and parent 2 and assuming the swapping probability is set as 0.5 the crossover proceeds as follows: a random number r is chosen from a uniform distribution over [0,1]. If r is equal or greater than 0.5 the value of the first gene of offspring 1 is assigned the same value as the first gene of parent 1 and the first gene of offspring 2 is assigned the same value as the first gene of parent 2. If the value of r is less than 0.5 the value of the first gene of offspring 1 is assigned the value of gene 1 of parent 2 and the value of the first gene of offspring 2 is assigned the same value as the first gene of parent 1. This process is repeated until all the genes of the offspring chromosomes

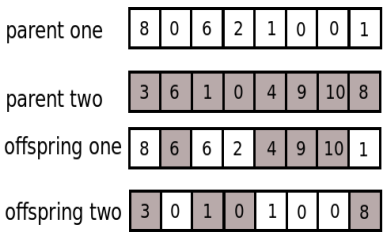


Fig. 2.4 Uniform crossover

are assigned values. Figure **2.4** illustrates uniform crossover.

According to [60] the choice of which crossover method to use for a GA depends on the researcher and the suitability of the method for the problem area as no one crossover method is better than other methods for all problems. For example, both one-point and two point crossover tend to populate offspring with genes that are closer together. This characteristic would be suitable for the vehicle routing problem if genes values represent cities, closer cities may be kept together which may lead to quicker convergence. Uniform crossover tends to populate each individual with one half of the genes from one of the parents and the other half of the genes from the other parent. This makes uniform crossover more suitable for problems where the value of the genes represent independent entities of a solution. It is important that the selected crossover method produces valid candidate solutions (offspring) otherwise a problem specific repair mechanism may be required to further process the offspring and this naturally leads to an extra overhead in-terms of processing time. Other crossover operators have been proposed [150, 191, 205].

2.3.1.5 Mutation

Mutation is applied to a single individual selected using a selection method. Mutation is used to provide diversity within a population and involves making small random changes in the chromosome resulting in a new individual. Mutation plays an important role in the prevention of the search from prematurely converging to a local minima/maxima [150]. The application rate of mutation given by p_m is normally less than the crossover rate. A very high mutation rate may result in the search becoming a random search.

A number of researchers have reported on different mutation operators such as swap mutation [49], inversion mutation [7] and scramble mutation [44], however, the most widely used mutation method is bit flip mutation [60].

Bit flip mutation is used in binary encoded GAs and involves considering each gene for mutation individually based on the mutation probability p_m. A number r between [0,1] is randomly generated and compared to the value of p_m. If r is equal to or greater than p_m then the value of the gene under consideration is flipped. If the value of the gene is 1 then mutation will change the bit to a 0 as illustrated in Figure **2.5** where the values of the third and fifth genes are mutated from 1 to 0 and 0 to 1 respectively. Bit flip mutation can be extended for other forms of encoding where the new value of the gene to be mutated is obtained from a set of valid values for that gene.

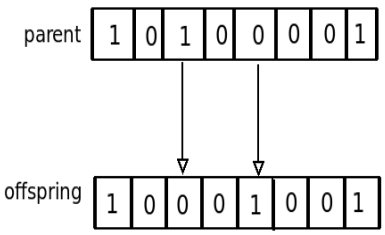


Fig. 2.5 Bit flip mutation

2.3.1.6 Population Replacement

The two commonly used population replacement methods for GAs are generational and steady-state. In generational replacement, the current population is replaced by a new population generated from crossover and mutation. Steady-state replaces a specific number n of individuals of the current population with n offspring. Sastry et al. [191] assert that the generational approach is preferable because it is simpler to implement and has less computational overheads than the steady-state approach. The overheads in the steady-state approach arise from having to determine the optimal value of n. Additionally, a selection method has to be used to select the individuals to be replaced. Elitism which involves copying a specified number of the fittest individuals from the current population into the next generation is also considered to be a replacement strategy [60].

2.3.1.7 Termination

The algorithm evolves from generation to generation until a termination criteria is met. Genetic algorithms are stochastic and there is no guarantee that the algorithm will either converge or find the optimal solution. The termination criteria can be set to be a maximum number of generations or a near optimal solution.

Complete execution of a genetic algorithm from population initialisation to executing the specified number of generations and outputting the solution is known as a run. Due to the stochastic nature of evolutionary algorithms, a number of runs have to be performed. Each run normally uses a different random number generator seed.

2.3.1.8 Applications of GA

Genetic algorithms have been applied in numerous problem areas, such as classification [106], design of experiments [218], function optimisation [32] feature selection [177] amongst others including designing other evolutionary algorithms [53].

2.3.2 Genetic Programming

Genetic programming is an EA that explores a program space. The concept of evolving software programs using evolutionary algorithms was presented by Cramer [41] in 1985 and in 1992, Koza [118] proposed Genetic Programming. Genetic programming is viewed as an extension of genetic algorithms [17]. In GP an individual is a computer program and the hope is that through evolution of the population of programs, fitter programs can be evolved until a program that provides an (near) optimal solution is generated.

The basic approach of a GP algorithm is to initially create a population of randomly generated programs. Each program is constructed from building blocks needed to solve the problem GP is being applied to. The fitness of each randomly generated program is then evaluated. If a specified termination criteria is not met good programs are then selected to act as parents for the generation of new programs. New programs are generated by applying genetic operators to parent programs and their fitness is evaluated. The process of selecting good programs and applying genetic operators to them is repeated until a stopping criteria is met and the best program is outputted. Therefore, unlike a GA which searches for a solution to a problem at hand in a solution space GP conducts a search in a program space for a program to solve a problem at hand.

In GP, programs (individuals) are traditionally represented as syntax trees which can be converted to their corresponding executable expressions usually in prefix notation [124]. Each node of the tree is considered to be a gene. Figure **2.6** is an example of a syntax tree.

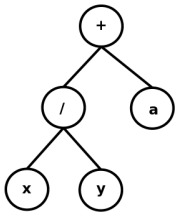


Fig. 2.6 GP syntax tree

The internal nodes of a syntax tree are known as functions while the external nodes are known as terminals. Functions and terminals are constructed from elements of a function and a terminal set respectively. Banzhaf et al.[17] describe a function set as constituting of application specific operators and a terminal set as constituting of inputs to the GP program for the problem being considered. The authors point out that functions and terminals are problem dependant primitives from which GP programs are constructed. The operators in a function set are defined with an arity, which represents the number of inputs a specific operator requires. The function set may include the following:

- arithmetic operators {+, -, *, / }.
- mathematical functions{sine, cosine, sqrt}.
- logical operators { AND, OR, NOR, NOT}.
- equality operators { <, >, <=, >=, == }.
- user defined domain specific operators.

Koza [120] describes two important properties that a function set and terminal set must satisfy, these are the closure property and sufficiency property. The closure property requires that any output from any of the functions / terminals must be a valid input for all the other functions. To meet this property the functionality of some operators are modified. An example of this is the arithmetic divide operator. Its functionality is modified to enable division by zero to be performed [17]. The common modification is to have it return a 1 or 0 when a division by 0 operation occurs. This is known as protected divide. The sufficiency property requires that elements of the function and terminal sets should be able to represent the solution. Other forms of representations are also found in the literature such as linear GP [29] and graph-based GP [148].

Algorithm **3** is a step by step outline of the GP algorithm.

Algorithm 3 Genetic Programming

```
1: Create an initial population of programs
2: Execute each program and establish the fitness
3:   while termination condition not met do
4:     Select fitter programs to participate in reproduction
5:     Create new programs using genetic operators and update the population
6:     Execute each new program and establish the fitness
7:   end while
8: return best program
```

The complete execution of the algorithm from start to finish is known as a run. A detailed algorithmic flow is presented as follows.

2.3.2.1 Initial Population Generation

Like most EAs, initial population generation of GP programs is performed randomly. The number of programs generated is specified by a user-defined *population size* parameter. To

create a program, a function for the root node is randomly selected from the function set and the rest of the tree is recursively constructed with the leaves of any new nodes filled until the *maximum tree depth* is reached. Maximum tree depth is defined as the number of nodes between the end node of a tree and the root node[17]. The value of this parameter is user-defined. Koza [119] proposes three methods for initial tree generation namely, full, grow and ramped half-and-half. The full method constructs trees in such a manner that all the nodes up to a depth of (*maximum tree depth - 1*) are functions and a depth equal to the *maximum tree depth* are terminals. The grow method creates trees of variable length. Nodes between the root and (*maximum tree depth - 1*) may randomly be assigned as functions or terminals and those at the *maximum tree depth* set as terminals. The ramped half-and-half method combines the full and grow method. At each depth from a depth of two, it creates half of the trees using the full method and the other half using the grow method up to the *maximum tree depth*. At each depth, an equal number of trees is created. The ramped half-and-half creates an initial population of varied shapes and sizes with the hope of increasing diversity [119].

Other methods have been proposed for initial population generation for GP as outlined by Poli et al. [175] and Luke and Paniat [141], however, the three methods originally proposed by Koza [119] are still predominantly used with the ramped half-and-half method being the most widely used [64]. Once an initial population has been created the fitness of each individual is evaluated.

2.3.2.2 Fitness Evaluation

The effectiveness of a GP program is measured using a fitness function which is normally problem dependant [17]. Each program is applied to a training set (fitness cases) and the result is assigned as the fitness of the program. Fitness can be measured in a number of ways. For example, it can be a measure of how close a program output is to a desired outcome such as the accuracy rate in classification [64] or a measure of how quick a GP program is at providing a solution [175]. A fitness function may be used to measure a single objective or multiple objectives. For example, Li [127] uses a fitness function which combines several objectives in applying GP for financial forecasting. The measured output is assigned to the program as its fitness.

2.3.2.3 Selection

As in other evolutionary algorithms selection in GP is biased towards fitter individuals. A number of selection methods are found in the literature but the two commonly used selection

methods are tournament selection and fitness proportionate selection [175]. These methods are applied as outlined in section **2.3.1.3**.

2.3.2.4 Genetic Operators

In GP crossover and mutation are the commonly used genetic operators. Crossover promotes convergence and is considered to be a local search operator since it combines genetic material that is already existing between two parents [17, 175]. However, unlike crossover, mutation is a global search operator which does not promote convergence but promotes diversity as it introduces new genetic material to an individual. In crossover, two programs selected as parents exchange code to create two new programs while in mutation a random change is made to a selected parent program. Poli et al. [175] point out that copies of parents are used to avoid making alterations to the parents as they stand a chance of being selected again, so they have to maintain their unaltered state. A number of crossover and mutation operators have been proposed and some are outlined in [175], however, in this section the review is restricted to subtree crossover, shrink mutation and grow mutation.

Subtree crossover is the most commonly used crossover operator in GP [124]. Koza [119] describes this form of crossover as an exchange of subtree branches between parents. Two parents are selected from the population using a selection method. A random crossover point is selected on each parent. Subtrees rooted at the randomly selected crossover points are known as crossover fragments. The two fragments are exchanged from one parent to the other thus creating two offspring. An example of this operation is shown in Figure **2.7.** A random point is selected in a copy of parent 1 resulting in fragment 1 and the same procedure is followed on a copy of parent 2 resulting in fragment 2. The two fragments are exchanged i.e. fragment 1 is inserted at the crossover point of parent 2 and fragment 2 is inserted at the crossover point of parent 1. The size of the offspring must not exceed a user-defined *offspring depth* limit. A number of methods exist in the literature [78, 180] for dealing with offspring that exceed *offspring depth* with the most widely used method being pruning [175].

Mutation is considered as an important operator for increasing diversity in a population during evolution [17].

Poli and Langdon [176] describe **shrink mutation** as an operator that replaces a randomly selected subtree with a randomly created terminal node. **Grow mutation** works by randomly selecting a terminal and replacing it with a subtree [12]. Figure **2.8** is an example of grow mutation where the highlighted terminal node is replaced by the subtree.

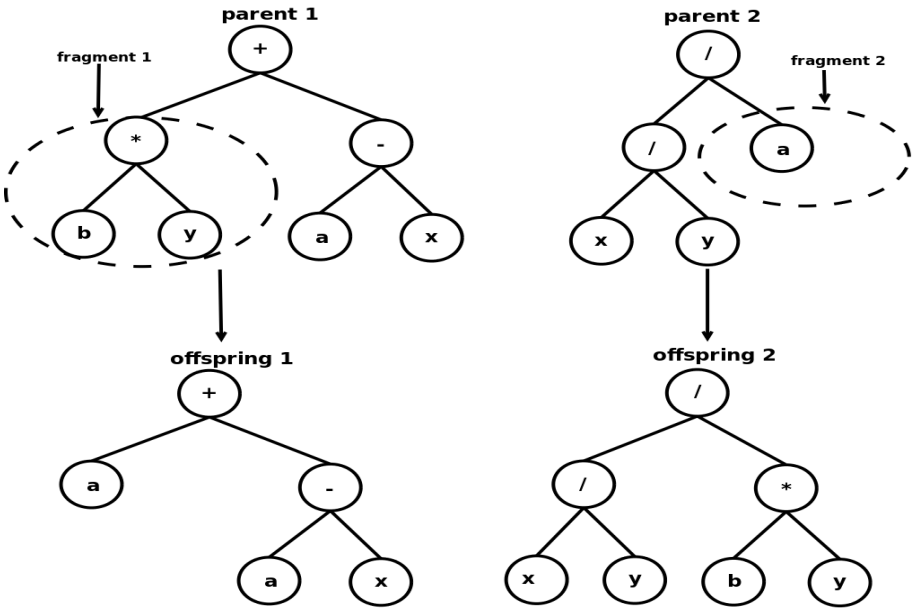


Fig. 2.7 GP subtree crossover

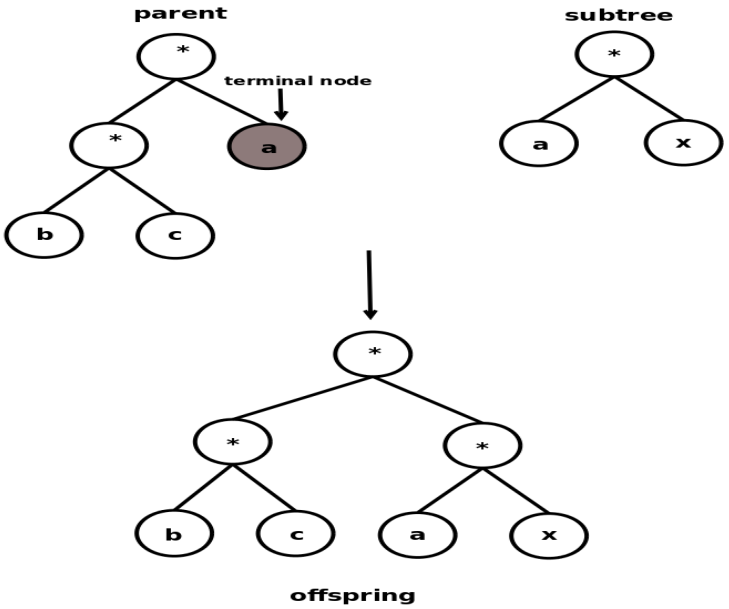


Fig. 2.8 Grow mutation

Grow mutation has the effect of increasing the size of a tree. The resultant offspring needs to conform to the specified offspring depth in a similar manner to the offspring resulting from crossover. Therefore, a parameter known as *mutation depth* is specified for a GP system. The purpose of this parameter is to control the size of a subtree created by a mutation operator.

The population size of the new generation is the same size as the initially randomly generated population. Part of the new population is evolved by crossover and the other part by mutation. The number of individuals created by crossover and those created by mutation are determined by the application rates parameters and they should sum up to the population size. The application rates are known as the *crossover rate* and the *mutation rate*. There are instances where some members of a current population are passed onto the next population this is known as reproduction and it may also be applied at a specified rate known as the *reproduction rate* [17].

2.3.2.5 Population Replacement

The most frequently used population replacement methods are generational and steady-state. These methods are implemented in the same manner as described in section **2.3.1.6**.

2.3.2.6 Termination

Two termination conditions are usually used in GP, a maximum number of generations or a problem specific solution is met. The maximum number of generations is a user-defined parameter and is specified before a run. According to Poli et al. [175] typical values lie in the range of 10 to 50 generations. However, other researchers insist that a balance has to be struck between the number of generations and the population size to achieve convergence [124]. A larger population size may require fewer generations or a smaller population size may require more generations to achieve convergence. What is clear is there are no optimum values for the parameters and most have to be arrived at through parameter tuning.

2.3.2.7 Applications of GP

The flexibility of GP enables it to be applied to a wide range of real-world problems. For example, GP has been successfully used in cyber-security [89, 203], bio-informatics [109, 188], medical domain [122] and text mining [103, 105] amongst a number of problems domains.

2.3.3 Grammatical Evolution

O'Neill and Ryan [166] describe grammatical evolution as an evolutionary algorithm capable of producing code in any language, providing a Backus Naur Form grammar which describes the output language and a fitness function are defined. Grammatical evolution which was proposed by Ryan et al. [186] is considered to be an extension of GP [175]. Unlike GP which

uses syntax trees to represent individuals, GE uses chromosomes of variable length binary strings. Each gene of the chromosome is an 8-bit binary string and is referred to as a codon. Codons contain information on how to select production rules from a BNF grammar. Figure **2.9** is an illustration of two examples of GE individuals where a) is a genome consisting of 6 codons and b) consists of 8 codons.

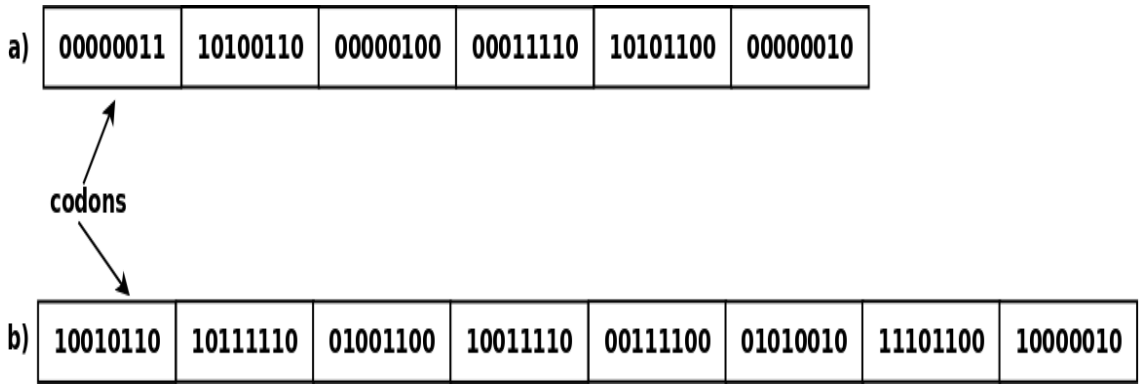


Fig. 2.9 Examples of GE individuals

Grammatical evolution uses a user-defined Backus Naur Form grammar to map variable length linear genomes to executable programs. Domain knowledge of the problem being solved is incorporated into the grammar. The grammar is used in conjunction with evolution to map a genotype to a phenotype. Harper and Blair [91] assert that the simplicity of the representation used by GE, i.e. bit strings, gives GE the flexibility of allowing a wider range of possible operators to be used during the search.

Grammatical evolution draws inspiration from molecular biology where a sequence of genetic material, deoxyribonucleic acid (DNA)(genotype) is translated into a protein which defines the characteristics of a phenotype. A grammar G can be represented by the four-tuple $<N, T, P, S>$, where N represents a set of non-terminals, T a set of terminals, P a set of production rules that map the elements of N to T and S (a member of N) the start symbol. An individual (genotype) is used to map the start symbol S to terminals by reading and converting each *codon* to its decimal value from which an appropriate production rule is selected by using the following mapping function:

$$Rule = (codon \quad decimal \quad value)\%(N^o \quad of \quad production \quad rules) \tag{2.12}$$

A derivation tree (phenotype) is evolved by iterating and mapping through the sequence of codons. The derivation process is performed from left to right starting with the left-most non-terminal. If the iteration process reaches the end of the sequence of codons before the derivation tree is evolved the procedure continues by looping to the start of the codon

sequence, a process called *wrapping*. The fitness of the phenotype is evaluated by applying it to a problem at hand.

Algorithm 4 Grammatical Evolution

1: Create an initial population of variable length binary strings
2: Map via a BNF grammar
   a) binary strings to expression using production rules
3: Evaluate fitness
4: **do while** {termination condition not met}
5: Select fitter individuals for reproduction
6: Recombine selected individuals
7: Mutate offspring
8: Evaluate fitness of offspring
9: Replace all individuals in the population with offspring
10: **end while**
11: **return** best individual

Algorithm **4** is an outline of the step by step GE algorithm.

2.3.3.1 Initial Population Generation

The first step of the GE algorithm is to randomly generate a population of variable length individuals. The population size and the variable length limits are user specified. Ryan and Azad [185] argue that random initialisation may lead the search to start in a poor area thus leading to either premature convergence or generation of invalid individuals. They proposed an approach known as *sensible initialisation*. This method generates derivation trees using the ramped half-and-half method. Random initialisation remains the most commonly used initial population generation method, however, *sensible initialisation* is becoming widely accepted and an in-depth analysis of this method is provided in [185].

2.3.3.2 Mapping

After population initialisation the randomly generated genotypes are mapped onto phenotypes. The mapping process involves the BNF grammar and the rule specified by equation **2.12** to produce valid phenotypes. The mapping is deterministic meaning if the grammar remains the same, a particular genotype will always be mapped to the same phenotype. An example of the mapping process is demonstrated using a simple grammar for generating pin numbers depicted in Figure **2.10**. The symbols enclosed within the angular brackets <> are non-terminals and those without are terminals. The production rules are indexed from 0 and are

<S> ::= <X><W><W> | <W><X><Y> | <X><W><Y>
<W> ::= <Y><X> | <Y><Z>
<X> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
<Y> ::= a | b | c | d | e | f | g | h | i | j
<Z> ::= @ | ~ | # | & | % | * |) | - | (| +

Fig. 2.10 Grammar for pin generation

separated by the | sign. Therefore, if a non-terminal for example <X> from the grammar has ten production rules the first one indexed by 0 the next one 1 until the last one 9. The indexes are used in conjunction with equation **2.12** to select the production rules. Using individual **a)** from Figure **2.9** as an example, the first step is to convert the binary codons to their decimal values which is 5,166,4,30,172,17. The next step is to map the start symbol <S> to terminal using rule **2.12**

1. 5%3 = 2 since there are 3 production rules rule 2 is selected i.e. <X><W><Y> . Next we process the left-most non-terminal.
2. 166%10 = 6 production rule 6 is selected mapping to 7<W><Y>
3. 4%2 = 0 production rule 0 $\mapsto$ 7<Y><X><Y>
4. 30%10 = 0 production rule 0 $\mapsto$ 7a<X><Y>
5. 172%10 = 2 production rule 2 $\mapsto$ 7a3<Y>
6. 17%10 = 7 production rule 7 $\mapsto$ 7a3-

The evolved phenotype (pin) is 7a3-. The fitness of the phenotype is measured using a problem dependent fitness function. For pin numbers, this can be standard password metrics such as complexity. The derivation tree for this example is shown in Figure **2.11**

If the termination criteria is not met a selection method is used to select genotypes to act as parents to generate offspring.

2.3.3.3 Selection

The commonly used selection methods in GE are tournament selection and fitness proportionate selection. These are implemented in the same manner as explained in section **2.3.1.3**.

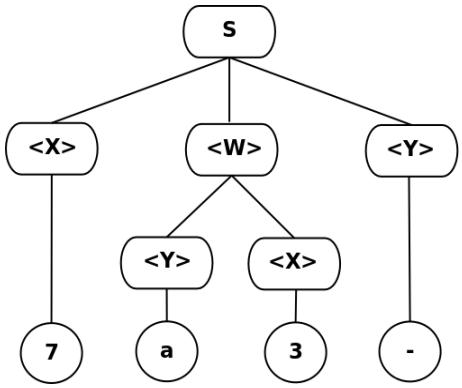


Fig. 2.11 Derivation tree pin generation

2.3.3.4 Genetic Operators

Crossover and mutation are the commonly applied genetic operators for grammatical evolution.

Crossover according to O'Neill et al. [167] one-point crossover is the most widely used crossover operator in GE. This is applied in a similar manner as in genetic algorithms except that in GE it is used on variable length genomes resulting in variable length offspring. Two parents are selected using a chosen selection method and a crossover probability rate is used to determine if the crossover operator should be applied. If crossover is to be applied a random crossover point r in the range [1, l -1] where l is the maximum length of the shortest parent is generated and used as the crossover point where the tails of each parent are exchanged. The application of one-point crossover in GE is illustrated in Figure **2.12**. Other variations of this crossover operator have been used in GE, where a crossover point is

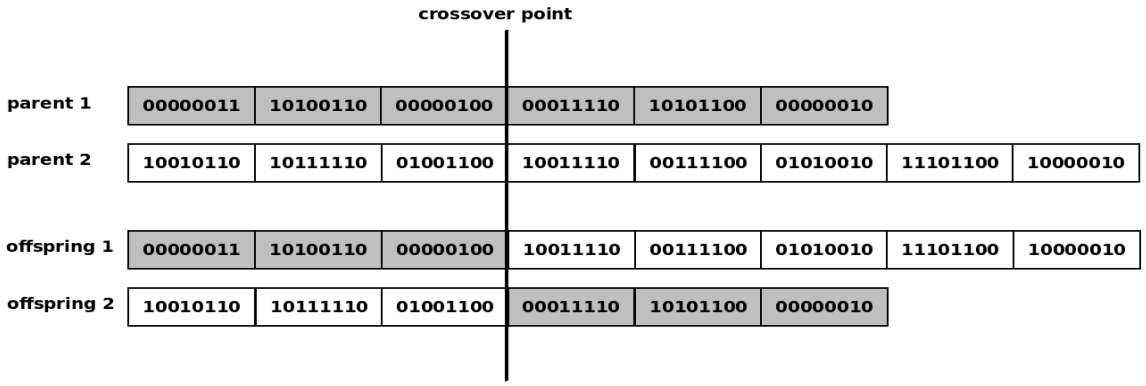


Fig. 2.12 Examples of GE individuals

randomly selected on each parent and the tails are exchanged [38].

Mutation bit flip mutation described in section **2.3.1.5** is the commonly used mutation operator in GE. This is enhanced by the fact that the representation in GE uses bit strings.

2.3.3.5 Population Replacement

Grammatical evolution follows a similar approach as the other EAs when it comes to population replacement. Population replacement can either be generational or steady-state replacement. Elitism may also be applied in GE.

2.3.3.6 Termination

Like the other EAs termination can occur either after a certain number of generations or if a problem specific solution is obtained.

2.3.3.7 Applications of Grammatical Evolution

Grammatical evolution has been applied in numerous domains particular to solve real-world problems. Sen and Clark [192] use GE to evolve programs that detect intrusions in a mobile ad hoc network (MANETS) In [34] Bryne et al. use GE for architectural design while Tanev et al. [209] apply GE in the automated design of a robot. The next section reviews the application of GP in classification.

2.4 GP and Classification

Generally, when used as a classification algorithm GP uses supervised learning. GP randomly initialises a population of classifiers and evaluates their fitness by applying each classifier to a training set. The classifiers are then improved gradually as they are evolved from generation to generation (learning) until a stopping criterion is met. The best returned classifier at the end of the evolution process is then applied on a test set. Predictive accuracy is the most commonly used fitness function to evaluate classifiers. As previously stated the most commonly used approach to represent GP individuals is using syntax trees. This is also the case for GP classification algorithms. Although there are studies in the literature that use other representations to evolve classifiers using GP. For example, Green et al. [81] describe a GP classification algorithm to classify human genomes. The system encodes GP individuals using a linear genome structure. Eiben et al. [58] argue that the ease with which a representation matches a problem makes it more suitable than other representations.

2.4.1 GP Classifier Models

The flexibility of the tree representation gives GP a distinct advantage of being able to model classification rules and decision trees [64]. The most commonly modelled classification rules using GP are *discriminant functions*. In this form, the GP classifier is represented as a mathematical expression constituting of operators and/or functions from the function set and features from the terminal set. The expression is resolved into a value (usually floating point) which is the output of the GP tree. The value is then translated to a class. Greene et al. [81] use the term *symbolic discriminant function* to define these GP classification rules. The operators and functions that constitute a function set for a GP classification algorithm determine the type of classifiers evolved. In this thesis, we restrict our focus to *arithmetic* and *logical* classifiers for classification rules and decision trees. If the function set is populated by arithmetic operators GP will evolve arithmetic classifiers and if the function set constitutes of logical operators, logical trees classifiers are evolved. If the function set is populated with features from the dataset and the terminal set with classes then decision trees are evolved.

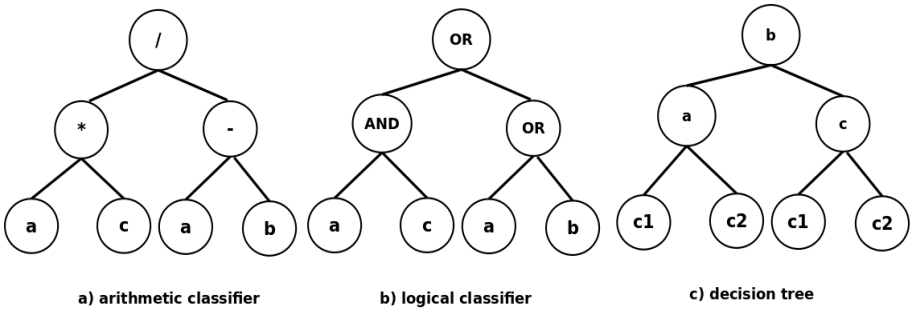


Fig. 2.13 Examples of GP classifiers

The graph-like structure of a decision trees makes it convenient to use syntax trees to model decision trees in GP. The classifier is evaluated from the root downwards in a top-down manner. Koza [117] was the first to propose the induction of decision trees using GP. Figure **2.13** is an illustration of examples of an arithmetic classifier, logical classifier and decision tree evolved by GP.

Another advantage of GP is that it is capable of performing automatic feature selection. Since features are randomly selected for each tree this reduces the dimensionality of the input data for classification as not all features are selected to be part of the tree.

GP classification algorithms that evolve arithmetic tree, logical tree and decision trees classifiers are widely used in numerous problem domains. Zhang and Nandi [232] propose a GP classification algorithm to classify faults in the manufacture of roller bearings. Arithmetic tree type classifiers are used and their effectiveness is compared to classifiers evolved by two-hybrid classification algorithms namely, a GA-ANN and a GA-SVM. In both of the

hybrid algorithms, the GA is used for feature selection, which GP performs automatically. GP is reported to outperform the hybrid classification algorithms. Griffin and Chen [84] use arithmetic classifiers evolved by GP to detect the occurrence of faults during grinding. Grinding is a process performed during the manufacture of a wide range of materials such as vehicle parts. During grinding acoustic signals are emitted and faulty grinds emit a different sound. Some faults are visually undetectable. In this study, GP is successfully used to detect faults based on the acoustic signals emitted. Raymer et al. [181] evaluate the effectiveness of GP arithmetic tree classifiers to detect the presence of water in protein molecules. The researchers compare GP classifiers to a GA on the same dataset. GP is reported to not only detect water molecules at a higher rate than the GA but it also uses fewer features in the detection thus reducing the computational cost of classification. Hong and Cho [94] show the effectiveness of arithmetic tree classifiers in the detection of lymphoma cancer. Bojarczuk et al.[26] describe a GP classification algorithm that evolves logical trees classifiers. The effectiveness of the classifiers is evaluated on five datasets from the University of California Irvine (UCI) repository [15] containing medical data. The performance of GP is found to be better than the C4.5 algorithm. Similarly, in [45] three datasets containing medical data obtained from the UCI repository are used to evaluate GP evolved logical classifiers. Hirsch [92] use GP to evolve logical classifiers for text classification. The performance of GP is reported to be comparable to other classification algorithms from the literature. Mousavi et al. [152] use GP to evolve arithmetic and logical classifiers for portfolio trading. An individual is represented by multiple trees depending on the number of stocks in the portfolio. If for example, a portfolio has 5 stocks then 1 individual will have 5 trees each representing a specific stock. Each of the trees may be arithmetic or logical and each represents a rule for each stock. A total of 30 stocks obtained from the Canada (15) and Iran (15) stock exchanges are used to evaluate the proposed classifiers. The proposed classifiers are reported to perform better than a GA and the Buy and Hold (BH) strategy. A BH strategy is where an investor buys stock and holds it for a long time ignoring the fluctuations [69]. Khoshgoftaar et al. [113] propose a GP system that evolves decision trees to classify the quality of a software program. Wang et al. [219] propose a GP system that evolves decision trees for financial forecasting. The study shows that GP is able to evolve classifiers able to extract financial forecasting rules. Fallah et al. [66] use decision trees evolved by GP to predict and control the water levels of a reservoir. Other examples of GP classification algorithms that evolve decision tree classifiers are presented in [65, 140, 165].

2.4.2 Population Initialisation

An initial population provides a starting point for a search. Since there is relatively no task-dependent knowledge used during the creation of the initial population, diversity is important. From the surveyed studies the ramped half-and-half method is the most widely used population creation method for GP classification algorithms [26, 37, 92, 152, 181]. Although the full and grow methods are also used no justification is provided in the literature for the choice of the ramped half-and-half method hence, it is assumed that the popularity of this method is due to the fact that the population consists of a mixture of individuals generated by the full and grow methods. Therefore, it is likely to provide a diverse initial population. However, Silva et al. [197] argue that diversity is not that important for the initial population for GP as it is likely to be introduced during evolution by mutation.

2.4.3 Fitness Function

The evaluation of classifiers evolved by GP can be a single measure i.e. single objective optimisation or multiple measures i.e. multi-objective optimisation. Depending on the problem the measured fitness function may be maximisation or minimisation. Predictive accuracy (equation **2.1)** is the most widely used fitness function for GP classification algorithms. This is a single objective measure which seeks to maximize the accuracy of classifiers. The following studies use predictive accuracy as the fitness function. [71, 73, 153, 164, 234]. Another commonly used single objective fitness function measure is the $f_{measure}$ (equation **2.5)** also known as the f_{score} . This fitness function is used in the following studies [21, 35, 46, 103, 105]. Parameters of the confusion matrix have been used as fitness functions such as the *true positive rate* [123], *sensitivity* [178] and *specificity* [195]. Other metrics which are used less often as fitness functions found in the literature are the *J-Measure* and *Matthews correlation coefficient (MCC)*. The *J-Measure* is a measure of the quality of disjunction rules that describe a class. This metric is used as a fitness function in [72, 104], while in [188, 189] the *Mathews correlation coefficient* (equation **2.13)** is used as the fitness function.

$$MCC = \frac{(t_p * t_n) - (f_p * f_n)}{\sqrt{(t_n + f_n) * (t_n + f_p) * (t_p + f_n) * (t_p + f_p)}} \tag{2.13}$$

The root mean square error which is commonly used for regression problems is also extensively used for classification problems in GP [109].

Barros et al. [20] identify three approaches to multi-objective optimisation i) weighted formula ii) pareto dominance and iii) lexicographic analysis. In this thesis, we focus on the weighted formula approach which is the commonly used strategy in GP classification

algorithms. In this approach, the fitness function consists of multiple objectives which contribute towards the final value of the fitness. The amount contributed by each objective is determined by weights i.e. the objectives are weighted depending on their importance in the application. A formula is normally then applied to manipulate the weighted amounts to a single value which represents the fitness.

The most widely used weighted approach in classification involves weighting accuracy and tree complexity. Tree complexity is usually taken as the size of the tree i.e. the number of nodes [194]. Although other researchers have used other metrics to define tree size such as the depth of the tree or the number of layers [217], in [18] 6 different fitness functions are used. Each fitness function is a summation of accuracy and another metrics. Bhowan et al. [22] describe a fitness function which is the square root of the product of the accuracy of each class, i.e. class 1 and class 2 for a binary problem. A weighted fitness function is used in [224] where 3 objectives are used to formulate a fitness function. Li [128] in applying GP to a financial forecasting problem proposes a similar approach which uses a fitness function that sums 3 objectives.

2.4.4 Selection

The two widely used selection methods are tournament and fitness proportionate selection. Tournament selection is the most commonly used and has a tournament size parameter associated with it. Typical values of the tournament size are discussed in section **2.4.8**. Tournament selection is used in the following studies [92, 196, 224, 232]. Fitness proportionate which selects classifiers based on their fitness is used in [26, 37, 67, 129, 152, 181].

2.4.5 Genetic Operators

The genetic operators presented in section **2.3.2.4** are the most widely used operators for GP in classification. Application rates are used with each operator to evolve new classifiers, these are discussed in section **2.4.8**.

2.4.6 Population Replacement

A generational model is the most popular population replacement strategy for GP classification algorithms. A new generation of classifiers is used to replace a current population for the next generation. The generational approach is exemplified in the following studies [196, 232].

2.4.7 GP and Multiclass Classification

In this section GP, strategies for multiclass classification are reviewed. The review is restricted to strategies that use tree representations to evolve classification rules. Hybrid strategies such as those that combine GP and other methods are not considered.

La Cava et al. [122] and Munoz et al. [154] argue that traditionally GP does not do well for multiclass classification when compared to other state-of-the-art classification methods. At a high level, two strategies are commonly followed to perform multiclass classification with GP i.e. those that extend binary classification and those that do not extend binary classification. Binary classification is extended by simply enumerating the number of classes considered. This approach is easily implemented when GP is used to model decision trees as the extra class(es) is(are) added to the terminal set. However, when GP models classification rules there is increased complexity. As mentioned earlier, in binary classification a tree (classifier) is interpreted as a classification rule which is a discriminant mathematical expression. A mathematical operation is carried out using an input fitness case from the dataset resulting in a floating point value. The floating value is used to associate the input vector to one of the two classes under consideration. An approach referred to as *static class boundary determination* (SBCD) which extends binary classification is presented by Zhang and Ciesielski [233]. In this approach, a numerical range is predefined for each class. So for an n-class problem, there would be n ranges. Consider for example a 3-class problem boundaries may be defined as follows: class 0 = [-inf to -5.0], class 1 =[-5.1 to 5.0] and class 2 = [5.1 to inf]. If for example, a numerical output value from a classifier is -3.8, then the object is in class 1. This approach to classification is considered to have an advantage of being simple and easy to implement. However, finding the boundaries that provide the best classification is no easy task as boundaries are selected arbitrarily. In [140] an improved version of SBCD is presented where boundaries are selected dynamically. A subset of the dataset is used to find the best boundaries and these are then used for the rest of the run. This approach is known as *dynamic boundary class determination* (DBCD). This method is shown to perform better than SBCD for complex problems.

Of the strategies that do not extend binary classification, binary decomposition is the most commonly implemented. This approach reduces an n-class problem to n binary problems. For each class, a run is performed with the other classes combined. This approach is followed in the following studies [114, 132, 133, 140, 198, 230]. In [153] Muni et al. present a multi-tree approach to solving the multiclass classification problem. In this approach a classifier (tree) is evolved for each class in a single GP run. For example, for a 4-class problem, 4 classifiers (trees) will be evolved. Other lesser used methods for multiclass classification are presented in [64].

2.4.8 GP Parameters for Classification Problems

This section presents a brief review of the numeric parameter values found in the literature for configuring GP for classification problems.

It is widely accepted that the success of an evolutionary algorithm is largely influenced by its configuration. Part of algorithm configuration is assigning values to numerical and categorical parameters. Genetic programming is amongst one of the most parameterised evolutionary algorithms. Espejo et al. [64] describe this to be one of the disadvantages of GP. In this thesis we consider the following GP numerical parameters: *population size, initial tree depth, maximum offspring depth, tournament selection size, crossover rate, mutation rate, mutation offspring depth* and *number of generations*.

- *population size*
  Poli et al. [175] describe the population size parameter as a very important parameter for GP. The value assigned to this parameter represents the number of classifiers considered for solving the problem. The following values are found in the literature: 100 classifiers are used in [6, 14, 133], 200 classifiers are used in [230], 500 classifiers are used in [140, 198], while in [132, 202] 2000 classifiers are used.

- *initial tree depth*
  According to [18] initial tree depth is usually set at low values at initialisation to speed up algorithm runtimes. In [175] Pol et al. emphasise the importance of the initial tree depth parameter for GP. Oltean and Dioşan [164] describe a GP classification algorithm that starts with a population size of 1 and an initial tree depth 1. The proposed approach doubles the population size if there is no improvement in the fitness of the best individual after a certain number of generations. New randomly created individuals are added to the population. The initial tree depth of the new individuals is also increased by one at each addition point. Aitkenhead [5] recommends restraining the initial tree depth of decision trees to a value of 2. This recommendation is supported by Papagelis and Kalles [168]. The premise is that during algorithm execution evolution will deepen the offspring. The majority of studies set the value of this parameter in the following range (2 - 7) [6, 140, 198, 202, 230].

- *maximum offspring depth*
  Maximum offspring tree depth is used to restrict the size of growth of classifiers as the algorithm executes. This parameter helps in controlling bloat. GP programs contain redundant code known as *introns* and the number of introns increase during evolution and this is referred to as bloat [17]. Controlling the size of trees using parameters such

as *offspring depth* are efforts of controlling bloat. A detailed analysis of the effects and control of bloat in GP is provided in [174]. Offspring depths is set to the following values are used in the following studies: 5 in [14], 6 in [230], 7 in [129], 8 in [133], 10 in[6], 15 in [198] and 17 in [140, 202, 214].

- *tournament selection size*
  Silva and Tseng [198] used a selection size of 50 with a population size of 500 in their study which is high when compared to values used in other studies. A high selection size also introduces an element of elitism as individuals with a higher fitness will stand a better chance of being selected frequently. Generally tournament selection size is usually set at low values with typical values distributed as follows: 3 in [14], 4 in [214], 5 in [133], 6 in [132], 7 in [6].

- *crossover rate*
  A wide range of values are used for the crossover rate which is considered to be the main genetic operator. In most studies a higher crossover rate than the mutation rate is used. The following values are commonly used and found in the literature: 50% [198], 80% [132], 85% [14, 133] and 90% [140, 214, 230]. Crossover is sometimes applied as the only genetic operator such as in [65] or it is used with reproduction such as in [202]. Loveard and Ciesielski [140] also used crossover and reproduction to evolve classifiers.

- *mutation rate*
  Generally, mutation is set at low rates as it introduces diversity it does not encourage convergence. According to Poli et al.[175], if the population size is low a higher mutation rate is required to maintain diversity during evolution. In [198, 202] mutation is used to evolve half of the offspring at each generation with the mutation rate set at 50%. The following are typical values used: 5% [14, 133], 10% in [214, 230] and 20% in [132].

- *mutation offspring depth*
  Mutation offspring depth is used to restrict the size of the subtree used for mutation. Typical values are found in the following of range values (2-9) [92, 97, 221].

- *number of generations*
  A fixed number of generation is used to terminate the algorithm. The following values are typically used for this parameter: 30 [6], 50 [14, 140], 100 [132], 200 [230], 250 [133], 500 [181, 232] and 1000 [84].

2.4.8.1 Parameter Tuning

There is no set of parameter values that will lead to an optimal solution for all problems [58, 175]. The determination of parameter values for configuration of GP classification algorithms is predominantly performed manually using parameter tuning. Poli et al. [175] argue that GP is a robust algorithm capable of finding acceptable solutions using many different parameter settings. The authors further argue against carrying out intensive parameter tuning for GP parameters. Using this recommendation some researchers, therefore, carry out parameter tuning using a subset of the problem instances and then apply the same parameters on the rest of the problem instances. However, other researchers prefer to carry out parameter tuning for each problem instance [142, 184].

Parameter tuning is an iterative trial and error approach. For each parameter, a finite set of values are considered usually based on the values obtained from the literature. Each parameter is considered one at a time and trial runs are performed while the values of the other parameters remain the same. A number of researchers are critical of this approach to setting the parameter values. Montero et al. [151] describe this as a time-consuming *brute-force* approach. This view is supported by Veček et al. where in [215] it is pointed out that setting a boundary of values to be considered may leave out the value which may lead to an optimal solution. Furthermore, the authors argue that the appropriate number of trial runs to be performed is set arbitrarily without justification. Hutter[98] argues that the correlation between the parameters is not considered during parameter tuning and additionally the search space is too wide for a human designer and this leads to human bias where values are eventually assigned based on preference. In [151] Montero and Riff argue that to effectively tune parameters one should have expert knowledge of the domain being considered, however, this is not always possible. Several parameter tuning methods have been proposed in the literature [24, 101, 155] but none have been universally adopted.

2.5 Automated Design

This section defines automated design in the context of this thesis followed by a review of the research carried within the context of the definition.

2.5.1 Definition of Automated Design

Generally, given an algorithm $f(x)$ with parameters p the process of finding a combination of parameters and their values such that $f(x) = \{p_1, p_2, p_3 .... p_n\}$ finds an optimal or near optimal solution for a given problem is an algorithm configuration problem [59]. In this thesis, we

define an automated approach to solving this problem as automated design. The focus is on the automated design of genetic programming classification algorithms using a GA and GE. During manual design of a parameterised algorithm, design decisions are made that lead to a configuration of the algorithm. Automated design therefore can also be described as the automation of these design decisions. The next section outlines the design decisions made to solve the algorithm configuration problem.

2.5.1.1 Design Decisions

Design decisions may include the following:

1. *Determining the parameters for an approach.*
   This design decision involves determining the parameters and their values, e.g for an EA it could be assigning a value to the population size parameter or assigning a selection method to use such as tournament selection over fitness proportionate selection.

2. *Determining the operators to use.*
   This design decision determines which operators are to be used. For example, determining which genetic operators to use for an EA, such as which crossover operator to use uniform crossover or single-point crossover.

3. *Determining the control flow of an approach.*
   In this design decision the control flow of the algorithm is determined. Sevaux et al. [193] describe algorithm control flow as the order in which components in an algorithm combine to achieve the final output. A component is a process of an algorithm such as a mutation operator in an evolutionary algorithm. Therefore, in this design decision the order in which the components are linked for algorithm execution is determined. For example crossover followed by mutation [54].

4. *Creating new construction heuristics*.
   Construction heuristics are often used with metaheuristics to solve a problem. These are rules of thumb that are usually manually derived. This involves automating the creation of these heuristics [33].

5. *Creating new operators.*
   This design decision involves the creation of new operators. For example Hong et al. in [95] create new mutation operators for evolutionary programming.

An algorithm designer needs to make these design decisions in order to arrive at a combination of components that result in *f(x)* yielding an optimal solution for a considered problem. While a number of definitions of the term *configuration* have been suggested, this thesis follows the definition suggested by Sevaux et al. [193] where a *configuration* is defined as a specific set of parameter values and control flow to solve a problem at hand. Algorithm configuration usually requires expert knowledge. In [193] manual configuration is described as an art which relies on intuitive decisions that are based on experience. Hence, the authors argue that novice designers cannot effectively configure algorithms that yield optimal solutions.

A large and growing body of literature has investigated the application of evolutionary algorithms to design and configure other algorithms. Research can be found under parameter tuning [156], parameter control [100, 137] and hyper-heuristics [33]. However, the review presented in the next section is restricted to those studies that are closely related to the research carried out in this thesis i.e. those that use a GA and GE to automate other metaheuristics. The review focuses on studies where a GA and GE take the place of an algorithm designer and determine the design decisions to configure the best algorithms for particular problems.

2.5.2 Automated Design using Genetic Algorithms

Genetic algorithms have been widely used for automating the design of other metaheuristics.

Preliminary work on using a GA to automate the design of an EA was undertaken by Grefenstette [82] where a GA was used to tune parameter values i.e. design decision 1 (section 2.5.1.1) for GAs. Each element of the population of the designing GA represented parameters for a GA. The evolved parameters were found to be effective at configuring GAs for solving numerical function optimisation problems. The designing GA was configured with the following parameters: population size 50, crossover rate 60%, mutation rate 0.1% and elitism. Twenty generations were used as the stopping criteria.

Similarly, Brain and Addicoat [28] used a generational GA to determine design decision 1 for GAs that were used to solve a computational chemistry problem. The designing GA was configured as follows: population size 80, tournament selection (size 2), single point crossover rate at a probability rate of 50%, random mutation at a probability rate of 2%. The termination criteria was set to 20 generations.

Souffriau et al. [204] used a steady-state genetic algorithm to determine parameter values for an Ant Colony Optimisation (ACO) metaheuristic. Each individual of the GA represents ACO parameter values and real number encoding was used to encode each chromosome. Each gene of the chromosome represents a specific parameter of an ACO algorithm. The GA was used to search for parameter values that yield the best ACO algorithm for the orienteering

problem, a form of the travelling salesman problem. The ACO algorithms configured with the GA evolved configurations were reported to perform better than those that were tuned manually. The GA was configured as follows: population size 100, stochastic universal selection, uniform crossover rate at a probability rate of 90%, integer flip mutation at a probability rate of 0.1% and elitism at 2%.

In [53] Diosan and Oltean described an approach that used a generational GA to evolve evolutionary algorithms. Each element of the GA was used to represent a complete EA, determining design decisions 1, 2 and 3. The best evolved EA was tested on numeric function optimisation problems and the results were reported to be competitive when compared to other approaches for solving function optimisation methods. The GA was configured as follows: population size 100, tournament selection with tournament size 2, single point crossover rate at a probability rate of 80%, gaussian mutation [80] at a probability rate of 10%. The termination criteria was set as 100 generations.

2.5.3 Automated Design using Grammatical Evolution

Grammatical evolution has also been extensively used for automated design.

Tavares and Pereira [211] used GE to configure ACO algorithms. The GE algorithm was used to determine design decision 1, 2 and 3 for the ACO algorithms. The performance of the GE evolved ACO algorithms was compared to the performance of manually designed ACO algorithms in solving the travelling salesman problem. The GE evolved ACO algorithms were reported to outperform the manually designed ACOs. The GE algorithm was configured as follows: population size 64, individual size 128, tournament selection with tournament size of 3, single-point crossover at a probability rate of 70%, standard integer-flip mutation at a probability rate of 5% and elitism. Twenty-five generations were set as the termination criteria of the algorithm.

Lourenço et al. [138] presented an approach that used GE to configure EAs. In this study GE was used to determine design decision 1, 2 and 3. The evolved EAs were reported to be effective at solving Royal Roads Functions. The GE algorithm was configured as follows: population size 100, individual size 10-16, tournament selection with tournament size of 5, single-point crossover at a probability rate of 90%, bit flip mutation at a probability rate of 1% and 50 generations as the termination criteria.

In [139] Lourenço et al. used GE in a hyper-heuristic to evolve EAs to solve the knapsack problem. Design decision 1, 2 and 3 was automated for each EA. The evolved EAs were shown to perform well on unseen problem instances when compared to other methods. The GE algorithm was configured as follows: population size 100, individual size 10-16, tournament selection with tournament size of 3, single-point crossover at a probability rate

of 90%, bit flip mutation at a probability rate of 1% and 50 generations as the termination criteria.

Drake et al.[57] also presented a hyper-heuristic which used GE to design a local search method, namely the variable neighbourhood search (VNS). The GE grammar automated design decision 1, 2, 3, 4 and 5 for the VNS. The GE evolved VNS was tested on instances of the vehicle routing problem. The GE algorithm was configured as follows: population size 1024, tournament selection with tournament size of 7, crossover at a probability rate of 90%, mutation at a probability rate of 5%, reproduction at a probability rate of 5% and 50 generations as the termination criteria.

Miranda and Prudêncios [149] used a steady-state GE to design and configure particle swarm optimisation (PSO) algorithms. GE is used to determine design decision 1, 2 and 3. The GE designed PSO algorithms were found to perform competitively when compared to other state-of-the-art methods for solving continuous optimisation problems. The GE algorithm was configured as follows: population size 50, individual size 30, fitness proportionate selection, single point crossover at a probability rate of 80%, bit mutation at a probability rate of 1% and 20 generations as the termination criteria.

2.5.4 Analysis of Automated Design

It is clear from the reviewed studies that there is no one set of standard configuration settings for configuring a GA or GE for automated design. Generally, all methods were reported to be effective in the problems that they were applied to. The parameter settings for each study are problem dependent. Some of the studies used an automated design approach to tune parameters (design decision 1) while others automated the determination of design decision 1, 2 and 3. and one study used a hyper-heuristic approach to determine design decision 1, 2, 3, 4 and 5.

From the studies that used a GA for automating the design, the majority used a generational control model for population control. In the majority of the studies, the representation used is problem dependent for example in [53] an individual represented an EA while in [82] an individual represented parameter values. Real number encoding was found to be the most commonly used form of encoding. Tournament selection is the most widely used selection method. Single point crossover is the commonly used crossover method although uniform crossover is also reported to be used as in [204]. Random mutation where values for each gene are randomly assigned from a range of allowed values is the most common form of mutation used. Elitism which involves preserving individuals with a high fitness is also commonly applied. Populations size values range from a low of 50 to a maximum of 100 individuals. In all the studies the crossover probability rates were set at much higher values

than mutation probability rates. The termination criteria value i.e. the number of generations ranged from a low of 20 to a maximum of 100 generations.

In all the studies that used GE for automated design, binary encoding was used for encoding variable length GE individuals. The majority of the studies determined design decision 1, 2, 3, however, this is problem dependent. Tournament selection was the most commonly used selection method although fitness proportionate selection was also used in [149]. Single-point crossover is the crossover operator used in all the studies with bit flip mutation. As expected higher crossover probability values and lower mutation probability values were used. Elitism was also commonly applied. Population size values ranged from 50 to 1064 and termination criteria values ranged from 20 to 50 generations.

2.6 Fitness Landscape and Fitness Landscape Analysis

A number of metrics have been proposed for the evaluation of evolutionary algorithms. Furthermore, it has been of interest to the evolutionary algorithm research community to try and gain an understanding of why and how evolutionary algorithms work. Fitness landscape analysis is one such approach that has been widely applied in trying to answer that question. This section introduces the concept of fitness landscape and then describes the techniques used to perform fitness landscape analysis.

2.6.1 Fitness Landscape

The concept of fitness landscape is adopted from theoretical biology where it was initially conceptualised by Wright [228] in the early 1930s. According to Pitzer and Affenzeller [172] viewing the search space as a fitness landscape is beneficial to gain an intuitive understanding of how and where heuristic algorithms such as EAs operate. An analysis of the fitness landscape provides a measure of the difficulty of solving a problem at hand or finding the optimal value for an EA [173]. In [146] it is argued that the knowledge of the fitness landscape may lead to the design of more efficient algorithms as the algorithm designer will be having domain knowledge of the search space. According to Jones [110] a fitness landscape is neither a search space nor a search algorithm, therefore, it is not possible to provide a precise strict formal definition of a fitness landscape. However, the author further argues that an informal definition may be provided through relating specific elements of both the search space and search algorithm. The fitness value of each individual in a population can be viewed as a point in the search space. In [172] it is pointed out that as much as the fitness values of individuals in a search space may be used to provide some form of

visualisation of the search space they still do not provide a complete picture as no information is provided on how the points are related or connected. The relationship between individuals may be through direct connectivity, through neighbours or operators such as mutation or crossover to distances measures or hypergraphs [173]. Merz and Freisleben [146] described a fitness landscape as the set of all candidate solutions in the search space with the fittest candidate being the peak of the landscape. The authors formally define the fitness landscape (***S, f, d***) of a problem instance of a given combinatorial optimisation problem as being made up of a set of candidate solutions ***S***, a fitness function $\boldsymbol{f}:\boldsymbol{S}\longmapsto \boldsymbol{R}$ which assigns a real-valued fitness to each solution in ***S*** and a distance metric ***d*** that defines the spatial structure of the landscape.

2.6.2 Fitness Landscape Analysis

Fitness landscape analysis has been widely carried out in the area of evolutionary algorithms. A number of fitness landscape analysis methods have been used to try and understand the behaviour of evolutionary algorithms such as neutrality and evolvability [83, 200] and epistasis variance [157] among others. A comprehensive survey of fitness landscape analysis is presented in [144]. There are also a number of application studies found in the literature that apply fitness landscape analysis to well known problems. In [207], Stadler and Schnabl performed a fitness landscape analysis on the travelling salesman problem, while in [206] a fitness landscape analysis of a graph-bipartitioning-problem [130] is presented. In [212] Tavares et al. carried out a fitness landscape analysis on a multidimensional knapsack problem. In the following studies [143, 162, 163] fitness landscape analysis is carried out on hyper-heuristic problems. The automated design approach proposed in this thesis bear similarities to hyper-heuristics, therefore, the metrics used in the fitness landscape analysis applied to hyper-heuristic studies are reviewed. These are presented in the next section.

2.6.2.1 Fitness Landscape Analysis Metrics

In this section two of the most commonly used statistical fitness landscape analysis methods, namely fitness distance correlation and auto-correlation analysis, are reviewed.

i) **Fitness distance correlation**
This measure is considered to be the most commonly used metric for evaluating problem difficulty for genetic algorithms [162]. Given a set of points p_1, p_2, p_3...p_n and their fitness values $f_1, f_2, f_3...f_m$, the fitness distance correlation coefficient ρ is defined as follows:

$$\rho(f,d_{opt}) = \frac{Cov(f,d_{opt})}{\sigma(f).\sigma(d_{opt})} \tag{2.14}$$

In the given equation *Cov(.,.)* is the covariance of two random variables and $\sigma(.)$ is the standard deviation. The fitness distance correlation is used to establish how closely related the set of fitness points and their distances are to the nearest optimum in the search space given as d_{opt}. A fitness distance correlation value that is less or equal to -0.5 (i.e. ρ <= *-0.5*) for maximisation problems is indicative of an easy problem. While a value of ρ = 1.0 indicates a difficult search as an increase fitness values also means an increase in distance to the optimal value.

ii) **Autocorrelation analysis**
This measure is commonly used to evaluate the ruggedness of a search space. According to Merz and Freisleben [146] a fitness landscape can be defined as rugged if it contains many peaks and if the neighbouring points have a low correlation. The evaluation of ruggedness using auto-correlation analysis was proposed by Weinberger [220]. Weinberger investigated the correlation structure of a fitness landscape based on the autocorrelation function[96]. Weinberger suggested that by performing a random walk across a landscape the ruggedness of the landscape may be described [143]. The approach is to perform a random walk of size T, on the landscape via neighbouring points. A time series of fitness values is then generated as at each step the fitness is recorded. The autocorrelation function of the time series is then calculated using equation 2.15

$$acf_s = \frac{\sum_{t=1}^{T-s}(f_t - \bar{f})(f_{t+s} - \bar{f})}{\sum_{t=1}^{T}(f_t - \bar{f})^2} \tag{2.15}$$

Where f_t is the fitness and $\bar{f}$ is the mean fitness of the T points. The auto-correlation function establishes the correlation of the fitness of two points in the search space separated by *s* steps. The ruggedness of landscapes can be compared using the correlation lengths. This length can be obtained from the *acf* using the following equation

$$cl = -\frac{1}{\ln|acf_1|} \tag{2.16}$$

A high correlation length is indicative of a smoother landscape while a low value indicates the landscape is more rugged.

In [162] Ochoa et al. performed both fitness distance correlation and autocorrelation analysis in analysing the landscape of a graph based constructive hyper-heuristic for timetabling problems. In [163] fitness distance correlation analysis is performed on a hyper-heuristic framework for dispatching rules for production scheduling. Maden et al. [143] perform autocorrelation analysis on a landscape for a perturbative hyper-heuristic framework.

2.7 Summary

This chapter presented a review of the background information. The chapter introduced data classification outlining the types of classification problems, types of classifiers and metrics for measuring the effectiveness of classifiers and classification algorithms. A general review of evolutionary algorithms was presented including a detailed review of the evolutionary algorithms of interest in this thesis, specifically, a genetic algorithm, grammatical evolution and genetic programming. The application and configuration of genetic programming as a classification algorithm was also described. A detailed outline of the manual design of genetic programming classification algorithms together with the disadvantages of this approach to design were presented. The concepts of automated design were introduced and related studies that use a similar approach to automated design was also outlined. The chapter concludes by presenting fitness landscape analysis techniques. The next chapter presents the methodology used to meet the objectives set out in **Chapter 1**.

Chapter 3

Methodology

3.1 Introduction

This chapter outlines the research methodology used in this study to meet the objectives set out in section **1.2** of **Chapter 1**. Section **3.2** discusses research methods commonly used in computer science while section **3.3** discusses the research method chosen for this study and how it is applied. Section **3.4** outlines how a comparative analysis will be done to achieve some of the objectives of this study. Section **3.5** presents the classification problem instances to be used in this study. The technical specifications and summary are provided in sections **3.6** and **3.7** respectively.

3.2 Research Methodologies

According to Demeyer [50] selecting a research methodology to use when conducting research in the field of computer science is not an easy task. The author argues that this is because computer science has its foundations based on a number of other fields. Mathematics, for example, has strong links to computer science. Would this then qualify the application of mathematical research methodologies such as the concepts of axioms, postulates and proofs to computer science problems? Engineering, it can also be argued has strong links to computer science thus can the concepts of quantification, measurements and comparison be applied in computer science as in engineering. A number of researchers have proposed a wide range of research methodologies that can be used in computer science.

Oates [161] presents *action research* and *design and creation* as being suitable research methods for computer science. *Action research* is described as a strategy that involves a *plan-act-reflect* cycle which is deemed a suitable research method for solving real-world

computing problems. *Design and creation* which is an iterative process involving five steps *awareness, suggestion, development, evaluation* and *conclusion* is described as being suitable for the creation of new software artefacts.

Johnson [107, 108] identifies four research methodologies that are also commonly used in computer science and outlined as follows: *proof by demonstration* is described as bearing similarities to methods used in engineering it works by iteratively testing and refining a computer system towards the desired solution or until no further improvements can be achieved. At each iteration, if the desired solution is not achieved the reasons for failure are sought and used as corrective measures to make adjustments of the system towards the desired outcome. *Empiricism* which is described as having four stages namely, *hypothesis generation, method identification, results compilation* and *conclusion* is mainly used to evaluate a certain hypothesis. The results stage may involve statistical tests being applied to refute or accept the hypothesis. *Mathematical proof* applies the concepts of formal mathematical techniques to evaluate a specific hypothesis. *Hermeneutics* is a research method adopted from social sciences and involves deploying and observing the proposed computer system in an environment where it is meant to function.

Oates [161] asserts that depending on the research being undertaken different research methods can be used to meet different objectives or one research method can be used to meet several objectives. The main aim of the study presented in this thesis is to evaluate whether evolutionary algorithms (GA or GE) are suitable for automating the design of GP classification algorithms. This requires that a GA system and a GE system be developed and implemented thus the proof by demonstration research methodology is deemed to be the most appropriate. The next section outlines how the proof by demonstration research methodology is used to meet some of the objectives set out in **Chapter 1**.

3.3 The Proof by Demonstration Methodology

3.3.1 Objectives One and Two

Objective one is to *automate the design of genetic programming classification algorithms using a genetic algorithm*.
Objective two is to *automate the design of genetic programming classification algorithms using grammatical evolution*.
To achieve these objectives two systems will be developed and implemented namely, a GA system and a GE system. The proof by demonstration methodology specifies that once a system has been implemented it should be tested and the results of the testing used to

refine the system towards the desired outcome. The process alternates between testing and refinement until the desired outcome is met or until no further improvements can be achieved. For this study, each of the two systems is implemented with an initial set of primitives and parameters. The steps to be taken using the proof by demonstration approach to develop each of the systems are outlined as follows:

GA System The main goal of the genetic algorithm system is to determine design decisions for GP classification algorithms, namely design decision 1 (determine parameters), 2 (determine operators) and 3 (determine control flow). The GA searches in a space of design decisions for the best configuration of GP classification algorithms. Each GA individual will represent a GP classification algorithm configuration specifying design decisions 1, 2 and 3. The initial configuration of the GA system will be based on parameters obtained from related studies outlined in section **2.5.2** of **Chapter 2** where the parameters are shown to have worked well in similar studies like the one presented in this thesis. The following steps of the proof by demonstration methodology will be performed:

i) develop and implement the GA system for automated design.

ii) test the GA system using randomly selected binary and multiclass classification problem instances from the list of problem instances outlined in section **3.5**.

   - Due to the stochastic nature of a GA the system will be tested several times (at least ten) using different random number generator seeds for each test for each problem.

iii) If the GA system fails to evolve a valid solution for at least one seed for each problem tested then changes will be made to one or more of the following:

   - genetic algorithm parameters and features such as representation, genetic operators and probability rates, selection method and associated values, control method, selection method, fitness function and the number of generations.
   - components and elements of the GP design decisions search space such GP representation, initial population generation method, initial tree depth, selection method, genetic operators, genetic operator application rates, offspring depth, number of generations.

iv) The revised GA system will be tested and if any failures occur these will be reported and the system will be revised again until the desired outcome is met.

Section **5.3** of **Chapter 4** provides a detailed outline of the proposed approach of using a GA to automate the design of GP classification algorithms.

GE System Similar to the GA system the main goal of the grammatical evolution system is to determine design decisions 1, 2 and 3 for GP classification algorithms. Using a grammar each GE genotype will be mapped on to a GE phenotype representing a GP configuration. The initial configuration of the GE system will be based on parameters obtained from related studies outlined in section **2.5.2** of **Chapter 2** where the parameters are shown to have worked well in similar studies which used GE in a similar approach as the one proposed in this thesis. The following steps of the proof by demonstration methodology will be performed

i) develop and implement the GE system for automated design.

ii) test the GE system using randomly selected binary and multiclass classification problem instances from the list of problem instances outlined in section **3.5**.

   - due to the stochastic nature of GE the system will be tested several times (at least ten) using different random number generator seeds for each test for each problem.

iii) If the GE system fails to evolve a valid solution for at least one seed for each problem tested then changes will be made to one or more of the following:

   - grammatical evolution parameters and features such as representation, genetic operators and probability rates, selection method and associated values, selection method, fitness function and number of generations.
   - GE grammar specifying components and elements of the GP design decisions search space such as GP representation, initial population generation method, initial tree depth, selection method, genetic operators, genetic operator application rates, offspring depth, number of generations.

iv) The revised GA system will be tested and if any failures occur these will be reported and the system will be revised again until the desired outcome is met.

Section **6.3** of **Chapter 4** provides a detailed outline of the proposed approach of using GE to automate the design of GP classification algorithms.

3.3.2 Objective Three and Four

Objective three *to compare the effectiveness of genetic programming classifiers evolved by a genetic algorithm to manually designed genetic programming classifiers.*

Objective four *to compare the effectiveness of genetic programming classifiers evolved by grammatical evolution to manually designed genetic programming classifiers.*
To meet objectives three and four a manually designed GP classification algorithm system will be developed. The following steps of the proof by demonstration methodology will be followed:

GP Classification algorithm system :

i) manually develop and implement a GP classification algorithm system.

ii) test the GP system using randomly selected binary and multiclass classification problem instances from the list of problem instances outlined in section **3.5**.

- due to the stochastic nature of GP the system will be tested several times (at least ten) using different random number generator seeds for each problem.

iii) If the GP system fails to evolve a valid solution for at least one seed for each problem tested then changes will be made to one or more of the following:

- representation
- initial population generation method
- initial tree depth
- selection method
- genetic operators
- genetic operators application rates
- offspring depth
- number of generations
- type of classifiers

iv) The revised GP system will be tested and if any failures occur these will be reported and the system will be revised again until the desired outcome is met.

Section **4.2** of **Chapter 4** provides a detailed outline of the manual GP approach used in this study.

3.4 Comparative Analysis

To achieve objectives three, four, five, six, seven and eight a comparative analysis of the results of the performance of the automated approaches (GA and GE) and the manual approach will be conducted. This entails conducting a number of experiments to measure the performance of each approach. The performance will be evaluated with respect to the predictive accuracy and design time obtained from applying each algorithm to real-world binary and multiclass classification problems in the following cases:

1. across varied problem domains summarised in Tables **3.1** and **3.2**.
2. in specific domains namely, computer security and financial forecasting.

3.4.1 Experiments

Experiments will be conducted using the three approaches, manual, GA and GE to enable comparisons. An overview (note a detailed description is provided in **Chapters 4, 5 and 6**) of the experiments to be conducted is described as follows:

3.4.1.1 Manual Approach

Three experiments will be conducted for the manual GP approach. These will be differentiated by the type of classifiers used. Experiment 1 will be configured to evolve arithmetic tree classifiers, experiment 2 will generate logical tree classifiers and experiment 3 will evolve decision tree classifiers. For each dataset and for each experiment 30 runs will be conducted using different random number generator seeds. As indicated earlier classification consists of training and testing. The best training classifier will then be applied to the test set resulting in a test predictive accuracy value.

3.4.1.2 Automated Design Approaches

For each dataset 30 runs of the GA automated design approach will be performed resulting in the best test predictive accuracy result. Similarly, 30 runs of the GE automated design approach will be performed for each dataset this will also result in the predictive accuracy result.

3.4.2 Statistical Tests

This section outlines how objectives three, four, five and six will be met. The results obtained from conducting the experiments on the stipulated datasets will be used to achieve objectives

three, four, five and six. A comparative analysis will be conducted using the results. The non-parametric Friedman test discussed in section **2.2.2** of **Chapter 2** will be used to determine the statistical significance of the obtained results. An analysis of the configurations evolved by the automated design approaches will also be presented to meet objective six. To achieve objective seven the manual design time will be compared to the automated design time.

3.4.3 Fitness Landscape Analysis

To meet objective eight fitness landscape analysis will be carried out using the autocorrelation analysis method described in section **2.6.2.1** of **Chapter 2**. To carry out the analysis a number of fitness landscape analysis experiments (described in section **6.4** of **Chapter 6**) need to be conducted. The data to be used for the fitness landscape analysis experiments is presented in section **3.5.3**.

3.5 Datasets

The proof by demonstration methodology requires systems to be implemented to prove or disprove the hypothesis. This entails experiments to be conducted to evaluate the systems. Problems instances are required to demonstrate the effectiveness of the proposed approaches. Thus this section presents the problem instances to be used in the experiments. As mentioned earlier the experimental analysis will be conducted over two scenarios across datasets from multiple problem domains and on problems from a specific domain.

3.5.1 Multiple Problem Domain Datasets

The experiments for the first case will make use of publicly available datasets containing binary and multiclass classification problems. These datasets are obtained from the UCI machine learning repository [15]. Tables **3.1** and **3.2** contains a listing of the 22 datasets to be used.

dataset	# attributes	# numeric	#nominal	# instances
australian credit data	14	8	6	690
appendicitis	7	7	0	106
breast cancer (Ljubljana)	9	0	9	277
cylinder band	19	19	0	365
diabetes(pima)	8	8	0	768
german credit data	20	7	13	1000
heart disease	13	13	0	270
hepatitis	19	19	0	80
liver disease(Bupa)	6	6	0	345
mushroom	22	0	22	5644
tictactoe	9	0	9	958

Table 3.1 Summary of binary datasets

dataset	# attributes	# numeric	#nominal	# instances	# classes
balance	4	4	0	625	3
post-operative	8	0	8	87	3
car	6	0	6	1728	4
lymphography	18	3	15	148	4
cleveland	13	13	0	297	5
page-block	10	10	0	5472	5
dermatology	34	34	0	358	6
flare	11	0	11	1066	6
glass	9	9	0	214	7
zoo	16	0	16	101	7
ecoli	7	7	0	336	8

Table 3.2 Summary of multiclass datasets

These problem instances are from varied problem domains. The data is from the following problem domains: medical(e.g breast cancer, diabetes), financial(e.g australian credit data), botany(e.g mushroom), industry(e.g car), dermatology, zoology(e.g zoo), games(e.g tictactoe), microbiology(e.g ecoli) and publishing(e.g page-block). This will allow the evaluation of the effectiveness of the proposed approach to generalise across multiple problem domains.

3.5.2 Single Domain Datasets

Two real-world problem domains namely, computer security and financial forecasting are selected for the single domain problem instances. It is widely accepted that data from the cybersecurity domain usually exhibits low volatility compared to data from financial forecasting which exhibits high volatility [62, 125]. Using data from these 2 problem domains will enable the evaluation of the ability of the proposed approach to classify problems with different degrees of volatility.

1. **Cybersecurity**

   - **NSL-KDD 99+20%**
     The widely used NSL-KDD 99+20% [210] dataset was selected and a set of 6 datasets were created. This dataset contains training and testing subsets and each record consists of 42 feature attributes including a class label. The label of each record is one of five main classes namely

     i) normal
     ii) denial of service (dos)
     iii) probe
     iv) user to root (u2r)
     v) root to local (r2l)

     A set of 6 datasets is created by grouping the records based on their classes in a similar approach as in [36]. This is achieved as follows:

     a) set 1 -normal + rest (dos, probe, u2r, r2l)
     b) set 2 -dos + rest (normal, probe, u2r, r2l)
     c) set 3 -probe + rest (normal, dos, u2r, r2l)
     d) set 4 -u2r + rest (dos, probe, normal, r2l)
     e) set 5 -r2l + rest (dos, probe, u2r, normal)
     f) set 6 -five distinct classes (normal, dos, probe, u2r, r2l)

     Each dataset consists of 5000 training instances and 2000 test instances.

2. **Financial forecasting**
   Fifteen stocks were selected from the NASDAQ, NYSE, XETRA and HKSE stock exchanges. A varied selection was made because different industry stock have a varied volatility, for example, stock from the technology sector, is more volatile than stock

from the banking sector. Each dataset comprises of data from 1500 trading days 03/01/2012 to 05/03/2018 (1000 training and 500 test). Each record contains a total of 4 attributes the opening price, highest price, lowest price and the closing price from which the binary class denoting the actual movement of the stock (up or down) on that day is determined. Table **3.3** is a listing of the stocks. The first column presents the name of the stock, the second and third columns number of days the data was split for training and testing. The fourth column identifies the sector of industry the stock emanates from and the fifth column is the exchange the stock is listed on.

dataset	#training(days)	#test(days)	sector	source
adobe	1000	500	technology	NASDAQ
amazon	1000	500	technology	NASDAQ
americanExpress	1000	500	financial	NYSE
barclays	1000	500	financial	NYSE
centerPoint	1000	500	energy	NYSE
dominosPizza	1000	500	food	NYSE
entergy	1000	500	energy	NYSE
horizonPharm	1000	500	pharmaceutical	NASDAQ
pfizer	1000	500	pharmaceutical	NYSE
mcDonalds	1000	500	food	NYSE
microsoft	1000	500	software	NASDAQ
SAP	1000	500	software	XETRA
standardChartered	1000	500	financial	HKSE
timeWarner	1000	500	entertainment	NYSE
waltDisney	1000	500	entertainment	NYSE

Table 3.3 Financial forecasting datasets

3.5.3 Fitness Landscape Analysis Datasets

The following datasets were selected to be used for the fitness landscape analysis comparison for the automated design approaches. The selections were done based on

domain		datasets	
binary class	australian credit	hepatitis	tictactoe
multiclass	balance	dermatology	ecoli
cybersecurity	set 1	set 3	set 6
financial forecasting	adobe	barclays	pfizer

Table 3.4 Financial forecasting datasets

the following criteria:

- binary class - varied problems domains i.e. financial, medical and game data.
- multiclass - varied problem domains and varying classes.
- cybersecurity - normal set, malicious set and multiclass set.
- financial forecasting - varied volatility i.e. adobe most volatile, barclays and pfizer medium and least volatile respectively.

3.5.4 Data Pre-processing

All continuous attributes of the datasets are normalised using equation **2.9** outlined in section **2.2.3.2** of **Chapter 2** after which discretisation is performed using the *equal frequency interval*. Data is partitioned using a 70% training and 30% test split for those datasets that do not have a training and test subset.

3.6 Technical Specification

The specification of the computer used to develop the software is as follows: Intel(R) Core(TM) i7-6500U CPU @ 2.6GHz with 16GB RAM running 64 bit Linux Ubuntu. The IBM SPSS statistical package is used for data discretisation. The simulations were performed using the CHPC (Centre for High Performance Computing) Lengau cluster large que. Java 1.8 was used as the software development platform on the Netbeans 8.1 Integrated Development Environment. R was used for autocorrelation analysis and calculation of the statistical significance of the differences of the test results.

3.7 Summary

This chapter presented the methodology to be used to investigate the automated design of genetic programming classification algorithms. The chapter also included a description of the problem instances to be used to carry out an evaluation of the automatically designed GP classifiers. The next chapter presents the manual design approach of genetic programming classification algorithms to be used in this thesis.

Chapter 4

Manual Design of Genetic Programming Classification Algorithm

4.1 Introduction

In this chapter, the manual design of the generational Genetic Programming classification algorithm used in this study is presented in detail. This is followed by an outline of the parameter tuning process and a presentation of the parameter values obtained from parameter tuning. The chapter is organised as follows: sections **4.2** to **4.6** present the manual GP algorithm while section **4.7** presents the parameter values obtained from parameter tuning for the considered problem instances. Section **4.8** provides a summary of the chapter.

4.2 Genetic Programming Classification Algorithm

The manual GP classification algorithm uses a generational GP algorithm. Algorithm **5** is an illustration of the step by step flow of the manual GP algorithm used in this study.

Algorithm 5 Generational Genetic Programming

```
1: Create an initial population of individuals (classifiers)
2: Apply each individual and establish its fitness
3:    while termination condition not met do
4:       Select fitter individuals to participate in reproduction
5:       Create new individuals using genetic operators and update the population
6:       Apply each individual and establish the fitness
7:    end while
8: return best individual
```

Each individual of the population is a classifier induced by the GP algorithm.

4.2.1 Classifier Type

The manual GP algorithm can evolve one of three types of classifiers namely, arithmetic tree classifiers, logical tree classifiers and decision tree classifiers. These three tree classifier types are illustrated in section **2.4.1**. of Chapter **2**. The type of classifier evolved is determined by the contents of the function and terminal sets. For this study the function and terminals sets were defined as follows:

- **arithmetic tree type**
  - function set ={ +,-,*,/(protected) }
  - terminal set = {attributes from fitness cases}
- **logical tree type**
  - function set ={AND,OR,EQUAL,DIFFERENT,NOT }
  - terminal set = {attributes from fitness cases}
- **decision tree type**
  - function set ={attributes from fitness cases }
  - terminal set = {class 0, class 1}(incremental for multiclass classification problems)

4.2.2 Fitness Function

Predictive accuracy is used as the fitness function. The predictive accuracy of each individual is evaluated as the sum of all correctly classified fitness cases divided by the total number of fitness cases. The fitness is assigned as a percentage.

4.2.3 Multiclass Classification Method

Static class boundary determination (SCBD) discussed in section **2.4.7** of **Chapter 2** is used for multiclass classification problems. The class boundaries are defined as follows:

i) three classes: Class1[-inf,-1], Class2[-1,1], Class3[1,inf].

ii) four classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,inf].

iii) five classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,inf].

iv) six classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,8], C6[8,inf].

v) seven classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,8], C6[8,10] C7[10,inf].

vi) eight classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,8], C6[8,10] C7[10,12] C8[12,inf].

4.3 Initial Population Generation

The first step of the algorithm is to create an initial population of GP individuals. Individuals are created by randomly selecting elements from a combination of the respective function and terminal sets outlined in section **4.2.1**. The function set and terminal set used corresponds to the chosen tree type for the individuals i.e. arithmetic trees, logical trees or decision trees. The ramped half-and-half method which is the most commonly used initial population generation method is used for initial tree generation [118, 175].

4.4 Selection

Tournament selection is used to select individuals from the current population to act as parents for the creation of offspring for the next generation. The values of the tournament size are discussed in section **4.7**.

4.5 Genetic operators

Subtree crossover and grow mutation are used as the genetic operators for the manual GP algorithm. The associated application rates are discussed in section **4.7**.

4.6 Algorithm Termination

Two stopping criteria, maximum number of generations and maximum fitness are defined. If the maximum number of generations is reached or the predictive accuracy is the maximum possible (100%) then the algorithm will terminate and the best classifier will be returned. Values used for the maximum generation are discussed in section **4.7**.

4.7 Parameter Tuning

In the manual design approach, parameter tuning is conducted using the iterative trial and error approach, discussed in section **2.4.8** of **Chapter 2**. For each parameter, an iterative approach is followed where one parameter is varied at a time while keeping the others constant. Trial runs are performed for each parameter value. The final parameter value is obtained from the trial run that achieves the highest predictive accuracy. This process is repeated for the next parameter until all parameters have been tuned.

As discussed earlier under manual design three experiments are conducted distinguished from each other by the type of classifier evolved as follows:

- **experiment 1** - arithmetic tree classifiers
- **experiment 2** - logical tree classifiers
- **experiment 3** - decision tree classifiers

For each experiment, parameter tuning is carried out to determine the parameter values to be used. In the following section, the values obtained from parameter tuning for the selected datasets are presented.

4.7.1 UCI Datasets

4.7.1.1 Binary classification

A similar approach to that used in [184] and [142] where parameter tuning was carried out for each dataset was followed to tune parameter values for binary classification problems listed in Table **3.1**. The commonly used values for GP classification algorithms discussed in section **2.4.8.1** of **Chapter 2** were used as the starting point values for the tuning process. From parameter tuning a population size of 300 was found to adequately represent the search space. The algorithm terminates when the maximum predictive accuracy has been achieved or when the maximum number of generations have occurred. Three hundred generations were found to be adequate to achieve algorithm convergence on the considered datasets. The initial tree depth parameter was tuned in the range from 2 to 20 while the maximum offspring depth parameter range was from 4 to 30 for arithmetic and logical trees and 4 to 10 for decision trees. A tournament selection size range of 2 to 20 was used. It is recommended that the crossover application rates should be higher than the mutation application rate [118]. Following this recommendation, the application rate for crossover was considered in the range 50% to 90% and the mutation range 50% to 10%. The mutation depth was tuned

in the range 2 to 10. The tuned parameter values for each dataset are presented in Table **4.1**, Table **4.2** and Table **4.3**. The first column of each table represents a parameter and subsequent columns are the datasets indexed as follows: i-*australian credit*, ii-*appendicitis*, iii-*breast cancer*, iv-*cylinder band*, v-*diabetes* ,vi-*german credit*, vii-*heart*, viii-*hepatitis*, ix-*liver disease*, x-*mushroom* and xi-*tictactoe*.

- **Arithmetic trees**

parameter	dataset										
	i	ii	iii	iv	v	vi	vii	viii	vix	x	xi
pop size	300	300	300	300	300	300	300	300	300	300	300
init tree depth	8	8	3	3	8	6	8	4	7	8	5
max offsp depth	10	10	10	15	10	10	10	12	15	10	10
selection size	4	12	4	4	4	4	8	4	8	12	4
crossover rate	60	90	80	60	60	70	70	65	85	60	90
mutation rate	40	10	20	40	40	30	30	35	25	40	10
mutation depth	5	6	6	6	6	6	4	8	4	4	6
number of gens	300	300	300	300	300	300	300	300	300	300	300

Table 4.1 Arithmetic parameter values

- **Logical trees**

parameter	dataset										
	i	ii	iii	iv	v	vi	vii	viii	vix	x	xi
pop size	300	300	300	300	300	300	300	300	300	300	300
init tree depth	8	8	4	2	4	3	6	3	8	6	4
max offsp depth	10	10	12	10	10	10	10	10	10	12	12
selection size	8	8	8	4	4	4	4	8	8	8	4
crossover rate	80	85	60	60	70	80	80	80	70	80	60
mutation rate	20	15	40	40	30	20	20	20	30	20	40
mutation depth	5	4	5	6	6	4	4	6	5	6	6
number of gens	300	300	300	300	300	300	300	300	300	300	300

Table 4.2 Logical parameter values

- **Decision trees**

parameter	dataset										
	i	ii	iii	iv	v	vi	vii	viii	vix	x	xi
pop size	300	300	300	300	300	300	300	300	300	300	300
init tree depth	2	2	2	3	3	2	2	2	4	3	2
max offsp depth	5	4	5	5	5	5	4	5	5	5	5
selection size	8	8	4	4	4	4	4	4	4	4	8
crossover rate	80	70	70	80	70	80	70	70	70	70	80
mutation rate	20	30	30	20	30	30	30	30	30	30	20
mutation depth	3	3	3	3	3	3	4	3	4	4	3
number of gens	300	300	300	300	300	300	300	300	300	300	300

Table 4.3 Decision tree parameter values

4.7.1.2 Multiclass classification

The more commonly used standard approach [20] of using one problem from the application domain to determine parameter values and using them on all problems was followed for tuning values for the multiclass classification datasets. This enables evaluation of the ability of the algorithm to generalise across problem domains. Using the parameter values found from literature as initial values a population size of 300 was found to be sufficient to represent the search space, initial tree depth was tuned in the range 2 to 20, maximum offspring depth in the range of 4 to 30 for the arithmetic and logical trees while the range of 4 to 10 was used for the decision trees.

parameter	arithmetic	logical	decision tree
population size	300	300	300
initial tree depth	3	3	2
max offspring depth	8	10	5
tournament size	8	10	8
crossover rate	80	90	70
mutation rate	20	10	30
mutation offspring depth	6	8	4
maximum generations	300	300	300

Table 4.4 Multiclass manual GP parameters

The crossover application rate was tuned in the range 50% to 90% and the mutation rate in the range 10% to 50%. The tournament size was tuned in the range 2 to 20, mutation offspring depth in the range 2 to 10 and 300 generations were set at the termination criterion. Table **4.4** presents the values obtained from parameter tuning. The first column represents parameters for arithmetic tree classifiers the second column represents logical trees classifiers and the last column decision tree classifiers.

4.7.2 Cybersecurity Datasets -NSL-KDD99 20% Values

Similarly, trial runs were used to tune parameter values to be used for binary and multiclass problems from the NSL-KDD99 20% datasets. A population size of 300 was found to be sufficient to represent the search as values greater than 300 did not yield significant improvements. The initial tree depth was tuned in the range 2 to 20, maximum offspring depth in the range of 4 to 30 for the arithmetic and logical trees while the range of 4 to 10 was used for the decision trees. The crossover application rate was tuned in the range 50% to 90% and the mutation rate in the range 10% to 50%. The tournament size value was tuned in the range 2 to 20, mutation offspring depth in the range 2 to 10 and 200 generations were found to be sufficient as the termination criterion. Tables **4.5** and **4.6** present the parameter values obtained for the NSL-KDD99 datasets.

parameter	arithmetic	logical	decision tree
population size	300	300	300
initial tree depth	4	3	2
max offspring depth	10	8	8
tournament size	8	10	8
crossover rate	80	70	90
mutation rate	20	30	10
mutation offspring depth	6	10	8
maximum generations	200	200	200

Table 4.5 NSL-KDD binary problem parameters

parameter	arithmetic	logical	decision tree
population size	300	300	300
initial tree depth	4	4	2
max offspring depth	12	8	6
tournament size	12	8	4
crossover rate	80	80	85
mutation rate	20	20	15
mutation offspring depth	4	8	4
maximum generations	200	200	200

Table 4.6 NSL-KDD multiclass parameter

4.7.3 Financial forecasting datasets

A parameter tuning approach similar to the cybersecurity datasets was followed for tuning parameter values for the financial forecasting datasets. Table 4.7 is a listing of the parameter values obtained for the financial forecasting datasets.

parameter	arithmetic	logical	decision tree
population size	300	300	300
initial tree depth	3	4	2
max offspring depth	12	8	5
tournament size	4	6	8
crossover rate	90	85	70
mutation rate	10	15	30
mutation offspring depth	4	5	4
maximum generations	200	200	200

Table 4.7 Financial forecasting parameters

4.8 Summary

This chapter outlined the manual design of genetic programming algorithms used in this study. The chapter also presented the parameter values obtained through the parameter tuning process. It can be argued that a different designer can design a different manual GP system or develop it quicker. As pointed out the manual design approach used in this thesis used initial values obtained from the literature and iteratively improved the system. This was done to

eliminate any form of bias and to follow the standard procedure of GP manual design. This was done in order to produce a manual design that would evolve classifiers that would allow a fair comparison with the automated design approach. The next chapter outlines the genetic algorithm approach to automated design for genetic programming classification algorithms.

Chapter 5

Design of GP classification algorithms using a Genetic Algorithm

5.1 Introduction

In this chapter, the approach for automating the design of genetic programming classification algorithms using a genetic algorithm is presented. Firstly the considered GP design decisions are outlined followed by a description of the GA automated design approach. The chapter is structured as follows: section **5.2** outlines the GP design decisions. This is followed by section **5.3** which describes the implementation details of the automated design using a GA. Finally section **5.4** presents a summary of the chapter.

5.2 GP Design Decisions

One of the main objectives of this study is to use a GA to design generational GP classification algorithms. This is achieved by using a GA to make design decisions and evaluating different GP classification algorithm designs. To enable this a GA is used to search in the design space for the most suitable configurations of GP classification algorithms. Based on the outline of design decisions presented in section **2.5.1.1** of **Chapter 2**, the following design decisions are considered for Genetic Programming classification algorithms for this study.

5.2.1 Determination of Parameters Values

As previously stated a GP classification algorithm requires categorical and numerical parameters. Both these types of parameters need to be specified. From the discussion presented in

section **2.4.8** of **Chapter 2** the parameters and options of possible values that can be assigned to each design decision are outlined as follows:

5.2.1.1 Categorical Parameters

a) **Tree type**
The three commonly used tree types by GP for data classification algorithms are the options made available for this design decision i.e. arithmetic trees, logical trees, and decision trees.

b) **Initial population generation method**
Initial population generation is performed using either the full method, grow method or ramped half-and-half method. These are the three methods available for this design decision.

c) **Fitness function**
Five fitness functions are specified for this design decision. These are based on the discussions presented in sections **2.2.1** and **2.4.3** of **Chapter 2** and are outlined as follows:

i) *accuracy*

$$accuracy = \frac{t_p + t_n}{t_p + t_n + f_p + f_n} \tag{5.1}$$

ii) $f_{measure}$

$$f_{measure} = 2(\frac{precision * recall}{precision + recall}) \tag{5.2}$$

iii) *weighted accuracy*
For this fitness function, a weighted fitness function is specified where *accuracy* and $f_{measure}$ are weighted and defined as follows:

$$weighted_{acc} = 0.5 * accuracy + 0.5 * f_{measure} \tag{5.3}$$

iv) *random weighted accuracy*
This fitness function is similar to the weighted accuracy, the difference is that the weight of the contribution of *accuracy* and $f_{measure}$ are randomly set before evaluation. The function is defined by the following equation:

$$weighted_{rand} = rand * accuracy + (1 - rand) * f_{measure} \tag{5.4}$$

v) *true positive rate*

$$tpr = \frac{t_p}{t_p + f_n} \tag{5.5}$$

d) **Selection method**
Two options tournament and fitness proportionate selection are the options specified for this design decision.

5.2.1.2 Numerical Parameters

a) **Population size**
Three population size values 100, 200, and 300 are available for this parameter. These values are deemed to adequately represent the search space.

b) **Maximum initial tree depth**
The maximum initial tree depth parameter value is randomly assigned from the range [2–15] for arithmetic trees and logical trees and [2–8] for decision trees. As pointed out in [20] decision trees can grow exponentially leading to high computational costs. Hence the commonly used range of maximum tree depths for decision trees are significantly narrower than for the other type of trees.

c) **Tournament selection size**
This parameter is randomly assigned any value in the range [2–10].

d) **Maximum offspring depth**
The maximum offspring depth parameter value is randomly assigned from the range [2–15] for arithmetic trees and logical trees and [2–8] for decision trees.

e) **Mutation depth**
The mutation depth parameter value is randomly assigned from the range [2–6].

f) **Termination**
Two termination criteria are defined namely, desired fitness and maximum number of generations. The desired fitness value is 100% while the maximum number of generations can be set in the range 50-200.

5.2.2 Determination of Genetic Operators

A generational GP algorithm makes use of genetic operators to evolve offspring for each subsequent generation after initial population generation. For this design decision three genetic operators are specified as follows:

1. **Crossover**
   Subtree crossover as described in section **2.3.2.4** of **Chapter 2** is specified for this design decision.

2. **Mutation**
   The two commonly used mutation operators are specified for this design decision namely, *grow mutation* and *shrink mutation*. These are implemented as described in section **2.3.2.4** of **Chapter 2**.

3. **Creation**
   This operator functions by replacing the current population with a new population. Genetic material from the previous population is not used in the regeneration of new individuals.

To evolve the next generation a combination of the genetic operators is used. Each combination has a specific application rate of a genetic operator. The genetic operator combination options are presented as follows:

a) **Crossover and mutation**
   For this combination, the application rates are randomly assigned from the range of [0%–100%]. For example, if crossover is randomly assigned an application rate of 71% the mutation application rate is assigned a value of 29%. The sum of the crossover application rate and mutation application rate should add up to 100%.

b) **100% crossover**
   This combination regenerates the whole population using only crossover. Although combination a) can achieve this crossover rate the probability of it being selected is very low (1%) and therefore it is explicitly presented as an option.

c) **100% mutation**
   This combination regenerates a new population using mutation only. Similar to 100% crossover this value of the mutation rate can be set by combination a) but the probability is also low therefore it is explicitly presented as an option.

d) **Crossover and mutation preset rates**
This combination makes available a range of rates which are multiples of 10. A pair will be randomly selected from the range and both crossover and mutation are applied using the selected rates. The range is defined as follows (crossover, mutation) [10,90; 20,80; 30,70; 40,60; 50,50; 60,40; 70,30; 80,20; 90,10]. For example, if the first pair is selected the new generation will be evolved by 10% crossover and 90% mutation.

e) **100% mutation and (rand%)crossover**
This combination applies 100% mutation to the current population and then a random rate of crossover in the range of [0%–100%] is applied to the new population.

f) **100% crossover and (rand%)mutation**
This combination applies 100% crossover to the current population and then a random application rate of mutation in the range [0%–100%] is applied to the new population.

g) **Creation**
This option is used to replace the current population with a new population.

5.2.3 Determination of the Control Flow

This design decision determines the order in which the processes of the algorithm occur. After initial population generation, a combination of genetic operators outlined in section **5.2.2** are randomly chosen to evolve the next generation. One of two options is randomly selected for this design decision. The control flow can be *fixed* or *random*. If the control flow is set to *fixed* a selected combination of genetic operators will be used for regeneration until the algorithm terminates. If the selection is *random* a different combination of genetic operators is randomly chosen to evolve each subsequent generation until algorithm termination.

5.3 Automated Design of GP Classification Algorithms using a Genetic Algorithm

This section outlines how a genetic algorithm is used to automate the design of GP classification algorithms. As defined earlier in **Chapter 2** the term configuration in this thesis is used to refer to a set of values for all the design decisions of an algorithm. Thus for automating the design of GP classification algorithms using a GA we use the GA to search for the best configuration of GP classification algorithms for a problem at hand. The GA makes the

design decisions for configuring the GP classification algorithm. In this thesis, the automated design of GP classification algorithms using a GA approach is termed *autoGA*.

5.3.1 Genetic Algorithm for *autoGA*

A generational GA is used by the *autoGA* approach. Algorithm **6** is an outline of the algorithm flow. A GA individual is a GP classification algorithm configuration. The GA searches for a configuration that will yield the best classifier. If after initial population generation and evaluation, the desired outcome is not met individuals undergo crossover and mutation, producing new GP configurations.

Algorithm 6 Generational Genetic Algorithm

```
1: Create initial population
2: Calculate fitness of all individuals
3:    while termination condition not met do
4:       Select fitter individuals for reproduction
5:       Recombine individuals
6:       Mutate individuals
7:       Evaluate fitness of all individuals
8:       Generate a new population
9:    end while
10: return best individual
```

The fitness of each GA individual is evaluated by using the configuration to configure GP classification algorithms. GP is then used to solve a classification problem. The accuracy of the best testing GP evolved classifier is assigned as the fitness of the GA individual. This process is repeated from generation to generation until the maximum number of specified generations is met or the desired accuracy is achieved. Figure **5.1** provides an overview of the proposed *autoGA* approach.

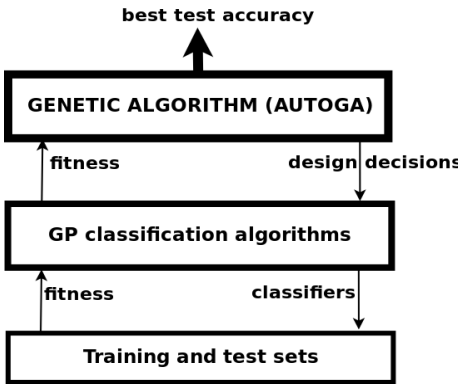


Fig. 5.1 AutoGA overview

5.3.1.1 Representation

Each element of the population is a fixed length chromosome representing the design decisions that need to be made for the GP classification algorithm. Each gene represents one of the design decisions specified in section **5.2**

$\mathbf{g}_0$	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$	$\mathbf{g}_5$	$\mathbf{g}_6$	$\mathbf{g}_7$	$\mathbf{g}_8$	$\mathbf{g}_9$	$\mathbf{g}_{10}$	$\mathbf{g}_{11}$	$\mathbf{g}_{12}$	$\mathbf{g}_{13}$
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Fig. 5.2 AutoGA chromosome

Figure **5.2** is an illustration of the structure of an *autoGA* chromosome. An individual is a chromosome made up 14 genes labelled from $\mathbf{g}_0$ to $\mathbf{g}_{13}$. Integer values are used to encode the genes. As argued in [63] integer encoding provides a more precise and accurate representation of real-world problems. Integer values are therefore used to represent the design decisions for GP classification algorithms. The first gene $\mathbf{g}_0$ of the chromosome, indexed by 0 represents the tree type (classifier type) i.e. arithmetic tree, logical tree or decision tree. The second gene $\mathbf{g}_1$ represents the population size parameter, the third gene $\mathbf{g}_2$ represents the tree generation method and the maximum initial tree depth parameter is represented by the fourth gene $\mathbf{g}_3$. The fifth gene $\mathbf{g}_4$ represents the maximum offspring depth while $\mathbf{g}_5$ represents the selection method, $\mathbf{g}_6$ the tournament selection size, $\mathbf{g}_7$ the reproduction rates, $\mathbf{g}_8$ the mutation type, $\mathbf{g}_9$ the mutation depth, $\mathbf{g}_{10}$ the genetic operator combinations, $\mathbf{g}_{11}$ the control flow, $\mathbf{g}_{12}$ the fitness function and $\mathbf{g}_{13}$ represents the number of generations. Table **5.1** summarises the options for the different design decisions for the GP classification algorithm described in section **5.2**.

#	**Design Decision**	**Range of possible values**
0	tree type	0 - arithmetic, 1 - logical, 2 - decision
1	population size	100, 200, 300
2	tree generation	0- full, 1- grow, 2- ramped half and half
3	initial tree depth	2 - 15 (decision tree 2 -8)
4	max offspring depth	2 - 15 (decision tree 2 -8)
5	selection method	0 - fitness proportionate, 1 - tournament selection
6	selection size	2 - 10
7	reproduction rates	0 - 100 crossover (mutation = 100-crossover)
8	mutation type	0 - grow mutation,1 - shrink mutation,
9	max mutation depth	2 - 6
10	control flow	0 - fixed 1 - random
11	operator combination	0 - 6
12	fitness type	0 - 4
13	number of generations	binary - [50-200](multiclass- 50,100,200)

Table 5.1 Design decisions and range of values

5.3.2 Initial Population Generation

Initial population generation is the first step of the *autoGA* algorithm. A specified number of GA chromosomes are randomly created. Each gene of a chromosome is randomly assigned a value from the list of possible values for each gene as specified in Table **5.1**. For example the possible values that can be assigned to $\mathbf{g}_0$ is 0,1, or 2. If $\mathbf{g}_0$ is randomly assigned a value of 2 then decision tree classifiers will be evolved by the GP algorithm. If $\mathbf{g}_1$ is randomly assigned a value of 200 this specifies a GP population size of 200. All the genes from $\mathbf{g}_0$ to $\mathbf{g}_{13}$ are assigned values randomly selected from the list of possible values.

0	**100**	**1**	**10**	**12**	**0**	**8**	**87**	**0**	**4**	**0**	**0**	**4**	**100**
$\mathbf{g}_0$	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$	$\mathbf{g}_5$	$\mathbf{g}_6$	$\mathbf{g}_7$	$\mathbf{g}_8$	$\mathbf{g}_9$	$\mathbf{g}_{10}$	$\mathbf{g}_{11}$	$\mathbf{g}_{12}$	$\mathbf{g}_{13}$

Fig. 5.3 AutoGA individual

Figure **5.3** is an example of a typical *autoGA* individual which depicts the GP classification algorithm configuration listed in Table **5.2**.

Parameter	**Value**
tree type	arithmetic
population size	100
tree generation method	grow
initial tree depth	10
max offspring depth	12
selection method	fitness proportionate
crossover rate	87 (mutation = 100-crossover = 13)
mutation type	grow
mutation depth	4
control flow	fixed
operator combination	crossover/mutation
fitness function	*tpr*
number of generations	100

Table 5.2 AutoGA evolved GP parameter settings

5.3.3 Fitness Function and Selection

The fitness of each *autoGA* individual is evaluated by applying the configuration to a GP classification problem. GP classification consists of two phases, a training phase, and testing phase. During the training phase, thirty runs of a GP classification algorithm configured using the *autoGA* configuration is performed using a training set. The best evolved classifier from the thirty runs is then applied to a test set resulting in a test accuracy result. The result of the testing phase is used as the fitness of the *autoGA* individual. Fitness proportionate is used to select parents as described in section **2.3.1.3** of **Chapter 2**

5.3.4 Crossover

For *autoGA*, uniform crossover is used to evolve GP configurations as described in section **2.3.1.4** of **Chapter 2**.

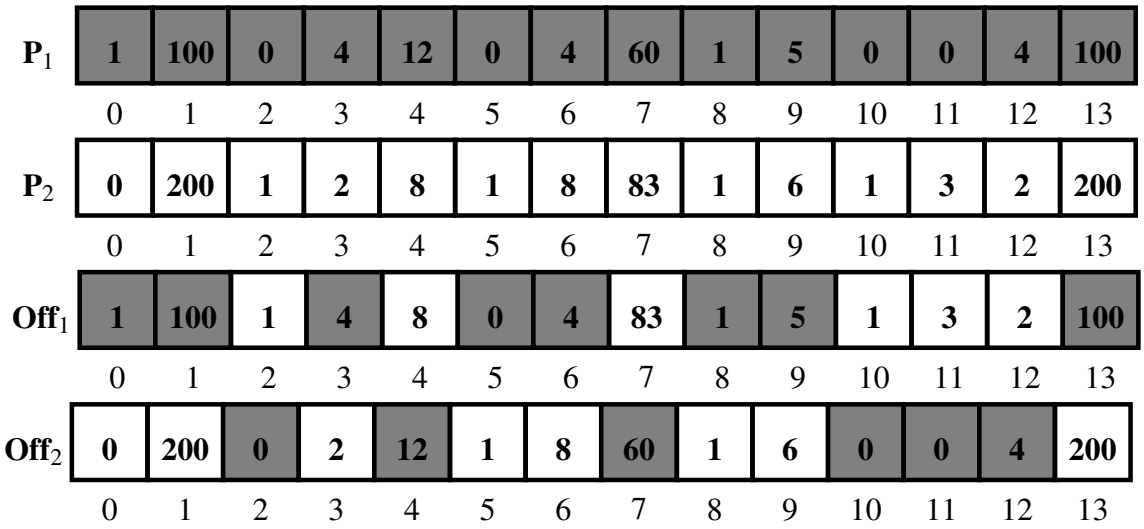


Fig. 5.4 AutoGA uniform crossover

Figure **5.4** is an illustration of an example of the uniform crossover operator implemented in this study. Assuming two individuals P_1 and P_2 are selected to act as parents and applying the crossover operator results in two offspring, namely Off_1 and Off_2. Off_1 consists of genes 0,1,3,5,6,8,9 and 13 from P_1 and genes 2,4,7,10,11 and 12 from P_2. Similarly Off_2 constitutes of genes 2,4,7,10,11 and 12 from P_1 and genes 0,1,3,5,6,8,9 and 13 from P_2. These offspring are added to the population of the next generation.

5.3.5 Mutation

The implemented mutation operator is discussed in section **2.3.1.5** of **Chapter 2**. This is illustrated in Figure **5.5** where an individual, *P* is selected to act as a parent.

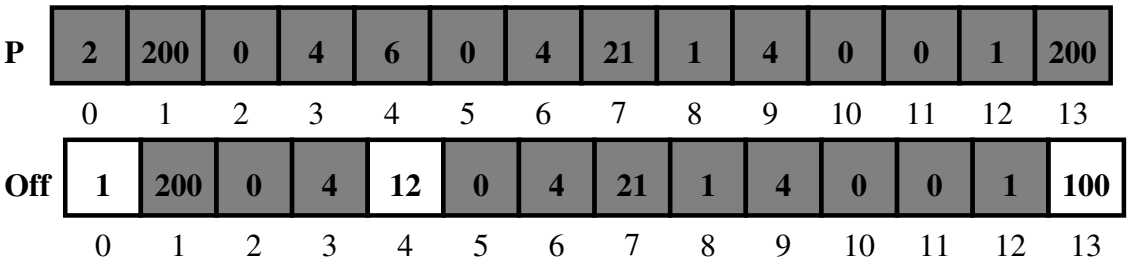


Fig. 5.5 AutoGA random mutation

Applying the mutation operator results in the offspring, *Off*. Genes 0,4 and 13 of parent P are mutated.

5.3.5.1 Elitism

Elitism, as previously stated, involves copying a specified number of the fittest individuals from the current population into the next generation. This concept is adopted in this implementation of the GA.

5.3.6 Termination

Two termination conditions are set for the automated design GA implementation. The algorithm proceeds from generation to generation until a preset number of generations have been completed then the algorithm terminates or if the desired fitness of the GA individuals is achieved. The individual with the best fitness is returned.

5.3.7 *AutoGA* Parameter Settings

Parameter tuning was carried out to establish parameter values for the genetic algorithm using trial runs. A population size of 20 was found to adequately represent the search space. A uniform crossover probability rate of 80% was established to suitable with a mutation rate of 10% . In order to preserve good configurations, an elitism rate of 10% was found to be adequate. Fifty generations were found to be an adequate value for convergence for the multiple problem domain datasets while a value of 30 was found to be suitable for the single domain problems i.e. cybersecurity and financial forecasting. Table **5.3** summarises the parameter settings for the *autoGA* approach.

parameter	value
population size	20
selection method	fitness proportionate
Uniform crossover rate	80%
Bit mutation rate	10%
Elitism	10%
fitness function	accuracy
maximum generations	50 (30 [1])

Table 5.3 AutoGA parameter settings

The *autoGA* approach uses the same function and terminal sets as the manual design configuration presented in section **4.2.1** of **Chapter 4**

[1] NSL-KDD and Financial forecasting

5.4 Summary

In this chapter, the automated design approach for configuring genetic programming classification algorithms using a genetic algorithm was presented. The chapter outlined the design and configuration of a genetic algorithm for automated design. The results of the effectiveness of using a genetic algorithm for automated design are presented in **Chapter 7**. The next chapter presents the grammatical evolution approach for automating the design of GP classification algorithms.

Chapter 6

Design of GP classification algorithms using Grammatical Evolution

6.1 Introduction

In this chapter, the approach for automating the design of genetic programming classification algorithms using grammatical evolution is presented. The Chapter outlines the GP design decisions made by grammatical evolution and then presents the approach of how GE is used to make the considered design decisions. The chapter is structured as follows: section **6.2** outlines the GP design decisions. This is followed by section **6.3** which describes the implementation details of the automated design approach using GE and section **6.4** present the fitness landscape analysis evaluation settings. Finally, section **6.5** presents a summary of the chapter.

6.2 GP Design Decisions

The design of generational GP classification algorithms using grammatical evolution involves GE making design decisions. This requires GE conducting a search for the most suitable configurations of GP classification algorithms in a GP design search space. Grammatical evolution is used to determine the following GP classification algorithm design decisions:

1. determination of parameter values:
    - categorical (as specified in detail in section **5.2.1.1** of **Chapter 5**)
    - numerical (as specified in detail in section **5.2.1.2** of **Chapter 5**)

2. determination of genetic operators (as specified in detail in section **5.2.2** of **Chapter 5**).

3. determination of the control flow (as specified in detail in section **5.2.3** of **Chapter 5**).

6.3 Automated Design of GP Classification Algorithms using Grammatical Evolution

This section outlines how grammatical evolution is used to automate the design GP classification algorithms. As stated earlier GE is used to make GP design decisions resulting in configurations of GP classification algorithms. The configurations are then evaluated on selected classification problems and the best configuration is then selected. In this thesis the GE approach for automating the design of GP classification algorithms is termed *autoGE*.

6.3.1 Grammatical Evolution Algorithm for *AutoGE*

A generational GE algorithm is used by the *autoGE* approach. Algorithm **7** is an outline of the step by step generational GE algorithm used by *autoGE*.

Algorithm 7 Generational Grammatical Evolution

```
Create an initial population of variable length binary strings
Map via a BNF grammar
    a) binary strings to expression using production rules
Evaluate fitness
do while {termination condition not met}
        Select fitter individuals for reproduction
        Recombine selected individuals
        Mutate offspring
        Evaluate fitness of offspring
        Replace all individuals in the population with offspring
end while
return best individual
```

An initial population of GE individuals(genotypes) is randomly created and the fitness of each individual is evaluated. Each individual of the GE population represents a GP classification algorithm configuration. The fitness of an individual is evaluated by using it to configure a GP classification algorithm. The GP classification algorithm is then applied to a classification problem. The predictive accuracy value of the best testing GP classifier

is assigned as the fitness of the GE individual. If after initial population generation and evaluation, the desired outcome is not met individuals undergo crossover and mutation, producing new GE individuals (GP configurations). This process is repeated from generation to generation until the maximum number of specified generations is met or the desired accuracy is achieved. Figure **6.1** provides an overview of the proposed *autoGE* approach.

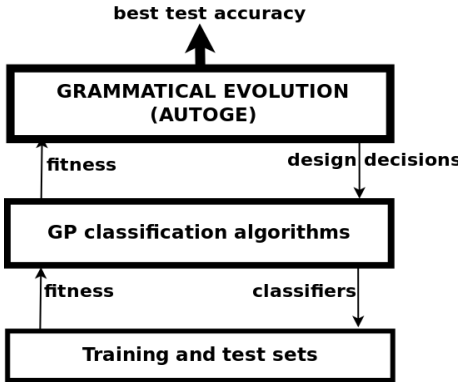


Fig. 6.1 AutoGE overview

6.3.1.1 Representation

Each individual of the population is a variable length chromosome of codons where each codon is an 8-bit binary string. Figure **6.2** is an illustration of an example of an *autoGE* individual which is 15 codons long.

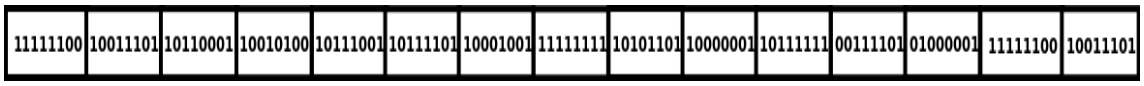


Fig. 6.2 AutoGE individual

6.3.2 Initial Population Generation

AutoGE individuals of the initial population are generated by randomly creating codons. The length of each genotype of the initial population lies in the range 14–16 codons and is randomly created during initial population generation. The number of genotypes created for the initial population is specified by the population size parameter.

6.3.3 Mapping

The grammar used to map *autoGE* genotypes to *autoGE* phenotypes is depicted in Figure **6.3**. The grammar specifies possible values of both categorical and numerical parameters required

by a GP classification algorithm. Genetic operator combinations and control flow options are also defined. Each genotype is mapped to a phenotype which represents a GP configuration. The mapping process uses equation **2.12** and follows the same procedure discussed in section **2.3.3.2** of **Chapter 2**. For example, given the following *autoGE* genotype with the following

```
<start>                  ::= <gp_parameters>
<gp_parameters>          ::= arith(<tree_gen >)|logical(< tree_gen >)|decision_tree(<tree_gen >)
<tree_gen >              ::= full(<params>)|grow(<params>)|ramped half-and-half(<params>)
<params>                 ::= <popSize> < itree_depth><maxOffspring_depth ><selection >
                             <reprod_rate><mut_type><maxMutation_depth><control_flow>
                             <operator_comb> <fitness_type ><generations >
<popSize>                ::= 100 | 200 | 300
<itree_depth>            ::= <ifDt >
<maxOffspring_depth >    ::= <ifDt >
<ifDt >                  ::= if(tree_type == dt) <dt_depth > else <depth_int >
<dt_depth >              ::= 2 | 3 | 4 | 5 | 6 | 7 | 8
<depth_init >            ::= 2 | 3 | ……..……|15
<selection >             ::= fitness proportionate | tournament selection(< tournament_size>)
<tournament_size>        ::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 10
<reprod_rate>            ::= 2 | 3 ,…….…….. |100
<mut_type >              ::= grow  | shrink
<maxMutation_depth>      ::= 2 | 3 | 4 | 5 | 6
<control_flow>           ::= fixed | random
<operator_comb >         ::= 0 | 1 | 2 | 3 | 4 | 5 | 6
<fitness_type>           ::= 0 | 1  |2 | 3 | 4
<generations>            ::= 50 | 100 | 200
```

Fig. 6.3 Grammar

codon sequence 10,215,30,48,7,8,220,30,40,73,5,11,21,112,32 the mapping proceeds as follows:

Starting with the start symbol there is only one production rule which maps to the non-terminal $<gp\_parameters>$, therefore, a codon value is not used. The first codon value 10 is used to translate the non-terminal $<gp\_parameters>$ to select one of three production rules using the mapping function this results in

$10 \ \% \ 3 = 1 \mapsto$ *logical*(<tree_gen>).

The second production rule is selected. This option defines the tree type design decision and specifies logical trees. The rule contains a non-terminal $<tree\_gen>$ which needs to be translated. The next codon 215 is used to translate the non-terminal and results in the following selection:

$215 \ \% \ 3 = 2 \mapsto$ *logical (ramped half-and-half(*<params>))

The selected production rule specifies the tree generation method to be used by GP. There is

also a non-terminal in the selected rule which requires mapping. However, there is only one rule for the non-terminal *<params>* and this is directly assigned without using a codon value.
logical (ramped half-and-half (<popSize>,<itree_depth>,< maxOffspring_depth>, <selection>, <sel_size>,<reprod_rates>, <mut_type>,<maxMutation_depth>, <reprod_seq>, <operator_pool>, <fitnes_type>, <generations>))
The next non-terminal to be mapped is *<popSize>* using the codon value 30 and this results in :
30 % 3 = 0 $\mapsto$ *logical (ramped half-and-half (100,<itree_depth>,< maxOffspring_depth>, <selection> ,<sel_size>, <reprod_rates>, <mut_type>, <maxMutation_depth>, <reprod_seq>, <operator_pool>, <fitnes_type>, <generations>))*
This specifies the population size and assigns the value 100. This process continues translating all non-terminals to terminals using the codon values. Figure 6.4 illustrates the complete mapping process. The left side of Figure 6.4 identifies the current step, the middle is the genotype to phenotype mapping at that step and the right side identifies the production rule evaluated from the codon value to translate the next non-terminal.

Steps	Prod rule
1. <gp_parameter>	10%3 = 1
2. logical<tree_gen>	215%3 = 2
3. logical(ramped half-and-half<params>)	none
4. logical(ramped half-and-hal(f<popSize><itree_depth><maxOffspring_Depth><selection> <reprod_rate><mut_type><maxMutation_depth><control_flow><operator_comb> <fitness_type><generations>))	30%3 = 0
5. logical(ramped half-and-half(100,<itree_depth><maxOffspring_Depth><selection> <reprod_rate><mut_type><maxMutation_depth><control_flow><operator_comb> <fitness_type><generations>))	48%14 = 6
6. logical(ramped half-and-half(100,8,<maxOffspring_Depth><selection><reprod_rate> <mut_type><maxMutation_depth><control_flow><operator_comb><fitness_type> <generations>))	7%14 = 0
7. logical(ramped half-and half(100,8,2,<selection><reprod_rate><mut_type><maxMutation_depth> <control_flow><operator_comb><fitness_type><generations>))	8%2 = 0
8. logical(ramped half-and-half(100,8,2,fitnessproportionate,<reprod_rate><mut_type> <maxMutation_depth> <control_flow><operator_comb><fitness_type><generations>))	220%99 = 22
9. logical(ramped half-and half(100,8,2,fitnessproportionate,24,<mut_type><maxMutation_depth> <control_flow><operator_comb><fitness_type><generations>))	30%0 = 0
10. logical (ramped half-and half(100,8,2,fitnessproportionate,24,grow,<maxMutation_depth> <control_flow><operator_comb><fitness_type><generations>))	40%5 = 0
11. logical(ramped half-and-half(100,8,2,fitnessproportionate,24,grow,2,random,<operator_comb> <fitness_type><generations>))	5%7 = 0
12. logical(ramped half-and-half(100,8,2,fitnessproportionate,24,grow,2,random,0,<fitness_type> <generations>))	11%5 = 1
13. logical(ramped half-and-half(100,8,2,fitnessproportionate,24,grow,2,random,0,1,<generations>))	21%3 = 0
logical(ramped half-and half(100,8,2,fitnessproportionate,24,grow,2,random,0,1,50))	

Fig. 6.4 Genotype-phenotype mapping

The resulting mapping (phenotype) is given as follows
logical (ramped half-and-half (100,8,2,fitnessProportionate,24 ,grow, 2,random,0,1,50))
This *autoGE* phenotype is then translated into a GP configuration as listed in Table 6.1.

Parameter#	Value
tree type	logical
population size	100
tree generation method	ramped half and half
initial tree depth	8
max offspring depth	2
selection method	fitness proportionate
crossover rate	24 (mutation = 100-crossover = 76)
mutation type	grow
mutation depth	2
control flow	random
operator combination	crossover-mutation
fitness function	$f_{measure}$
number of generations	50

Table 6.1 AutoGE evolved GP parameter settings

6.3.4 Fitness Function and Selection

The fitness of each genotype is evaluated by using the mapped phenotype to configure a GP classification algorithm and then applying the GP classification algorithm to a classification problem at hand. Training and testing are carried out. During training, thirty independent runs of the GP classification algorithm are performed using a training set. The best evolved classifier from the training is then applied to the test set resulting in a test accuracy value. This value is assigned as the fitness of the genotype. Tournament selection is used to select parents to evolve the next generation as described in section **2.3.1.3** of **Chapter 2**.

6.3.5 Crossover

Single point crossover is used as described in section **2.3.3.4** of **Chapter 2**.

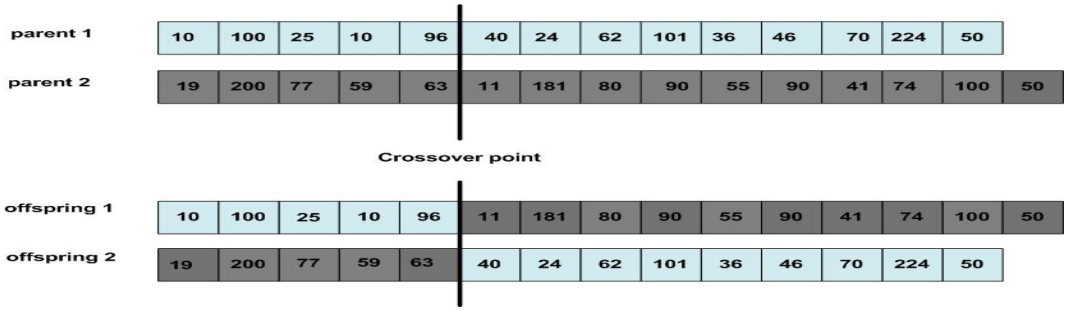


Fig. 6.5 AutoGE single point crossover

As stated tournament selection is used to select two genotypes from a current population to act as parents. Figure **6.5** is an illustration of the single point crossover implemented. Although in Figure **6.5** the codons are presented in their integer values single point crossover can be applied to either binary strings or integer values. In this thesis, binary strings are used. If crossover does not take place as determined by the crossover probability rate the two selected parents are added to the next population.

6.3.6 Mutation

Bit mutation described in section **2.3.1.5** of **Chapter 2** is used as the mutation operator for *autoGE*. Tournament selection is used to select a parent for mutation. After undergoing mutation the genotype is added to the new population.

6.3.7 Elitism

In order to preserve fitter genotypes elitism is used. A percentage of the fittest genotypes from the current generation is copied to the next generation.

6.3.8 Termination

The algorithm proceeds from generation to generation until a preset number of generations have been completed or until the desired fitness is met then the algorithm terminates. The genotype with the best fitness is returned. This is the genotype with the best GP classification configuration for the problem considered.

6.3.9 *AutoGE* Parameter Settings

Parameter tuning was carried out using trial runs to establish optimal(near) parameter values for the *autoGE* approach. The *autoGE* algorithm is configured with a population size of 20. The size of chromosomes in the initial population is randomly selected in the range of 14 to 16 with a codon size of 8 bits. A crossover probability rate of 85% and a bit mutation probability rate of 5% were found to be suitable. A tournament selection size of 4 was established. An elitism rate of 10% was found to be adequate while 30 generations were found to be suitable for algorithms convergence Table **6.2** summarises the parameter settings for the autoGE approach.

parameter	value
Population size	20
Selection method	tournament (size 4)
Single point crossover rate	85%
Bit mutation rate	5%
Elitism	10%
fitness function	accuracy
Individual size	14-16
Wrapping	yes
Maximum generations	30

Table 6.2 AutoGE settings

The *autoGA* approach uses the same function and terminal sets as the manual design configuration presented in section **4.2.1** of **Chapter 4**

6.4 Fitness Landscape Analysis Settings

The fitness landscapes of the design spaces searched by the automated design approaches are evaluated using the autocorrelation analysis method described in section **2.6.2.1** of **Chapter 2**. For each approach, a random walk of 30 steps (i.e. T = 30) is generated and the best fitness is recorded. Each step consists of 30 generations of the considered approach. As in [143] mutation is used as the neighbourhood operator. From the obtained results the autocorrelation function will be calculated using equation **2.15**. The autocorrelation length is evaluated from the autocorrelation function using equation **2.16**. The problem instances used for fitness landscape analysis are presented in section **3.5.3** of **Chapter 3**. As stated previously it is not possible to carry out a fitness landscape analysis comparison of the automated approaches to manual design since manual design does not search a structured a design space. The results of the comparison of the fitness landscape analysis of the *autoGA* and *autoGE* approaches are presented in section **7.6** of **Chapter 7**.

6.5 Summary

In this chapter, an outline of the grammatical evolution approach for automating the design of GP classification algorithms was presented. The design decisions considered by the GE approach are also presented. The chapter also outlined a listing of the GE grammar for the

automated design approach. The setting used for the *autoGE* approach and for performing the fitness landscape analysis were also presented. The results of evaluating the effectiveness of the *autoGE* approach are presented in the next chapter.

Chapter 7

Results and Discussion

7.1 Introduction

In this chapter, the results of applying the *autoGA* and *autoGE* approaches to the classification problem instances described in **Chapter 3** are presented. Furthermore, the results are compared to those obtained from applying the *manual design* approach outlined in **Chapter 4** to the same problem instances. As previously stated the *manual design* approach involves conducting three experiments differentiated by the classifier type used. For the convenience of presenting the results, these are termed as *arithmetic, logical* and *decision tree*. The chapter is structured as follows: section **7.2** presents the results of applying the three approaches, namely *autoGA*, *autoGE* and *manual design* on multiple datasets from multiple problem domains. This is followed by sections **7.3** and **7.4** which present the results of applying the three approaches to specific problem domains, namely cyber-security and financial forecasting respectively. Section **7.5** presents the design times achieved across all experiments and finally, section **7.6** presents the results of an evaluation of the fitness landscape analysis.

7.2 Multiple Domain Problems

In this section the results of applying the three approaches on binary and multiclass classification problems obtained from multiple domains are presented and compared. Section **7.2.1** presents the binary classification results while the multiclass classification results are presented in section **7.2.2**.

7.2.1 Binary Classification Results

7.2.1.1 Training

Table **7.1** presents the training results. Each row represents a dataset while each column is the applied approach. The results are the best training fitness ± standard deviation (at the 95% percentile confidence interval) over thirty independent runs for each approach. From the

dataset	arithmetic	logical	decision tree	autoGA	autoGE
aus credit	0.89±0.01	**0.91±0.01**	0.86±0.01	0.89±0.01	**0.91±0.01**
appendicitis	**0.97±0.02**	0.89±0.02	0.89±0.02	0.95±0.02	0.95±0.02
breast cancer	0.98±0.01	0.97±0.01	0.94±0.02	0.98±0.01	**0.99±0.01**
cylinder band	0.74±0.01	0.77±0.01	0.64±0.01	0.75±0.01	**0.80±0.04**
diabetes (pima)	**0.78±0.07**	**0.78±0.07**	0.69±0.07	0.75±0.01	0.74±0.04
german credit	0.76±0.06	0.76±0.06	0.73±0.06	0.85±0.07	**0.86±0.07**
heart disease	0.92±0.01	0.94±0.01	0.79±0.01	0.87±0.01	**0.95±0.01**
hepatitis	**0.98±0.03**	**0.98±0.03**	0.88±0.02	0.93±0.02	**0.98±0.03**
liver disease	**0.80±0.01**	0.73±0.01	0.62±0.01	**0.80±0.01**	0.76±0.01
mushroom	0.86±0.00	0.86±0.00	0.60±0.00	**0.88±0.00**	**0.88±0.00**
tictactoe	0.87±0.07	0.84±0.07	0.72±0.07	0.94±0.07	**0.99±0.00**
averages	0.87±0.03	0.86±0.02	0.76±0.03	0.87±0.02	**0.89±0.02**

Table 7.1 Training accuracy for binary classification problems

obtained results, the *autoGE* approach trained better than the other classification approaches on 5 of the 11 datasets and tied on 3 datasets. The *autoGA* approach tied on 2 datasets, while the manually designed *arithmetic* tree classifiers trained well on 1 dataset and tied on 3 datasets. *Logical* tree classifiers tied on the 3 datasets. On average across all datasets, the *autoGE* approach had the best training results of 89% followed by the *autoGA* and *arithmetic* tree classifiers which tied at 87%. The *logical* and *decision* tree classifiers averaged 86% and 76% respectively.

7.2.1.2 Testing

Table **7.2** presents the test accuracy results of the best training individual ± standard deviation. From the test results both the *autoGA* and *autoGE* approaches tested better on 4 datasets each and tied on 1 while *logical* tree classifiers tested well on 1 dataset. On average across all datasets, the *autoGE* approach tests better than all the other approaches. This was the expected result as the training results showed the *autoGE* approach trained better than the other approaches, therefore, it was anticipated that it should test better as well. The test average across all datasets of the *autoGE* approach was found to be 81% while the *autoGA*

approach averaged 79%. The manually designed *arithmetic* and *logical* tree classifiers both had an equivalent test accuracy average of 74% and the *decision tree* classifiers averaged 69%. The statistical significance of the testing accuracy results was evaluated using the non-

dataset	arithmetic	logical	decision tree	autoGA	autoGE
aus credit	0.83±0.01	0.84±0.01	0.85±0.01	**0.88±0.01**	0.86±0.01
appendicitis	0.84±0.03	0.78±0.03	0.85±0.03	0.91±0.03	**0.94±0.03**
breast cancer	0.97± 0.02	0.93±0.03	0.90±0.04	0.97±0.02	**0.98±0.02**
cylinder band	0.66±0.01	0.68±0.01	0.69±0.01	**0.75±0.01**	0.74±0.01
diabetes (pima)	0.64±0.01	**0.75±0.01**	0.69±0.01	0.70±0.07	0.60±0.01
german credit	0.65±0.01	0.65±0.01	0.65±0.01	**0.68±0.01**	0.66±0.05
heart disease	0.77±0.02	0.64±0.02	0.44±0.01	0.72±0.02	**0.81±0.08**
hepatitis	0.67±0.03	0.75±0.03	0.75±0.03	0.75±0.03	**0.88±0.02**
liver disease	0.64±0.01	0.64±0.01	0.44±0.01	**0.71±0.01**	0.65±0.01
mushroom	0.78±0.00	0.75±0.00	0.66±0.00	**0.81±0.00**	**0.81±0.00**
tictactoe	0.73±0.01	0.76±0.01	0.65±0.01	0.86±0.01	**0.98±0.01**
averages	0.74±0.01	0.74±0.02	0.69±0.02	0.79±0.02	**0.81±0.02**

Table 7.2 Testing accuracy for binary classification problems

parametric Friedman test with a post-hoc Bonferroni–Dunn test for pairwise comparison as recommended by [51] for comparing multiple classification approaches on multiple datasets. The details of this approach to evaluating statistical significance were presented in section **2.2.2** of **Chapter2**. The testing results were ranked with the best performing approach assigned a rank of 1 and the least a rank of 5. If approaches tied the affected positions were averaged amongst the approaches involved in the tie. Following the procedure of the Friedman test, the average ranks for each approach were calculated and these are presented in Table **7.3**. From the table of ranks, the *autoGE* approach was ranked 1^{st} with an average

algorithm	arithmetic	logical	decision	autoGA	autoGE
average rank	3.818	3.590	4	1.818	1.772
position	4	3	5	2	1

Table 7.3 Average ranks for binary classification problems

rank of 1.772 followed by the *autoGA* approach with an average rank of 1.818. The best performing manual design approach with an average rank of 3.818 was the *logical* tree classifiers which were ranked 3rd. *Arithmetic* and *decision tree* classifiers were ranked 4th and 5th respectively. Using the values from Table **7.3** and Equation **2.6** the calculation of the Friedman's χ_F^2 is given by:

$$\chi_F^2 = \frac{12*11}{5*6}\left[(3.818^2 + 3.590^2 + 4^2 + 1.818^2 + 1.772^2) - \frac{5*6^2}{4}\right] = 21.756 \quad (7.1)$$

and from Equation **2.7** Iman's F_f statistic is given by:

$$F_f = \frac{(11-1)*21.756}{11*(5-1) - 21.756} = 9.78 \quad (7.2)$$

With 5 approaches and 11 data sets, F_F is distributed according to the F distribution with 5 -1 = 4 and (5-1)*(11-1) = 40 degrees of freedom. The critical value of $F_{(4,40)}$ for α = 0.05 is 2.608 and since $F_F > F_{0.05}(4,40)$(9.78 >2.608) the null hypothesis which states that all the classification approaches perform equivalently was rejected. Using the two-tailed Bonferroni-Dunn test a pairwise comparison is carried out between the best performing manual design approach, in this case, the *logical* tree classifiers and the automated designed algorithms. The critical value $q_{0.05}$ for 5 classification methods is 2.498, therefore, from Equation **2.8** the critical difference CD is evaluated to:

$$CD = 2.498 * \sqrt{\frac{5*6}{11*6}} = 1.68 \quad (7.3)$$

The difference between the average rank of the *logical* tree classifiers and *autoGE* is 1.82 and the difference between the *logical* tree classifier average and *autoGA* is 1.77. Both these values are greater than the CD (1.68) value which suggests that the performance of the automated designed classification approaches are significantly better than the manually designed approaches for the considered datasets. However, the difference in performance between *autoGA* and *autoGE* is found to be not significant, therefore, their performance is considered to be equivalent.

approach	difference	significance
logical - *autoGE*	1.82(>1.68)	yes
logical - *autoGA*	1.77 (>1.68)	yes
autoGE - *autoGA*	0.05(<1.68)	no

Table 7.4 Statistical significance summary: binary classification problems

Table **7.14** presents a summary of the statistical significance analysis of the performances of the automated design approaches and the best performing manual design approach.

From the obtained results the *autoGE* approach consistently trains and tests better than the other approaches. Although the *autoGE* approach achieves better training and test averages the differences in performance between the *autoGE* approach and *autoGA* approach across all

datasets is not statistical significant. Both automated design approaches perform significantly better than manual design. Either approach of automated design is found to be suitable to solve the binary classification problems considered.

7.2.1.3 Configurations

Table **7.5** outlines the best configurations for the binary datasets evolved by the automated design approach for each dataset. The first column represents the parameters in the configuration and the subsequent columns are the parameter values for each dataset indexed as follows: i-*australian credit*, ii-*appendicitis*, iii-*breast cancer*, iv-*cylinder band*, v-*german credit*, vi-*heart*, vii-*hepatitis*, viii-*liver disease*, ix-*mushroom*, x-*mushroom*(tie) and xi-*tictactoe*. The last column xii is an average of the manually tuned parameters.

parameter	dataset										
	i	ii	iii	iv	v	vi	vii	viii	vix	x	xii
tree type	0	0	0	2	1	0	1	0	0	1	-
pop size	200	200	100	300	200	200	300	200	200	200	**300**
tree gen method	2	0	0	2	0	0	0	0	2	0	**2**
init tree depth	8	8	7	3	6	8	4	3	8	5	**5**
max offsp depth	10	9	6	5	5	8	8	9	11	3	**9**
selection method	0	0	1	0	0	1	1	0	0	0	**1**
selection size	-	-	6	-	-	6	8	-	-	-	**6**
crossover rate	21	80	89	31	33	6	56	77	60	46	**70**
mutation type	1	0	1	1	1	0	1	1	0	0	**0**
mutation depth	5	2	5	3	5	3	2	3	4	3	**4**
control flow	0	0	1	1	0	1	1	0	1	0	**0**
operator comb	3	0	2	2	2	1	1	5	5	1	**0**
fitness function	0	0	3	0	0	1	1	3	3	3	**0**
number of gens	55	200	200	161	109	200	100	170	138	200	**300**

Table 7.5 Binary class auto-designed configurations

The *arithmetic* tree classifiers were included by the automated design approaches in 6 of the 10 configurations while *logical* tree classifiers were used in 3 configurations and *decision tree* classifiers were included once. The expectation was for *logical* tree classifiers to constitute the majority of the configurations since they ranked first among the manual design approach. This seems to indicate that automated design was able to perform a wider search than human design resulting in better configurations which use *arithmetic* tree classifiers for the considered datasets. A *population size* of 200 was used on 7 configurations, 300 on 2 configurations and 100 on 1 configuration. Experienced human evolutionary algorithm

designers prefer the *ramped half-and-half* method for initial population generation, however, from the automatically generated configurations this method was only used 3 times with the *full* method included more frequently and was used 7 times while the *grow* method was not used. *Initial tree depth* was set to a value in the range of [5-8] in 7 configurations while 3 configurations had values less than 5. These values correlated with the manual design values which had an average of 5 for this parameter. *Maximum offspring depth* was set to values in the range [5-11] in all configurations with the exception of 1 configuration in which the value was set to 3. *Tournament selection* was used as the selection method 3 times with *fitness proportionate selection* used 7 times. On the 3 occasions, that *tournament selection* was used, the *tournament size* was set to a value of 6 twice and 8 once. Five configurations used fixed genetic operator application rates. Of those 6 only 1 configuration for appendicitis used the initially set application rate of 80% *crossover* and 20% *mutation*. The other 5 of the 6 were configured as follows; 2 used 100% *crossover*, 1 used 100% *mutation*, 1 used the preset rates and 1 used 100% *crossover* and then random mutation. *Shrink mutation* was selected in 6 configurations while 4 configurations used *grow mutation*. This is contrary to the manual design approach which used *grow mutation*. *Maximum mutation depth* values were set in the range [2-6] across all the configurations with the value of 3 being set more frequently. The manual average was 4. Three fitness functions of the possible 5 were used, namely *accuracy*, *f-measure* and $weighted_{rand}$. *Accuracy* is used in 4 configurations, $weighted_{rand}$ in 4 configurations and *f-measure* in 2 configurations. Maximum *generations* were configured within the range of [100 - 200] generations except for one configuration which used 55 generations.

Automated design seems to work well due to its unconstrained ability to be able to combine and consider what can be considered as unconventional configurations. Humans are constrained by logic and chronology for example in considering values for the *crossover rate* the values were considered in steps of 5's, yet automated design was able to establish 31% to be a very good value. The manual parameter tuning carried out for this part of the study followed a similar approach performed by manual design i.e. parameters were tuned for each dataset. However, despite this approach the automated design significantly out performed manual design. Significantly manual design presented a wider search space for GP classifiers (300) and a higher number of generations 300 than the automated design. Yet automated design was able to evolve configurations that yield GP classifiers that achieve better performances from a narrower search space (200 on average) and achieved convergence in less generations.

The differences in evolved configurations also lend weight to the widely held notion that when it comes to classification the data also influences the parameter settings. Therefore no set of parameters can be optimal across different data instances.

7.2.2 Multiclass Classification Results

7.2.2.1 Training

The training results for the classification of multiclass problems are presented in Table **7.6**. Similarly to Table **7.1** and Table **7.2** the rows represent datasets and the columns the classification approach. The results are the best training fitness ± standard deviation (at the 95% percentile confidence interval) over thirty independent runs for each approach. From

dataset	arithmetic	logical	decision tree	autoGA	autoGE
balance	0.76 ±0.02	0.84±0.03	0.69±0.03	**0.99±0.03**	0.92±0.02
post-operative	0.81±0.04	0.29±0.04	0.76±0.04	0.80±0.04	**0.86±0.04**
car	**0.83±0.02**	0.23±0.02	**0.83±0.02**	**0.83±0.02**	**0.83±0.02**
lymphography	0.85±0.06	0.84±0.06	0.79±0.05	**0.92±0.06**	0.86±0.06
cleveland	0.62±0.06	0.21±0.05	0.61±0.07	**0.67±0.06**	0.60±0.06
page-blocks	0.95±0.04	0.96±0.04	0.51±0.04	**0.97±0.04**	**0.97±0.04**
demartology	0.77±0.05	0.35±0.06	0.67±0.06	0.75±0.08	**0.88±0.06**
flare	0.71±0.03	0.38±0.03	0.75±0.03	**0.76±0.03**	0.75±0.03
glass	0.63±0.07	0.49 ±0.07	0.57±0.07	0.59±0.07	**0.65±0.07**
zoo	**0.87±0.07**	0.62±0.07	0.84±0.07	0.86±0.07	**0.87±0.07**
ecoli	0.84±0.05	0.77±0.05	0.71±0.05	**0.88±0.05**	0.64 ±0.06
averages	0.79±0.05	0.54±0.05	0.70 ±0.05	**0.82±0.05**	0.80 ±0.05

Table 7.6 Training multiclass classification problems

the results the *autoGA* approach trained well on 5 of the 11 datasets and tied on 2 datasets, while the *autoGE* approach trained well on 3 datasets and tied on 3 datasets. The manually designed *arithmetic* tree classifiers tied on 2 datasets and the *decision* tree approach tied on 1 dataset. On average across all datasets the *autoGA* approach trained better than the other approaches achieving a training average fitness of 82%. The *autoGE* approach was the next best training approach. *AutoGE* achieved an average fitness of 80% followed by *arithmetic* tree classifiers with an average of 79% and decision trees with an average of 70%. The *logical* tree classifiers had an average training fitness of 54%.

7.2.2.2 Testing

Table **7.7** presents the test accuracy results for the multiclass classification problems. The

dataset	arithmetic	logical	decision tree	autoGA	autoGE
balance	0.81±0.03	0.76±0.03	0.68±0.06	**0.98±0.01**	0.92±0.03
post-operative	0.61±0.09	0.25±0.09	0.71±0.09	**0.75±0.09**	0.64±0.09
car	0.40±0.03	0.18±0.03	0.64±0.04	**0.66±0.04**	0.46±0.04
lymphography	0.73±0.07	0.76±0.07	0.78±0.07	**0.82±0.07**	0.78±0.07
cleveland	0.53±0.09	0.17±0.07	0.48±0.07	**0.57±0.07**	0.55±0.07
page-blocks	0.55±0.03	0.57±0.03	0.38±0.03	**0.60±0.03**	0.59±0.03
demartology	0.67±0.08	0.38±0.06	0.57±0.08	0.69±0.08	**0.78±0.08**
flare	0.68±0.05	0.43±0.05	0.67±0.05	0.67±0.04	**0.71±0.04**
glass	0.24±0.09	0.19±0.09	0.45±0.09	**0.53±0.09**	0.24±0.09
zoo	**0.81±0.09**	0.56±0.09	0.72±0.09	**0.81±0.09**	**0.81±0.09**
ecoli	0.61±0.08	0.40±0.08	0.31±0.08	**0.90±0.05**	0.43±0.09
averages	0.60±0.07	0.42 ±0.06	0.58±0.06	**0.73±0.06**	0.63±0.06

Table 7.7 Testing multiclass classification problems

testing accuracy results show that the *autoGA* approach tested well on 8 of the 11 datasets and tied on 1 dataset. The *autoGE* approach tested well on 2 datasets and tied on 1 while the *arithmetic* tree classifiers tied on 1 dataset. The *autoGA* approach achieved an average testing accuracy of 73% across all datasets followed by the *autoGE* approach with an average of 63%. The manually designed *arithmetic* tree classifiers achieved an average of 60% while *decision tree* and *logical* tree classifiers achieved an average of 58% and 42% respectively. To evaluate the statistical significance of the testing results the Friedman test was used. The average ranks for each approach were evaluated and the results are presented in Table **7.8**. From the table of average ranks, the *autoGA* approach was ranked 1^{st} followed by the *autoGE*.

approach	arithmetic	logical	decision	autoGA	autoGE
average rank	3.364	4.409	3.545	1.409	2.273
position	3	5	4	1	2

Table 7.8 Average ranks for multiclass classification problems

The *arithmetic* tree classifiers, were ranked 3rd followed by the *decision tree* classifiers and finally, the *logical* tree classifiers. Using the average ranks the Friedman χ_F^2 was calculated as follows:

$$\chi_F^2 = \frac{12*11}{5*6}\left[(3.364^2+4.409^2+3.545^2+1.409^2+2.273^2)-\frac{5*6^2}{4}\right] = 24.10 \quad (7.4)$$

and from Equation **2.7** Iman's F statistic was evaluated to:

$$F_f = \frac{(11-1)*24.10}{11*(5-1)-24.10} = 12.11 \tag{7.5}$$

The critical value of $F_{(4,40)}$ for α = 0.05 was established to be 2.608 and since F_F >$F_{0.05}$(4,40) (12.11 >2.608) the null hypothesis which states that the classification approaches for multiclass classification problems considered in this study perform equivalently was rejected. Using the Bonferroni-Dunn post-hoc test the critical difference for $F_{(4,40)}$ was shown to be 1.68 in section **7.2.1.2**. The difference between the average rank of *autoGA* and the average rank of the best performing manually designed classifier (*arithmetic*) was found to be 1.955. As this value was greater than the critical difference (1.68) this indicated that the differences in performance of the *autoGA* approach and the best performing manually designed approach *arithmetic* tree classifiers were statistically significant. The difference in average rank between *autoGE* and *arithmetic* tree classifiers was calculated to be 1.09. This value was less than the critical difference, therefore, there was no significant difference in performance between the 2 approaches considered. The differences between the automated

approach	difference	significance
arithmetic - *autoGA*	1.96(>1.68)	yes
arithmetic - *autoGE*	1.09 (<1.68)	no
autoGE - *autoGA*	0.86(<1.68)	no

Table 7.9 Statistical significance summary: multiclass classification problems

designed approaches was also found not to be significant although *autoGA* was found to be able to evolve classifiers which achieved higher accuracies than *autoGE* on average across all datasets. Table **7.9** is an illustration of the statistical analysis results.

The *autoGA* approach consistently trained and tested better than the other approaches on this set of classification problems. *AutoGA* performed significantly better than the manual design approach but not significantly better than the *autoGE*. Based on the significance of the differences in performance *autoGA* was found to be most suitable to use in multiclass classification.

7.2.2.3 Configurations

Table **7.10** presents the configurations used by the best testing automated design algorithms. The datasets are indexed as follows: i-*balance*, ii-*post operation*, iii-*car*, iv-*lymphography*, v-*cleveland*, vi-*page blocks*, vii-*dermatology*, viii-*flare*, ix-*glass*, x-*ecoli* and xi-*manual averages*. From the 10 configurations, 8 were configured to use the arithmetic tree type and 2

use the logical tree type. This was in alignment with the accuracy results as the *arithmetic* tree classifier performed the best among the manual designs. A *population size* of 300 was used in 4 of the 10 configurations and values of 200 and 100 were each used 3 times. The *full* tree generation method was used 7 times while the preferred method by manual algorithm designers the *ramped half-and-half* method was used only twice and the *grow* method was used once. The *initial tree depth* parameter values were set in the range [4-10] yet the possible values range from [2-15]. The *maximum offspring depth* values were set in the range [6-12] yet the possible values range from [2-15]. *Tournament selection* was used in 8 configurations and *fitness proportionate selection* in 2 configurations. *Tournament selection* size values were set in the range of [2-9].

parameter	dataset										
	i	ii	iii	iv	v	vi	vii	viii	ix	x	xi
tree type	0	0	0	1	0	1	0	0	0	0	-
pop size	300	100	100	300	300	200	200	200	300	100	**300**
tree gen method	1	0	0	0	0	0	0	0	2	2	**2**
init tree depth	6	10	9	8	4	8	5	6	4	5	**3**
max offsp depth	9	8	10	12	10	12	6	6	6	10	**8**
selection method	1	0	1	1	1	1	1	1	0	1	**1**
selection size	9	-	2	8	8	7	2	4	-	7	**9**
crossover rate	82	80	27	18	36	78	81	51	77	69	**80**
mutation type	0	0	0	1	1	1	1	1	0	1	**0**
mutation depth	6	6	5	2	6	6	4	4	2	3	**6**
control flow	1	0	0	0	1	0	0	0	1	1	**0**
operator comb	3	3	1	1	3	1	2	3	3	0	**0**
fitness function	0	0	0	0	0	0	0	0	0	0	**0**
number of gens	200	200	50	200	50	200	100	200	100	50	**300**

Table 7.10 Multiclass auto-designed configurations

The *crossover* rate was set at a higher rate than the *mutation* rate in 6 configurations, while the *mutation* rate was set higher than *crossover* in 3 configurations. In 1 configuration *crossover* and *mutation* were set to rates of 51% and 49% respectively. As stated the manually designed approach sets the *crossover* rate to a higher value than the mutation rate. *Grow* mutation was used in 4 configurations while *shrink* mutation was used in the other 6 configurations. The *maximum mutation depth* values were set to values in the range [2-6]. Five configurations used random preset rates while 3 configurations used 100% *crossover* and 1 configuration used 100% *mutation*. A *maximum generation* value of 200 was used 5 times, 100 used twice and a value of 50 was used 3 times.

Similarly to binary classification configurations there is no discernible pattern regarding the configurations, as these are evolved to suite the presented problem instance.

7.3 Cybersecurity

In this section the results of applying the 5 classification approaches to problems from a specific domain namely the cyber-security domain are presented. The cybersecurity problem instances were obtained from the NSL-kDD dataset discussed in section **3.5.2** of **Chapter 3**.

7.3.1 Training

Table **7.11** presents the training results. The rows represent datasets and the columns the classification approach. The results are the best training fitness ± standard deviation (at the 95% percentile confidence interval) over thirty independent runs for each approach. From the training results obtained the *autoGA* approach trained well on 2 datasets and tied on 2 while the *autoGE* trained well on 1 and tied on 2 datasets. The manually designed *arithmetic* tree

dataset	arithmetic	logical	decision tree	autoGA	autoGE
normal	0.97±0.03	0.96±0.03	0.92±0.03	**0.98±0.02**	**0.98±0.02**
dos	**0.99±0.01**	0.97±0.02	0.92±0.04	**0.99±0.01**	**0.99±0.01**
probe	**0.99±0.01**	0.98±0.02	0.91±0.04	0.98±0.02	0.98±0.02
u2r	0.99±0.01	0.99±0.01	0.99±0.01	**1.00±0.00**	0.99±0.01
r2l	0.98±0.02	0.98±0.02	0.98±0.02	0.98±0.02	**0.99±0.01**
multi	0.81±0.02	0.68±0.02	0.59 ±0.02	**0.82±0.02**	0.72 ±0.02
averages	**0.96±0.02**	0.93±0.02	0.89±0.03	**0.96±0.01**	0.94±0.02

Table 7.11 Cyber security training results

classifier trained well on 1 dataset and tied on 1. On average across all datasets the *autoGA* approach and the *arithmetic* tree classifier approach trained better than the other 3 approaches and achieved an average fitness of 96%. This is followed by the *autoGE* approach with a training average of 94%. The *logical* and *decision* tree classifiers achieve an average of 93% and 89% respectively.

7.3.2 Testing

Table **7.12** presents the cyber security testing results. The *autoGE* approach tested well on 3 datasets while the *autoGA* approach tested well on 2 datasets and the *arithmetic* tree classifier tested well on 1 dataset. On average across all datasets the *autoGA* approach achieved the

best testing average accuracy of 96% while the *autoGE* and *logical* tree classifiers achieved the same average of 94% . The *arithmetic* and *decision tree* classifiers achieved an average of 93% and 90% respectively. To evaluate the statistical significance of the differences in

dataset	arithmetic	logical	decision tree	autoGA	autoGE
normal	0.97±0.03	0.96±0.04	0.92±0.06	0.96±0.04	**098±0.02**
dos	0.90±0.02	0.96±0.03	0.93±0.02	0.98±0.01	**0.99±0.01**
probe	**0.99±0.01**	0.95±0.04	0.91±0.06	0.98±0.03	0.98±0.03
u2r	0.99±0.01	0.99.±0.01	0.98±0.01	**1.00±0.00**	0.99±0.01
r2l	0.98±0.02	0.98±0.01	0.98±0.02	0.98±0.02	**0.99±0.01**
multi	0.75±0.02	0.80 ±0.03	0.66±0.03	**0.81±0.02**	0.70±0.03
averages	0.93±0.02	0.94±0.03	0.90±0.03	**0.96±0.02**	0.94±0.02

Table 7.12 Cybersecurity test results

performance the Friedman test was used. The average ranks were evaluated and are presented in Table **7.13**. From the average rankings the *autoGE* approach ranks 1^{st} followed by the *autoGA* approach and the *arithmetic* tree classifier. The *logical* and *decision tree* classifiers rank 4th and 5th respectively.

	arithmetic	logical	decision	autoGA	autoGE
average rank	2.92	3.167	4.583	2.25	2.083
position	3	4	5	2	1

Table 7.13 Average ranks cybersecurity

Using the average ranks and Equation 2.6 the Friedman χ_F^2 is calculated to be :

$$\chi_F^2 = \frac{12*6}{5*6}\left[(2.92^2+3.167^2+4.583^2+2.25^2+2.083^2)-\frac{5*6^2}{4}\right] = 9.50 \tag{7.6}$$

and from Equation 2.7 Iman's F statistic was evaluated to:

$$F_f = \frac{(6-1)*9.50}{6*(5-1)-9.5} = 3.28 \tag{7.7}$$

The critical value $F_{(4,20)}$ for $\alpha = 0.05$ is given as 2.87 and since $3.28 > 2.87$ the null hypothesis which states that the 5 approaches perform equivalently was rejected and a post-hoc test using the Bonferroni–Dunn test for pairwise comparison was carried out. The critical difference CD at the 95% level was evaluated to:

$$CD = 2.498*\sqrt{\frac{5*6}{6*6}} = 2.28 \tag{7.8}$$

The differences of the average ranks between the auto designed approaches and the best performing manual approach, *arithmetic* tree classifiers, was found to be not greater than the critical difference, therefore, the differences in performance between the automated designed approaches and the best manual approach were not statistically significant. Table **7.14** presents the results of the statistical analysis.

approach	difference	significance
arithmetic - *autoGE*	0.84(<2.28)	no
arithmetic - *autoGA*	0.67(<2.28)	no
autoGE - *autoGA*	0.17(<2.28)	no

Table 7.14 Significance for cyber security problems

On the cybersecurity domain the best performing manual design approach performed equivalently to the automated design approaches. By the ranking metric the *autoGE* performed better while through the training and test accuracy the *autoGA* performed better. However no one approach was significantly better than the others. However, it is important to note that if the multiclass dataset i.e. *multi* is considered in isolation as the other sets are binary problems then the *autoGA* performed significantly better on that dataset. This is consistent with the results obtained from the binary and multiclass problems from sections 7.2.1 and 7.2.1. Similarly if binary cybersecurity datasets are considered on their own the *autoGE* approach performs better than the other approaches also inline with the previous results. This further enhances the recommendation of the *autoGE* approach for binary class problems and the *autoGA* approach for multiclass problems.

7.3.3 Configurations

Table **7.15** presents the best performing GP configurations evolved by the automated design approaches. *Arithmetic* tree type were used in 4 of the 5 configurations and the *logical* tree type was used once for the *u2r* dataset. A *population size* parameter value of 200 was set in 3 configurations and a value of 100 was used twice. The *grow* tree generation method was used in 3 configurations while the *ramped half-and-half* and the *full* methods were used once each. The *initial tree depth* was set to values in the range [4-8] with values of 4 and 5 being used twice each and 8 once. The manual design average for the *initial tree depth* was 3. *Maximum offspring depth* was set in the range [6-10] in 4 of the 5 configurations and 3 in one configuration. The *fitness proportionate selection* method was configured once while *tournament selection* was used in 4 configurations with a *selection size* set in the range [3-5]. *Crossover* rates were set to higher values than *mutation* rates in 3 configurations. *Grow*

mutation was used 3 times while *shrink mutation* was used twice with the *mutation depth* set in the range [2-6]. *Control flow* was set to random twice. Three configurations had fixed application rates with the *normal* dataset configured with a *crossover* rate of 24% and a *mutation* rate of 76%. The *r2l* configuration used a *crossover* rate of 52% and a *mutation* rate of 48%. The best configuration applied to the *multiclass* dataset had a *crossover* rate of 82% and a *mutation* rate of 18%. On all configurations, *accuracy* was configured as the fitness function. Three configurations were set to terminate after 100 *generations* and 2 after 50 generations.

parameter	parameter values					
	normal	dos	u2r	r2l	multi	avg manual
tree type	0	0	1	0	0	-
pop size	100	100	200	200	200	**300**
tree gen method	1	2	0	1	1	**2**
init tree depth	5	4	8	5	4	**3**
max offsp depth	6	3	9	10	8	**9**
selection method	1	1	0	1	1	**1**
selection size	4	3	-	4	5	**9**
crossover rate	24	25	58	52	82	**80**
mutation type	0	0	0	1	1	**0**
mutation depth	4	2	6	2	6	**8**
control flow	0	1	1	0	0	**0**
operator comb	0	1	1	1	0	**0**
fitness function	0	0	0	0	0	**0**
number of gens	100	100	100	50	50	**200**

Table 7.15 NSL-KDD auto-designed configurations

7.4 Financial Forecasting

In this section, the results of applying the 5 classification methods to financial forecasting problems are presented. Section **7.4.1** presents the training results while section **7.4.2** presents the test accuracy results including the statistical analysis of the differences in performance. Finally sections **7.4.3** presents the automated design evolved configurations.

7.4.1 Training Results

Table **7.16** presents the training results. The rows represent datasets and the columns the classification approach. The results are the best training fitness ± standard deviation (at the

95% percentile confidence interval) over thirty independent runs for each approach. Training results reveal that *autoGE* trained well on 5 datasets and tied on 3 datasets and the *autoGA* approach trained well on 2 datasets and tied on 3 datasets while the manually designed *arithmetic* classifiers trained well on 1 dataset and tied on 1 dataset. Decision trees trained well on 1 dataset. On average across all datasets, the *autoGE* approach trained better than the other approaches with a training average of 71% followed by the *autoGA* with a training average of 69%. *Arithmetic* tree classifiers achieved a training average of 68%, *decision tree* classifiers averaged 64% and finally the *logical* tree classifiers achieved an average of 62%.

dataset	arithmetic	logical	decision tree	autoGA	autoGE
adobe	0.68±0.02	0.64±0.02	**0.89±0.02**	0.78±0.02	0.73±0.02
amazon	0.61±0.03	0.55±0.03	0.55±0.04	0.60±0.03	**0.65±0.03**
american express	0.66±0.03	0.59±0.03	0.71±0.02	**0.73±0.04**	0.72±0.03
barclays	0.76±0.03	0.59±0.03	0.59±0.03	0.75±0.03	**0.78±0.03**
centerPoint	**0.80±0.02**	0.55±0.02	0.54±0.02	**0.80±0.02**	0.79±0.02
dominos pizza	0.64±0.03	0.61±0.03	0.56±0.03	0.63±0.03	**0.67±0.03**
entergy	0.56±0.03	0.52±0.03	0.55±0.06	**0.60±0.03**	**0.60±0.03**
horizon pharmacy	0.57±0.03	0.52±0.03	0.58±0.03	0.57±0.03	**0.60±0.03**
pfizer	0.63±0.03	0.64 ±0.03	**0.70±0.03**	0.60±0.03	0.62±0.03
mcdonalds	0.63±0.03	0.59±0.03	**0.69±0.03**	0.60±0.03	0.68±0.03
microsoft	**0.89±0.02**	0.77±0.03	0.73±0.03	0.62±0.03	0.83±0.03
sap	0.55±0.03	0.54±0.03	0.54±0.03	**0.70±0.03**	**0.70±0.03**
standardchartered	**0.86±0.02**	0.77±0.02	0.70±0.02	0.89±0.02	**0.86±0.02**
time warner	0.72±0.03	0.83±0.03	0.77±0.03	0.82±0.02	**0.87±0.02**
walt disney	0.58±0.03	0.53±0.03	0.55±0.05	**0.69±0.03**	0.61±0.03
averages	0.68±0.03	0.62±0.03	0.64 ±0.03	0.69±0.03	**0.71±0.03**

Table 7.16 Financial forecasting training results

7.4.2 Testing Results

Table **7.17** presents the test results. The test results revealed that *autoGA* tested well on 4 datasets and tied on 5 datasets. The *autoGE* approach tested well on 3 datasets and tied on 6. The manually evolved *decision trees* tested well on 1 dataset and tied on 2 while the *arithmetic* tree classifiers tied on 2 datasets and the *logical* tree classifiers tied on 1 dataset. On average across all datasets, the *autoGE* approach tested better than the other approaches with a testing average of 69% followed by the *autoGA* approach with an average of 67%. The manually designed *arithmetic* tree classifiers achieved an average of 63%, *logical* tree classifiers averaged 61% and *decision trees* averaged 59%. The Friedman test was used to carry out the statistical analysis. The performance of the classification methods were

dataset	arithmetic	logical	decision tree	autoGA	autoGE
adobe	0.69±0.04	0.66±0.04	0.47±0.04	**0.70±0.03**	0.69±0.04
amazon	0.69±0.04	0.52±0.04	0.55±0.04	**0.71±0.04**	0.66±0.04
american express	0.62±0.04	0.61±0.04	0.64±0.06	**0.66±0.04**	**0.66±0.04**
barclays	0.70±0.04	0.54±0.04	0.55±0.04	0.80±0.03	**0.81±0.03**
center point	0.77±0.03	0.57±0.03	0.58±0.03	0.80±0.03	**0.81±0.03**
dominos pizza	073.±0.04	0.58±0.04	0.54±0.04	**0.77±0.04**	**0.77±0.04**
entergy	**0.54±0.04**	0.53±0.04	0.51±0.04	**0.54±0.04**	0.53±0.04
horizon pharmacy	0.48±0.04	**0.53±0.04**	0.51±0.04	**0.53±0.04**	**0.53±0.04**
pfizer	0.64±0.04	0.56±0.04	**0.74±0.04**	0.65±0.04	0.65±0.04
mcdonalds	0.58±0.04	0.64±0.04	0.60±0.04	0.66±0.04	**0.69±0.04**
microsoft	0.69±0.04	0.72±0.03	0.70±0.03	0.65±0.05	**0.81±0.03**
sap	0.54±0.04	0.55±0.04	**0.58±0.04**	0.56±0.04	**0.58±0.04**
standard chartered	**0.82±0.03**	0.75±0.03	0.57±0.03	0.81±0.02	**0.82±0.03**
time warner	0.41±0.04	0.86±0.04	0.74±0.04	**0.89±0.03**	**0.89±0.03**
Walt Disney	0.49±0.03	0.49±0.04	**0.51±0.04**	**0.51±0.04**	**0.51±0.04**
averages	0.63±0.04	0.61 ±0.04	0.59±0.04	0.67±0.04	**0.69±0.04**

Table 7.17 Financial forecasting tests results

ranked accordingly. Table **7.18** presents the average ranks of each method across all datasets. From the table, *autoGE* was ranked 1^{st} followed by the *autoGA* approach. The *arithmetic* tree classifiers were ranked 3rd followed by the *decision trees* and finally the *logical* tree classifiers.

	arithmetic	logical	decision	autoGA	autoGE
average rank	3.47	3.87	3.63	2.10	1.80
position	3	5	4	2	1

Table 7.18 Average ranks financial forecasting

Using the average ranks and Equation **2.6** the Friedman χ_F^2 was calculated to be :

$$\chi_F^2 = \frac{12*15}{5*6}\left[(3.47^2+3.87^2+3.63^2+2.10^2+1.80^2)-\frac{5*6^2}{4}\right] = 16.93 \tag{7.9}$$

and from Equation **2.7** Iman's F statistic was evaluated to:

$$F_f = \frac{(15-1)*16.93}{15*(5-1)-16.93} = 5.50 \tag{7.10}$$

The critical value $F_{(4,56)}$ for $\alpha = 0.05$ is given as 2.54 and since $5.50 > 2.54$ the null hypothesis which states that the 5 approaches perform equivalently was rejected and a post-hoc test using the Bonferroni–Dunn test for pairwise comparison was carried out. The critical difference

CD at the 95% level was evaluated to:

$$CD = 2.498 * \sqrt{\frac{5*6}{6*15}} = 1.44 \tag{7.11}$$

The difference between the average rank of *autoGE* and the best performing manually designed method *arithmetic* tree classifiers was found to be 1.67, which was greater than the critical difference implying that the performance of *autoGE* was significantly better. The difference between the average rank of *autoGA* approach and *arithmetic* tree classifiers is 1.37 which was less than the CD value which means the performance of *autoGA* was not significantly better than the best performing manual design approach. The differences in performance between the automated design approaches was not significant although *autoGE* evolved classifiers that achieved a higher testing accuracies. Table **7.19** is a summary of the statistical analysis results.

approach	difference	significance
arithmetic - autoGE	1.67(>1.44)	yes
arithmetic - autoGA	1.37(<1.44)	no
autoGE - autoGA	0.03(<1.44)	no

Table 7.19 Significance for financial forecasting problems

On this class of problems on average the *autoGE* approach trained well and tested better than the other approaches. The approach performed significantly better than the manual approach. Once again there was an element of consistency as the *autoGE* approach performed well on binary problems as the financial forecasting approach presented in this study is presented as a binary problem.

7.4.3 Configurations

Table **7.5** presents a listing of the 9 best performing automated designed configurations labelled as follows: i-*adobe*, ii-*amazon*, iii-*american express*, iv-*barclays*, v-*center point* vi-*dominos pizza*, vii-*mcdonalds*, viii-*microsoft* ix-*time warner* and x-*average of the manual design*. From the table, 8 configurations used the *arithmetic* tree types, and 1 was configured as a *decision tree* type. A *population size* of 200 was used in 5 configurations and a *population size* of 100 was used in 4 configurations. The *ramped half-and-half* and *full* tree generation methods were used in 4 configurations each while the *grow* method was used in 1 configuration only. *Initial tree depth* was set in the range [4-7]. *Maximum offspring depth* was set in the range [2-8] while the average manual value was 9. *Tournament selection* was

parameter	dataset									
	i	ii	iii	ii	iii	iv	v	vi	vii	viii
tree type	0	0	0	0	0	0	0	2	0	-
pop size	100	100	100	200	200	200	200	100	200	**300**
tree gen method	2	0	1	2	2	0	0	2	0	**2**
init tree depth	7	2	5	5	6	6	4	6	5	**3**
max offsp depth	8	6	2	6	5	5	5	6	4	**9**
selection method	1	1	1	1	0	0	0	1	1	**1**
selection size	4	6	3	2	-	-	-	3	4	**6**
crossover rate	63	70	23	46	10	47	47	2	1	**82**
mutation type	1	1	1	1	1	1	0	0	1	**0**
mutation depth	3	2	2	2	2	3	2	3	3	**3**
control flow	0	0	1	1	1	1	0	1	1	**0**
operator comb	1	3	2	3	2	1	2	1	3	**0**
fitness function	3	3	1	0	0	2	1	2	0	**0**
number of gens	200	100	100	200	200	50	200	200	50	**200**

Table 7.20 Financial forecasting: automated design configurations

used 6 times with the selection size set in the range [2-6]. *Fitness proportionate selection* was used 3 times. The *crossover rate* was set to values less than 50% on 7 of the 9 configurations. This was in contradiction with the popular convention of manual design which specifies that the crossover rate should normally be higher than the *mutation rate* [124]. *Shrink mutation* was used 7 times. *Mutation depth* was set to the value of 2 in 5 configurations and to the value of 3 in 4 configurations. The *random control flow* was used 6 times. *Predictive accuracy* was used 3 times as the fitness function, the *rate of missing chances*, *rate of failure* and the *weighted fitness function* were each used 2 times. Two hundred generations were set as the *termination* criteria 4 times, 50 twice and 100 once. Some of the parameter values determined by the automated design approach are not likely to be configured by a human designer. For example, in manual design, the intuitive norm is to set a crossover rate that is higher than the mutation rate. There is no discernible correlation between the automatically designed configurations and the dataset(problem) characteristics. Similarly, the averages of the manual designs outlined in viii of Table 7.5 also do not reflect any correlations with the automatically designed configurations. The differences in configurations reinforce the assertion that even for problem instances from the same problem domain, different configurations are required for effective classification.

7.5 Design Times

In this section, the design times of the automated design approaches are presented and discussed in relation to the design time of the manual approach. As previously stated parameter tuning under manual design is performed as described in section **2.4.8.1** of **Chapter 2**. The time taken for the manual design of GP classification algorithms for the values outlined in sections **4.7.1.1**, **4.7.1.2** and **4.7.2** of **Chapter 4** ranged from 8 - 10 days and those from section **4.7.3** of **Chapter 4** ranged from 5 - 7 days for each dataset. This included performing trial runs. Each day constituting approximately 10 man hours on average. The iterative nature of manual design led to high design times. For example, to tune for the *initial tree depth* parameter value in the range [2-20], 19 values were considered and for each value a number of trial runs (minimum 10) were performed because of the stochastic nature of GP which requires several runs to attain a normal distribution.

Table **7.21** presents the automated design times for binary classification problems. From the table, the average design time across all datasets, performed by *autoGA* is 29.38 hrs, with a minimum time of 16 hrs and a maximum of 36.39 hrs for the *mushroom* dataset. While *autoGE* averaged approximately 21 hrs with a minimum of 12 hrs and a maximum of 28 hrs hrs.

dataset	autoGA	autoGE
aus credit	36.02	20.18
appendicitis	16.26	17.48
breast cancer	32.22	19.02
cylinder band	34.46	28.16
diabetes (pima)	30.25	27.15
german credit	35.12	21.28
heart disease	25.43	18.21
hepatitis	16.36	12.21
liver disease	26.54	14.09
mushroom	36.39	27.39
tictactoe	34.12	22.34
average	29.38	20.69

Table 7.21 Binary class design times(hrs)

dataset	autoGA	autoGE
balance	17.13	15.39
post-operative	13.21	8.28
car	33.41	23.08
lymphography	16.15	12.50
cleveland	25.17	17.18
page blocks	46.35	42.11
dermatology	14.31	6.24
flare	36.12	21.08
glass	15.08	18.43
zoo	12.33	7.52
ecoli	33.25	18.18
average	24.26	17.27

Table 7.22 Multiclass design times(hrs)

Table **7.22** presents the automated design times for multiclass problems. *AutoGA* averaged 24.26 hrs with a maximum of 46.35 hrs for the *page blocks* dataset and a minimum of 12.33 hrs for the *zoo* dataset. *AutoGE* averaged 17.27 hrs with a maximum of 42.11 hrs for the *page block* and a minimum of 7.52 hrs for the *zoo* dataset.

Table **7.23** presents the automated design times for the NSL-KDD datasets. From the table,

the *autoGA* approach averaged longer design times averaging 56 hours while the *autoGE* averaged 53 hours. For the *autoGA* approach the *dos* dataset had the longest design time of approximately 67 hours. The shortest design time was achieved on the *u2r* dataset. For the *autoGE* approach the longest duration recorded was approximately 59 hours and the shortest was approximately 50 hours. A duration of 67 hours may seem like a long time but this is relatively a shorter period when compared to the manual design times.

Table **7.24** is a listing of the design times for each dataset for the *autoGA* and *autoGE* approaches applied to financial forecasting problems. On average across all datasets, the *autoGE* approach took approximately 15 hours to evolve classifiers while the *autoGA* took 12 hours. The shortest design time achieved by the *autoGA* was on the *barclays* dataset with a time of 6.72 hrs while the longest was experienced on the *walt disney* dataset. The shortest design time achieved by the *autoGE* approach was 9.73 hrs on the *entergy* dataset and the longest was 18.68 on the *microsoft* dataset.

dataset	autoGA	autoGE
normal	50.40	56.55
dos	66.42	51.89
probe	59.49	50.20
u2r	49.21	58.68
r2l	52.54	49.52
multi	61.32	54.33
average	56.56	53.52

Table 7.23 Design times(hrs)

dataset	autoGA	autoGE
adobe	13.61	12.14
amazon	12.78	14.72
american express	8.52	12.63
barclays	6.72	13.53
center point	14.60	16.58
dominos pizza	9.76	17.98
entergy	9.38	9.73
horizon pharmacy	11.96	13.00
pfizer	8.38	15.76
mcdonalds	11.30	17.05
microsoft	16.13	18.68
sap	11.90	13.73
standard chartered	15.20	14.63
time warner	13.71	16.60
walt disney	16.10	18.03
average	12.00	14.99

Table 7.24 Design times(hrs)

The automated design times were found to be shorter than the manual design times and as argued by Hutter et. al.[101] automated design liberates the algorithm designer to attend to other tasks.

7.6 Fitness Landscape Analysis

This section presents and compares the results of carrying out fitness landscape analyses of the design space searched by the automated design approaches. As outlined in section **3.4.3** of **Chapter 3** to evaluate the fitness landscape autocorrelation analysis was carried out on a selection of problem instances from the specified problem domains. The tables present the correlation length *cl* calculated from equations **2.15** and **2.16** while *n/cl* is the correlation length in relation to the fitness landscape diameter *n*. A low autocorrelation *cl* value is indicative of a more rugged landscape while a higher value indicates smoothness.

The results are presented as follows:

7.6.1 Binary Classification Problems

Table **7.25** presents the results of the fitness landscape analysis applied on binary classification problems.

	Dataset					
	australian credit		hepatitis		tictactoe	
	autoGA	*autoGE*	*autoGA*	*autoGE*	*autoGA*	*autoGE*
n	690	690	80	80	958	958
cl	3.29	3982.33	3.27	32.90	9.44	48.42
n/cl	209.72	0.17	24.46	2.43	101.48	19.78

Table 7.25 Fitness landscape analysis binary

Across all datasets, the *autoGE* approach searched smoother design space fitness landscapes than the *autoGA* approach. This means that the *autoGE* approach found it easier to solve the selected binary class problems than the *autoGA* approach. This is also in concurrence with the training and test accuracy results as the *autoGE* approach performed better than the *autoGA* approach. The smoothest landscape was on the *australian credit* dataset with a high correlation length of 3982.33. The most rugged landscape was also on the *australian credit* searched by the *autoGA* approach.

7.6.2 Multiclass Classification Problems

Table **7.26** presents the results of the fitness landscape analysis applied on selected multiclass classification problems.

The results reveal that the *autoGA* approach searches smoother landscapes than the *autoGE* approach on 2 of the 3 datasets, namely *dermatology* and *ecoli*. This result can be marginally correlated to the test accuracy results as the *autoGA* approach performed better

	Dataset					
	balance		dermatology		ecoli	
	autoGA	*autoGE*	*autoGA*	*autoGE*	*autoGA*	*autoGE*
n	625	625	358	358	336	336
cl	34.67	54.96	52.23	25.20	40.70	34.63
n/cl	18.02	11.37	6.85	14.21	8.26	9.70

Table 7.26 Fitness landscape analysis multiclass

than the *autoGE* on multiclass problems. This implies that the *autoGA* approach finds it easier to solve multiclass problems than the *autoGE* approach. However, the smoothest landscape was searched by the *autoGE* approach on the *balance* dataset although the *autoGA* approach managed to find the highest peak as it found the highest accuracy (98%) for this datasets. The *autoGE* also searched the most rugged landscape on the *dermatology* dataset.

7.6.3 Cybersecurity Problems

Table **7.27** reports on the fitness landscape analysis of the design space searched by *autoGA* and *autoGE* when applied to *set 1 (normal)*, *set 3 (probe)* and *set 6 (multiclass)* cybersecurity datasets. The *autoGA* approach searched smoother landscapes than the *autoGE* approach on

	Dataset					
	set 1		set 3		set 6	
	autoGA	*autoGE*	*autoGA*	*autoGE*	*autoGA*	*autoGE*
n	5000	5000	5000	5000	5000	5000
cl	33.08	20.32	12.40	7.48	6.12	5.19
n/cl	151.14	246.06	403.22	668.45	816.99	963.39

Table 7.27 Fitness landscape analysis cybersecurity

all three cybersecurity datasets as indicated by the higher correlation length attained by the *autoGA* approach. The smoothest landscape was on *set 1* searched by the *autoGA* approach with *n/cl* of 151.14. The *autoGE* approach searched the most rugged landscape on the *set 6 (multiclass)* dataset. The fitness landscape analysis results are in concurrence with the test accuracy results as the *autoGA* approach achieved the highest test average across all cybersecurity datasets. This implies that the *autoGA* approach found it easier to solve the cybersecurity problems than the *autoGE* approach. Although the *autoGE* approach achieved a better average ranking across all datasets.

7.6.4 Financial Forecasting Problems

From Table **7.28** a comparison of the correlation lengths evaluated shows that the *autoGA* approach has shorter lengths than the *autoGE* approach for each dataset.

	Dataset					
	Adobe		Barclays		Pfizer	
	autoGA	*autoGE*	*autoGA*	*autoGE*	*autoGA*	*autoGE*
n	1500	1500	1500	1500	1500	1500
cl	14.42	99.50	4.57	245	18.73	33.98
n/cl	104.02	15.08	336.59	61.23	80.10	44.1

Table 7.28 Fitness landscape analysis financial forecasting

This implies that the automated design space searched by the *autoGA* approach is more rugged than the design space searched by the *autoGE*. The correlation length differences are quite high on the *barclays* dataset which is also the least volatile. The *autoGA* approach evolves the most rugged landscape on the *barclays* dataset with an *n/cl* value of 336.59. This implies that for this class of problems the *autoGE* approach has less difficulty in finding more accurate solutions than the *autoGA* approach. This assertion is also supported by the test accuracy results as the *autoGE* performed better than the *autoGA*.

7.7 Summary

This chapter presented the results of the experiments conducted to achieve the objectives set out in this study. The effectiveness of automated designed classifiers compared to manually designed classifiers was firstly evaluated on binary classification problems obtained from multiple problem domains. The automated designed classifiers trained and tested better than manually designed classifiers. The automated designed classifiers were found to significantly perform better than manually designed classifiers. The difference in the classification accuracy performance between the *autoGA* and *autoGE* was found to be not significant although *autoGE* had higher training and test accuracies. A fitness landscape analysis of the design space revealed that the *autoGE* approach searched the smoothest fitness landscape when compared to the *autoGA* approach. Therefore *autoGE* found it easier to solve problems in this class of problems.

The effectiveness of the proposed automated design approaches was also evaluated on multiclass classification problems from multiple domains. The *autoGA* approach was found to perform better than *autoGE* and manual design on these class of problems. On average across all the multiclass datasets, *autoGA* trained better and achieved a higher predictive test

accuracy. The difference in performance between the *autoGA* approach and manual design was found to be statistically significant. However, the difference in performance between the *autoGA* approach and *autoGE* approach were found to be not statistically significant. The difference in performance between the *autoGE* approach and manual design were also found to be not statistically significant. This meant that the *autoGA* approach was found to be the most suitable for multiclass classification problems. Fitness landscape analysis was carried out on 3 datasets from the multiclass problems. The *autoGA* approach was found to have searched smoother landscapes on 2 of the 3 datasets.

Two problem domains, namely cybersecurity and financial forecasting were selected to evaluate the effectiveness of automated design when applied on instances from a single problem domain. The *autoGE* approach ranked better for the cybersecurity class of problems although the *autoGA* on average achieved a higher predictive test accuracy. The differences in performance between the automated design approaches and the manual design were found to be not statistical significant. The *autoGA* was found to perform better on the multiclass cybersecurity dataset, this was found to be consistent with the conclusion drawn from evaluating the multiclass problems i.e. *autoGA* performs well on multiclass problems. A fitness landscape analysis of the design space for the automated design approaches for this problem domain revealed that the *autoGA* approach evolves landscapes that are less rugged than the landscapes searched by the *autoGE* approach. Once again this was consistent with predictive accuracy test results as on average the *autoGA* approach achieved the highest result. On the second single domain class of problems i.e. financial forecasting the *autoGE* approach also trained and tested better. The approach ranked first and performed significantly better than manual design. The difference in performance between the *autoGA* and *autoGE* was found to be not statistically significant. The fitness landscape analysis revealed that the *autoGE* approach searched smoother landscapes than the *autoGA* approach. This result was found to be consistent with the training and test results as *autoGE* performed better than *autoGA*.

An analysis of the configurations revealed that different configurations worked well for different problem domains and different configurations worked well for different instances of the same problem domain.

An analysis of the design times showed that automated design approaches design GP configurations in significantly shorter durations than manual design.

The next chapter presents the conclusion of the study presented in this thesis.

Chapter 8

Conclusion and Future Work

8.1 Introduction

The main aim of the research carried out in this thesis was to test the hypothesis that the design of genetic programming classification algorithms for data classification may be automated by a genetic algorithm and grammatical evolution. The focus was to automate the design of GP classification algorithms thereby removing the dependence on a human expert designer and therefore, removing the weaknesses associated with manual design such as long design times, human bias and human preference. The aim was achieved by developing two new approaches that use a genetic algorithm and grammatical evolution to carry out the GP classification algorithm design process. The GP classifiers evolved by the new approaches were evaluated by applying them to binary and multiclass classification problems selected from i) the UCI benchmark problem repository, ii) cybersecurity and iii) financial forecasting problem domains.

The rest of the chapter is structured as follows: section **8.2** presents a brief analysis of automated design compared to manual design, section **8.3** presents a summary of the objectives achieved in this thesis, in section **8.4** a discussion of the conclusion is provided, future work is discussed in section **8.5**. Finally, section **8.6** provides a summary of the chapter.

8.2 Manual Design vs Automated Design

Manual design is unstructured and does not follow a formal approach. According to Barros et al. [20] to design an effective classification algorithm may take a number of weeks to several months depending on the algorithm complexity. This assertion is supported by the

design times experienced during the manual design of GP classification algorithms presented in this study. There are a number of advantages associated with automated design. The first advantage of automated design is the reduced design times achieved by automated design when compared to manual design. Thus manual design has a time overhead. Automated design reduces the man-hours spent on algorithm design. Automated design performs an unbiased search for the best configuration and the automation yields classifiers which are more accurate and relieves the human to attend to other tasks. Different combinations of design decisions which may not make logical sense to a human designer are tried and tested in a shorter time than manual design. Although, the search space of design decisions for the automated design is also not exhaustive it is much wider than the search space considered during manual design.

In [187] it is argued that effective manual design requires expert knowledge of the problem domain and of classification. However, this is not always possible [20]. To effectively use the approaches presented in this thesis the user is not required to be an expert in either the problem domain nor in classification.

Admittedly the automated design approaches presented in this study are themselves manually designed. However, this is only done once. Once the automated design approaches have been designed and implemented they can be used to design genetic programming classification algorithms suitable to evolve GP classifiers for a number of problem domains. As mentioned this is achieved with minimal alterations to the automated design applications to make them suitable to evolve GP classification algorithms that are suitable for the problem domain at hand.

Certain problem domains require a dynamic classification algorithm i.e. one which is able to adapt to changes in the problem. For example, the financial forecasting problem domain presents problems that are volatile. According to Lee et. al [125] a financial forecasting tool is expected to meet the following criteria: i) an acceptable accurate estimation of a forecast ii) cost-benefit trade-off i.e. is the effort of using the tool worth the benefit obtained from the forecast and iii) timeliness i.e. as events change and new information becomes available can the forecast be updated in time. The presented automated design approaches are shown to be able to evolve GP classification algorithms that meet the outlined criteria. As shown by the results of experiments using the financial forecasting problems automatically designed classifiers are able to achieve better forecasting results than manually designed classifiers thus meeting criterion i) by achieving acceptable accurate estimations of forecasts. Automated design also meets criterion ii) cost-benefit trade-off as the accuracy from automated design is significantly better than that achieved by manual design hence it is worth the effort. The third criterion relates to timeliness. Automated design meets this criterion as events that occur

today can be factored into the classification process the next day as indicated by the average design times (15 hrs) for financial forecasting problems presented here. It has been shown that manual design cannot meet the timeliness criterion.

There is a shift in the machine learning research community towards algorithms that are considered as *generalist* as opposed to *specialist* algorithms [33, 159]. Specialist algorithms are problem-tailored while generalist algorithms are able to generalise within (or across) a problem domain [199]. According to Haraldsson and Woodward [90] algorithms developed by automated design tend to be generalist algorithms. However, the authors argue against this approach and advocate for a general automated design framework that is able to design specialist algorithms that are able to solve problems in multiple domains. The automated design approaches presented in this study conform to this specification. With minor alterations, either of the presented automated design framework (*autoGA* or *autoGE*) is able to design GP classification algorithms which can be applied to a number of problem domains. Classification requires a specialist algorithm as the objective is to maximise the predictive accuracy. Classifiers are mappings of the relationship between data instance features and their class labels. Therefore, the data has an influence on the extracted classifier meaning a specialist algorithm is suitable for data classification.

The effectiveness of evolutionary algorithms depends on its configuration. The usefulness of genetic operators is still being debated in certain quarters of research [55]. In this thesis, this question is not answered however, it is left to the proposed approach to determine which operator works best for specific problems or problem instances. The selection of using one or more operators is determined by the automated design. Additionally, finding the right balance regarding operator application rates without bias if more than one operator is used is determined automatically. The automated design approaches evolve different GP classification algorithm configurations for different problem domains. Also different GP classification algorithm configurations are evolved for problem instances from the same problem domain.

8.3 Objectives

This section lists the objectives which were set out in section **1.2** of **Chapter 1** and describes how they were met.

1. **To automate the design of genetic programming classification algorithms using a genetic algorithm.**
   To achieve this objective a genetic algorithm system that automatically designs GP classification algorithms was designed and developed. The system used a genetic

algorithm to make GP classification algorithm design decisions. The genetic algorithm was used to determine i) parameter values ii) the genetic operators and iii) the control flow for the GP classification algorithms. The proposed approach termed *autoGA* used a genetic algorithm to evolve GP classification algorithms. Each individual of the *autoGA* system represented a GP classification algorithm configuration. The evolved configurations were gradually refined from generation to generation until a stopping criterion was met and the best configuration was outputted.

2. **To automate the design of genetic programming classification algorithms using grammatical evolution.**

   Similarly to objective 1 a system to automatically configure GP classification algorithms using grammatical evolution was designed and implemented. The system was termed *autoGE*. Each individual of the *autoGE* system was encoded to represent a GP classification algorithm configuration. A grammar which contained domain knowledge of GP classification algorithm design decisions was defined. The *autoGE* system was used to determine i) parameter values ii) the genetic operators and iii) the control flow for the GP classification algorithms. Using grammatical evolution GP classification algorithm configurations were evolved until the best configuration was obtained.

3. **To compare the effectiveness of genetic programming classifiers evolved by a genetic algorithm to manually designed genetic programming classifiers.**

   To achieve this objective, a standard GP classification algorithm system was designed and developed using a manual design approach. The manual system was developed to evolve the 3 commonly used GP classifier types, namely *arithmetic tree*, *logical tree,* and *decision tree*. The two systems, viz. manual GP and *autoGA* were used to evolve GP classifiers which were applied to binary and multiclass classification problems. Three classes of binary classification problems were considered as follows: i) multiple domain problems ii) cybersecurity problems and iii) financial forecasting problems. Two classes were used for multiclass classification problems. these were problems from multiple domains and cybersecurity problems The predictive accuracy of the best performing manually evolved GP classifier was compared to the predictive accuracy of the best performing GP classifier evolved by the *autoGA* approach. For all class of problems considered on average the best *autoGA* evolved GP classifier performed better than the best manually evolved GP classifier. On average the performance of the best *autoGA* evolved GP classifiers were found to be significantly better than the best manually designed GP classifiers on the multiple domain class of problems. Although the difference in performance on the cybersecurity class of problems was found to

be not significant, the *autoGA* evolved GP classifiers on average achieved a higher predictive accuracy.

4. **To compare the effectiveness of genetic programming classifiers evolved by grammatical evolution to manually designed genetic programming classifiers.**

   To achieve this objective, the predictive accuracy of the best performing GP classifiers evolved by the *autoGE* approach applied to the binary and multiclass classification problems considered in the study were compared to the results of the best performing manually designed GP classifiers for the same classification problems. The results indicated that the *autoGE* approach significantly outperformed the manual design approach on the multiple domain binary problems and on the financial forecasting problems. On the multiclass multiple domain problems and cybersecurity, there was no significant differences in performance although on average across all datasets the *autoGE* approach achieved a higher predictive accuracy.

5. **Compare the performance of genetic programming classifiers evolved by a genetic algorithm to the performance of genetic programming classifiers evolved by grammatical evolution on selected problems.**

   To achieve this objective the predictive accuracy of the best performing GP classifiers evolved by *autoGA* were compared to the predictive accuracy of the best performing GP classifiers evolved by *autoGE*. For all the class of problems considered the differences in performance between the *autoGA* and *autoGE* approaches were found to be not significant. The GP classifiers evolved by *autoGA* achieved a better predictive accuracy average across multiclass problems than *autoGE* evolved GP classifiers. On average *autoGE* evolved GP classifiers performed better than *autoGA* evolved GP classifiers on binary classification problems.

6. **Compare the manual design configurations to the automated design configurations.**

   To achieve this objective parameter tuning was performed for each dataset in the set of binary class of problems from multiple domains. For the multiclass problems, cybersecurity and financial forecasting problems a subset of problems was selected and parameter settings were tuned and used for the rest of the problems. These set of parameters were compared to the configurations of the best performing automated designed classifiers. From the comparisons, there were no discernible patterns or similarities observed. Configuration were established to be problem dependent. Automated design was able to consider a wider range of design decisions than manual design.

7. **Compare the manual design time to the automated design time on selected problems.**

   Manual design time was found to be longer than automated design time. The iterative nature of manual design time compounded by the trial runs led to longer design times. The differences in design times of the automated approaches were not significantly different, although the *autoGE* achieves shorter design times than *autoGA* in most cases.

8. **Compare the fitness landscape of the design space searched by a genetic algorithm to the fitness landscape searched by grammatical evolution for a specific class of problems.**

   A fitness landscape analysis is generally used to analyse the ease with which an algorithm can solve a problem at hand. To achieve this objective a fitness landscape analysis of the design spaces searched by the *autoGA* and *autoGE* approaches was carried using the autocorrelation approach. The results of the analysis were compared for the respective data instances analysed. The *autoGE* approach was found to search smoother landscapes than the *autoGA* approach on binary and financial forecasting class of problems. While the *autoGA* approach searched less rugged landscapes than the *autoGE* approach on multiclass and cybersecurity problems.

8.4 Conclusion

This section presents conclusions that can be drawn from the objectives presented above.

This study proposed an approach that automates the design of GP classification algorithms for data classification. The proposed strategy is able to automatically configure GP classification algorithms. Although it can be argued that the proposed approach introduces more parameters as both the genetic algorithm in *autoGA* and grammatical evolution in *autoGE* need to be configured. However, to configure GP classification algorithms there are more parameters to be considered than for *autoGA* or *autoGE* as a result the configuration of *autoGA* or *autoGE* is less complex than for GP classification algorithms. The state-of-the-art classification methods are usually tailored for a specific problem by classification algorithm designers (experts). These systems are usually expensive and proprietary. In this study, we proposed an approach that requires less expert knowledge and is suitable to be applied to multiple problem domains. Both proposed approaches i.e. *autoGA* and *autoGE* reduce the man-hours required for the design of GP classification algorithms. The proposed methods

are shown to work on problems from specific domains as well as being able to generalise across problem domains.

8.5 Future Work

Extension of the research presented in this thesis will involve the following:

- **Automated design of multi-objective algorithms**
  Generally, classification is a multi-objective problem considering the number of metrics which can be obtained from a classification process such as *predictive accuracy, error rate, false positive rate, true positive rate, etc*. The popular approach is to present the classification result as a single value of predictive accuracy or a weighted final value. An automated approach that incorporates a multi-objective fitness function will present more meaningful results. This will involve investigating the automated design of multi-objective GP classification algorithms for classification problems.

- **Correlation of the solution space and the design space**
  In the study presented in this thesis an analysis of the fitness landscape of the design spaces evolved by the proposed approaches were compared. It will also be beneficial to analyse the design space in correlation to the solution space. The results obtained from such a study may aid in increasing the number of design decisions for automated design. Additionally, autocorrelation was selected for the fitness landscape analysis. Using other metrics to analyse the design search space may convey further useful information that may lead to designs that improve the functionality of the proposed automated design.

- **Automating the design of the automating algorithm.**
  Genetic programming is a parameterised evolutionary algorithm and in this study we proposed using a genetic algorithm and grammatical evolution to automate the design of GP. Both GA and GE are also parameterised evolutionary algorithms this raises the question of whether is there any benefit in automating the design of the automating application i.e *autoGA* and *autoGE*. This approach is worth investigating as manual design is carried out for the design of both automated design approaches. If there are benefits to automating the GP classification algorithms there may be benefits in automating *autoGA* and *autoGE*. What was apparent from the results was that on some problem domains one approach may on average have performed better than the other but not all problem instances of that domain. For example, on average the *autoGE* approach performed well on financial forecasting problems but the *autoGA* performed

better on certain instances of the problem domain such as the *amazon* dataset. This presents an opportunity for further research such as automated algorithm selection of the automated design approach. Where the best performing automated design approach may be used for a specific problem instance.

- **Co-evolution of *autoGA* and *autoGE***
  Co-evolution is generally described as the evolution of two or more populations. The populations are usually in competition and they have a coupled fitness function. According to Nolfi and Floreano [160] the competition between the populations leads to the evolution of complex structures which may be very efficient at solving the problem at hand. Future research on this aspect will involve investigation the conditions under which co-evolution may used by populations evolved by *autoGA* and *autoGE* to evolve complex GP classification algorithms capable of achieving higher classification accuracies.

8.6 Summary

This chapter presented a brief comparison of manual design and automated design based on the results obtained in this study. An outline of the objectives and how they were met is also presented. A conclusion of the study and future work are presented.

References

[1] Affenzeller, M., Wagner, S., Winkler, S., and Beham, A. (2009). *Genetic algorithms and genetic programming: modern concepts and practical applications*. Crc Press.

[2] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the workshop on languages in social media*, pages 30–38. Association for Computational Linguistics.

[3] Aggarwal, C. C. (2014). *Data classification: algorithms and applications*. CRC Press.

[4] Ahn, C. W. and Ramakrishna, R. S. (2002). A genetic algorithm for shortest path routing problem and the sizing of populations. *IEEE transactions on evolutionary computation*, 6(6):566–579.

[5] Aitkenhead, M. J. (2008). A co-evolving decision tree classification method. *Expert Systems with Applications*, 34(1):18–25.

[6] Al-Sahaf, H., Zhang, M., Johnston, M., and Verma, B. (2015). Image descriptor: A genetic programming approach to multiclass texture classification. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2460–2467. IEEE.

[7] Albayrak, M. and Allahverdi, N. (2011). Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications*, 38(3):1313–1320.

[8] Aleti, A. and Moser, I. (2016). A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)*, 49(3):56.

[9] Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141.

[10] Alpaydm, E. (1999). Combined $5\times$ 2 cv f test for comparing supervised classification learning algorithms. *Neural computation*, 11(8):1885–1892.

[11] Aly, M. (2005). Survey on multiclass classification methods. *Neural Netw*, 19:1–9.

[12] Angeline, P. J. (1997). Subtree crossover: Building block engine or macromutation. *Genetic programming*, 97:9–17.

[13] Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer.

[14] Arcanjo, F. d. L., Pappa, G. L., Bicalho, P. V., Meira Jr, W., and da Silva, A. S. (2011). Semi-supervised genetic programming for classification. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1259–1266. ACM.

[15] Asuncion, A. and Newman, D. (2007). Uci machine learning repository.

[16] Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424.

[17] Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic programming: an introduction*, volume 1. Morgan Kaufmann San Francisco.

[18] Barros, R. C., Basgalupp, M. P., and de Carvalho, A. C. (2015). Investigating fitness functions for a hyper-heuristic evolutionary algorithm in the context of balanced and imbalanced data classification. *Genetic Programming and Evolvable Machines*, 16(3):241–281.

[19] Barros, R. C., Basgalupp, M. P., Freitas, A. A., and De Carvalho, A. C. (2014a). Evolutionary design of decision-tree algorithms tailored to microarray gene expression data sets. *IEEE Transactions on Evolutionary Computation*, 18(6):873–892.

[20] Barros, R. C., Carvalho, A. C. P. d. L., Freitas, A. A., et al. (2014b). On the automatic design of decision-tree induction algorithms. In *Congresso da Sociedade Brasileira de Computação, XXXIV; Concurso de Teses e Dissertações, XXVII*. Sociedade Brasileira de Computação (SBC).

[21] Bartoli, A., Davanzo, G., De Lorenzo, A., Mauri, M., Medvet, E., and Sorio, E. (2012). Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 1477–1478. ACM.

[22] Bhowan, U., Johnston, M., and Zhang, M. (2009). Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2802–2809. IEEE.

[23] Bibi, S., Tsoumakas, G., Stamelos, I., and Vlahavas, I. (2008). Regression via classification applied on software defect estimation. *Expert Systems with Applications*, 34(3):2091–2101.

[24] Birattari, M., Stützle, T., Paquete, L., and Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc.

[25] Bojarczuk, C. C., Lopes, H. S., and Freitas, A. A. (1999). Discovering comprehensible classification rules using genetic programming: a case study in a medical domain. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, pages 953–958. Morgan Kaufmann Publishers Inc.

[26] Bojarczuk, C. C., Lopes, H. S., Freitas, A. A., and Michalkiewicz, E. L. (2004). A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medicine*, 30(1):27–48.

[27] Braga-Neto, U. M. and Dougherty, E. R. (2004). Is cross-validation valid for small-sample microarray classification? *Bioinformatics*, 20(3):374–380.

[28] Brain, Z. E., Addicoat, M. A., et al. (2010). Using meta-genetic algorithms to tune parameters of genetic algorithms to find lowest energy molecular conformers. In *ALIFE*, pages 378–385.

[29] Brameier, M. F. and Banzhaf, W. (2007). *Linear genetic programming*. Springer Science & Business Media.

[30] Bramer, M. (2013). *Principles of data mining*. Springer Publishing Company, Incorporated.

[31] Brieman, L., Friedman, J., Olshen, R., and Stone, C. (1984). Classification and regression trees. belmont (ca): Wadsworth. *Google Scholar*.

[32] Brindle, A. (1981). Genetic algorithms for function optimization.

[33] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer.

[34] Byrne, J., Fenton, M., Hemberg, E., McDermott, J., O'Neill, M., Shotton, E., and Nally, C. (2011). Combining structural analysis and multi-objective criteria for evolutionary architectural design. In *European Conference on the Applications of Evolutionary Computation*, pages 204–213. Springer.

[35] Carvalho, M. G., Laender, A. H., Gonçalves, M. A., and da Silva, A. S. (2008). Replica identification using genetic programming. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1801–1806. ACM.

[36] Chareka, T. and Pillay, N. (2016). A study of fitness functions for data classification using grammatical evolution. In *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016*, pages 1–4. IEEE.

[37] Chen, Z. and Lu, S. (2007). A genetic programming approach for classification of textures based on wavelet analysis. In *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on*, pages 1–6. IEEE.

[38] Chennupati, G. (2015). *Grammatical Evolution+ Multi-Cores= Automatic Parallel Programming'*. PhD thesis, PhD thesis, University of Limerick.

[39] Chmielewski, M. R. and Grzymala-Busse, J. W. (1996). Global discretization of continuous attributes as preprocessing for machine learning. *International journal of approximate reasoning*, 15(4):319–331.

[40] Corcoran, A. L. and Sen, S. (1994). Using real-valued genetic algorithms to evolve rule sets for classification. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 120–124. IEEE.

[41] Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the first international conference on genetic algorithms*, pages 183–187.

[42] Dandekar, T. and Argos, P. (1992). Potential of genetic algorithms in protein folding and protein engineering simulations. *Protein Engineering, Design and Selection*, 5(7):637–645.

[43] Darwin, C. (1968). On the origin of species by means of natural selection. 1859. *London: Murray Google Scholar*.

[44] Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164.

[45] De Falco, I., Della Cioppa, A., and Tarantino, E. (2002). Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269.

[46] De Freitas, J., Pappa, G. L., da Silva, A. S., Gonc, M. A., Moura, E., Veloso, A., Laender, A. H., de Carvalho, M. G., et al. (2010). Active learning genetic programming for record deduplication. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[47] De'ath, G. and Fabricius, K. E. (2000). Classification and regression trees: a powerful yet simple technique for ecological data analysis. *Ecology*, 81(11):3178–3192.

[48] DeJong, K. A. and Spears, W. M. (1990). Learning concept classification rules using genetic algorithms. Technical report, GEORGE MASON UNIV FAIRFAX VA.

[49] Della Croce, F., Tadei, R., and Volta, G. (1995). A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24.

[50] Demeyer, S. (2011). Research methods in computer science. In *ICSM*, page 600.

[51] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30.

[52] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923.

[53] Diosan, L. S. and Oltean, M. (2007). Evolving evolutionary algorithms using evolutionary algorithms. In *Proceedings of the 9th annual conference companion on Genetic and evolutionary computation*, pages 2442–2449. ACM.

[54] Dobslaw, F. (2010). A parameter tuning framework for metaheuristics based on design of experiments and artificial neural networks. In *International Conference on Computer Mathematics and Natural Computing*. WASET.

[55] Doerr, B., Happ, E., and Klein, C. (2012). Crossover can provably be useful in evolutionary computation. *Theoretical Computer Science*, 425:17–33.

[56] Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Machine Learning Proceedings 1995*, pages 194–202. Elsevier.

[57] Drake, J. H., Kililis, N., and Özcan, E. (2013). Generation of vns components with grammatical evolution for vehicle routing. In *European Conference on Genetic Programming*, pages 25–36. Springer.

[58] Eiben, A. E., Michalewicz, Z., Schoenauer, M., and Smith, J. E. (2007). Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer.

[59] Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31.

[60] Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

[61] Eiben, Á. E., van Kemenade, C. H., and Kok, J. N. (1995). Orgy in the computer: Multi-parent reproduction in genetic algorithms. In *European Conference on Artificial Life*, pages 934–945. Springer.

[62] Engen, V. (2010). *Machine learning for network based intrusion detection: an investigation into discrepancies in findings with the KDD cup'99 data set and multi-objective evolution of neural network classifier ensembles from imbalanced data.* PhD thesis, Bournemouth University.

[63] Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In *Foundations of genetic algorithms*, volume 2, pages 187–202. Elsevier.

[64] Espejo, P. G., Ventura, S., and Herrera, F. (2010). A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):121–144.

[65] Estrada-Gil, J. K., Fernández-López, J. C., Hernández-Lemus, E., Silva-Zolezzi, I., Hidalgo-Miranda, A., Jiménez-Sánchez, G., and Vallejo-Clemente, E. E. (2007). Gpdti: A genetic programming decision tree induction method to find epistatic effects in common complex diseases. *Bioinformatics*, 23(13):i167–i174.

[66] Fallah-Mehdipour, E., Haddad, O. B., and Mariño, M. (2013). Developing reservoir operational decision rule by genetic programming. *Journal of Hydroinformatics*, 15(1):103–119.

[67] Faraoun, K. and Boukelif, A. (2006). Genetic programming approach for multi-category pattern classification applied to network intrusions detection. *International Journal of Computational Intelligence and Applications*, 6(01):77–99.

[68] Fisher, R. A. (1938). The statistical utilization of multiple measurements. *Annals of Human Genetics*, 8(4):376–386.

[69] Flood, R., Hodrick, R. J., and Kaplan, P. (1986). An evaluation of recent evidence on stock market bubbles.

[70] Fogel, L. J. (1994). Evolutionary programming in perspective: The top-down view. *Computational intelligence: Imitating life*.

[71] Folino, G. and Pisani, F. S. (2015). Combining ensemble of classifiers by using genetic programming for cyber security applications. In *European Conference on the Applications of Evolutionary Computation*, pages 54–66. Springer.

[72] Folino, G., Pizzuti, C., and Spezzano, G. (2001). Parallel genetic programming for decision tree induction. In *Tools with Artificial Intelligence, Proceedings of the 13th International Conference on*, pages 129–135. IEEE.

[73] Freitas, A. A. (1997). A genetic programming framework for two data mining tasks: classification and generalized rule induction. In *Genetic Programming 1997: Proc 2nd Annual Conf*, pages 96–101. Citeseer.

[74] Freitas, A. A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In *Advances in evolutionary computing*, pages 819–845. Springer.

[75] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

[76] García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064.

[77] Garcia, S., Luengo, J., Sáez, J. A., Lopez, V., and Herrera, F. (2013). A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750.

[78] Gathercole, C. and Ross, P. (1996). An adverse interaction between crossover and restricted tree depth in genetic programming. In *Proceedings of the 1st annual conference on genetic programming*, pages 291–296. MIT Press.

[79] Golberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. addion wesley. *Reading*.

[80] Goldberg, D. E. (2006). *Genetic algorithms*. Pearson Education India.

[81] Greene, C. S., Hill, D. P., and Moore, J. H. (2010). Environmental sensing of expert knowledge in a computational evolution system for complex problem solving in human genetics. In *Genetic Programming Theory and Practice VII*, pages 19–36. Springer.

[82] Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, 16(1):122–128.

[83] Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 2031–2038. IEEE.

[84] Griffin, J. M. and Chen, X. (2009). Multiple classification of the acoustic emission signals extracted during burn and chatter anomalies using genetic programming. *The International Journal of Advanced Manufacturing Technology*, 45(11-12):1152.

[85] Gunn, S. R. et al. (1998). Support vector machines for classification and regression. *ISIS technical report*, 14(1):5–16.

[86] Hagan, M. T., Demuth, H. B., Beale, M. H., et al. (1996). *Neural network design*, volume 20. Pws Pub. Boston.

[87] Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

[88] Hand, D. J. (1997). *Construction and assessment of classification rules*. Wiley.

[89] Hansen, J. V., Lowry, P. B., Meservy, R. D., and McDonald, D. M. (2007). Genetic programming for prevention of cyberterrorism through dynamic and evolving intrusion detection. *Decision Support Systems*, 43(4):1362–1374.

[90] Haraldsson, S. O. and Woodward, J. R. (2014). Automated design of algorithms and genetic improvement: contrast and commonalities. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1373–1380. ACM.

[91] Harper, R. and Blair, A. (2005). A structure preserving crossover in grammatical evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 3, pages 2537–2544. IEEE.

[92] Hirsch, L., Saeedi, M., and Hirsch, R. (2005). Evolving text classification rules with genetic programming. *Applied Artificial Intelligence*, 19(7):659–676.

[93] Holst, T. and Pulliam, T. (2001). Aerodynamic shape optimization using a real-number-encoded genetic algorithm. In *19th AIAA Applied Aerodynamics Conference*, page 2473.

[94] Hong, J.-H. and Cho, S.-B. (2004). Lymphoma cancer classification using genetic programming with snr features. In *European Conference on Genetic Programming*, pages 78–88. Springer.

[95] Hong, L., Woodward, J., Li, J., and Özcan, E. (2013). Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. In *European Conference on Genetic Programming*, pages 85–96. Springer.

[96] Hordijk, W. (1996). A measure of landscapes. *Evolutionary computation*, 4(4):335–360.

[97] Hu, T. and Banzhaf, W. (2008). Nonsynonymous to synonymous substitution ratio k_ {\mathrm a}/k_ {\mathrm s}: Measurement for rate of evolution in evolutionary computation. In *International Conference on Parallel Problem Solving from Nature*, pages 448–457. Springer.

[98] Hutter, F. (2009). *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia.

[99] Hutter, F., Babic, D., Hoos, H. H., and Hu, A. J. (2007a). Boosting verification by automatic tuning of decision procedures. In *Formal Methods in Computer Aided Design, 2007. FMCAD'07*, pages 27–34. IEEE.

[100] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

[101] Hutter, F., Hoos, H. H., and Stützle, T. (2007b). Automatic algorithm configuration based on local search. In *Aaai*, volume 7, pages 1152–1157.

[102] Iman, R. L. and Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics-Theory and Methods*, 9(6):571–595.

[103] Isele, R. and Bizer, C. (2011). Learning linkage rules using genetic programming. In *Proceedings of the 6th International Conference on Ontology Matching-Volume 814*, pages 13–24. CEUR-WS. org.

[104] Isele, R. and Bizer, C. (2012). Learning expressive linkage rules using genetic programming. *Proceedings of the VLDB Endowment*, 5(11):1638–1649.

[105] Isele, R. and Bizer, C. (2013). Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23:2–15.

[106] Ishibuchi, H., Nozaki, K., Yamamoto, N., and Tanaka, H. (1995). Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on fuzzy systems*, 3(3):260–270.

[107] Johnson, C. (2006a). What is Research in Computing Science. http://http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/research.html. [Online; accessed 19-January-2017].

[108] Johnson, C. (2006b). What is research in computing science. *Computer Science Dept., Glasgow University. Electronic resource: http://www. dcs. gla. ac. uk/˜ johnson/teaching/research_skills/research. ht ml*.

[109] Johnson, H. E., Gilbert, R. J., Winson, M. K., Goodacre, R., Smith, A. R., Rowland, J. J., Hall, M. A., and Kell, D. B. (2000). Explanatory analysis of the metabolome using genetic programming of simple, interpretable rules. *Genetic Programming and Evolvable Machines*, 1(3):243–258.

[110] Jones, T. et al. (1995). *Evolutionary algorithms, fitness landscapes and search*. PhD thesis, Citeseer.

[111] Karafotias, G., Hoogendoorn, M., and Eiben, Á. E. (2015). Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19(2):167–187.

[112] Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In *Proceedings of the tenth national conference on Artificial intelligence*, pages 123–128. Aaai Press.

[113] Khoshgoftaar, T. M., Seliya, N., and Liu, Y. (2003). Genetic programming-based decision trees for software quality classification. In *Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on*, pages 374–383. IEEE.

[114] Kishore, J. K., Patnaik, L. M., Mani, V., and Agrawal, V. (2000). Application of genetic programming for multicategory pattern classification. *IEEE transactions on evolutionary computation*, 4(3):242–258.

[115] Kotsiantis, S. and Kanellopoulos, D. (2006). Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58.

[116] Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117.

[117] Koza, J. R. (1990). Concept formation and decision tree induction using the genetic programming paradigm. In *International Conference on Parallel Problem Solving from Nature*, pages 124–128. Springer.

[118] Koza, J. R. (1992a). *Genetic Programming II, Automatic Discovery of Reusable Subprograms*. MIT Press, Cambridge, MA.

[119] Koza, J. R. (1992b). *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press.

[120] Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112.

[121] Kurgan, L. A. and Cios, K. J. (2004). Caim discretization algorithm. *IEEE transactions on Knowledge and Data Engineering*, 16(2):145–153.

[122] La Cava, W., Silva, S., Danai, K., Spector, L., Vanneschi, L., and Moore, J. H. (2018). Multidimensional genetic programming for multiclass classification. *Swarm and Evolutionary Computation*.

[123] Langdon, W. B. and Buxton, B. F. (2001). Genetic programming for combining classifiers. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 66–73. Morgan Kaufmann Publishers Inc.

[124] Langdon, W. B. and Poli, R. (2013). *Foundations of genetic programming*. Springer Science & Business Media.

[125] Lee, A. C., Lee, J. C., and Lee, C. F. (2009). *Financial analysis, planning & forecasting: Theory and application*. World Scientific.

[126] Lessmann, S., Baesens, B., Seow, H.-V., and Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, 247(1):124–136.

[127] Li, J. (2000). *FGP: a genetic programming based tool for financial forecasting.* PhD thesis, University of Essex.

[128] Li, J. (2001). Fgp: A genetic programming based financial forecasting tool. *Unpublished doctoral dissertation, Department of Computer Science, University of Essex*.

[129] Li, X. and Ciesielski, V. (2004). Using loops in genetic programming for a two class binary image classification problem. In *Australasian Joint Conference on Artificial Intelligence*, pages 898–909. Springer.

[130] Liao, W. (1987). Graph bipartitioning problem. *Physical review letters*, 59(15):1625.

[131] Lim, A. H. and Lee, C.-S. (2010). Processing online analytics with classification and association rule mining. *Knowledge-Based Systems*, 23(3):248–255.

[132] Lin, J.-Y., Ke, H.-R., Chien, B.-C., and Yang, W.-P. (2007). Designing a classifier by a layered multi-population genetic programming approach. *Pattern Recognition*, 40(8):2211–2225.

[133] Lin, J.-Y., Ke, H.-R., Chien, B.-C., and Yang, W.-P. (2008). Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Systems with Applications*, 34(2):1384–1393.

[134] Liu, H., Hussain, F., Tan, C. L., and Dash, M. (2002). Discretization: An enabling technique. *Data mining and knowledge discovery*, 6(4):393–423.

[135] Liu, H. and Setiono, R. (1997). Feature selection via discretization. *IEEE Transactions on knowledge and Data Engineering*, 9(4):642–645.

[136] Lobo, F., Lima, C. F., and Michalewicz, Z. (2007). *Parameter setting in evolutionary algorithms*, volume 54. Springer Science & Business Media.

[137] López-Ibánez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

[138] Lourenço, N., Pereira, F., and Costa, E. (2012). Evolving evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 51–58. ACM.

[139] Lourenço, N., Pereira, F. B., and Costa, E. (2013). The importance of the learning conditions in hyper-heuristics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1525–1532. ACM.

[140] Loveard, T. and Ciesielski, V. (2001). Representing classification problems in genetic programming. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 2, pages 1070–1077. IEEE.

[141] Luke, S. and Panait, L. (2001). A survey and comparison of tree generation algorithms. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 81–88. Morgan Kaufmann Publishers Inc.

[142] Ma, C. Y. and Wang, X. Z. (2009). Inductive data mining based on genetic programming: automatic generation of decision trees from data for process historical data analysis. *Computers & Chemical Engineering*, 33(10):1602–1616.

[143] Maden, İ., Uyar, A., and Ozcan, E. (2009). Landscape analysis of simple perturbative hyperheuristics. In *Mendel*, volume 2009, page 15th.

[144] Malan, K. M. and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241:148–163.

[145] Marsland, S. (2015). *Machine learning: an algorithmic perspective*. CRC press.

[146] Merz, P. and Freisleben, B. (2000). Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91.

[147] Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384.

[148] Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *European Conference on Genetic Programming*, pages 121–132. Springer.

[149] Miranda, P. B. and Prudêncio, R. B. (2015). Gefpso: A framework for pso optimization based on grammatical evolution. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1087–1094. ACM.

[150] Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

[151] Montero, E., Riff, M.-C., and Neveu, B. (2014). A beginner's guide to tuning methods. *Applied Soft Computing*, 17:39–51.

[152] Mousavi, S., Esfahanipour, A., and Zarandi, M. H. F. (2014). A novel approach to dynamic portfolio trading system using multitree genetic programming. *Knowledge-Based Systems*, 66:68–81.

[153] Muni, D. P., Pal, N. R., and Das, J. (2004). A novel approach to design classifiers using genetic programming. *IEEE transactions on evolutionary computation*, 8(2):183–196.

[154] Munoz, L., Silva, S., and Trujillo, L. (2015). M3gp–multiclass classification with gp. In *European Conference on Genetic Programming*, pages 78–91. Springer.

[155] Nannen, V. and Eiben, A. (2007a). Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 103–110. IEEE.

[156] Nannen, V. and Eiben, A. E. (2007b). Relevance estimation and value calibration of evolutionary algorithm parameters. In *IJCAI*, volume 7, pages 975–980.

[157] Naudts, B. and Kallel, L. (2000). Comparison of summary statistics of fitness landscapes. *IEEE Trans. Evol. Comp*, 4:1–15.

[158] Nazif, H. and Lee, L. S. (2012). Optimised crossover genetic algorithm for capacitated vehicle routing problem. *Applied Mathematical Modelling*, 36(5):2110–2117.

[159] Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2012). Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming. In *European Conference on Genetic Programming*, pages 121–133. Springer.

[160] Nolfi, S. and Floreano, D. (1998). How co-evolution can enhance the adaptive power of artificial evolution: Implications for evolutionary robotics. In *European Workshop on Evolutionary Robotics*, pages 22–38. Springer.

[161] Oates, B. J. (2005). *Researching information systems and computing*. Sage.

[162] Ochoa, G., Qu, R., and Burke, E. K. (2009a). Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 341–348. ACM.

[163] Ochoa, G., Vázquez-Rodríguez, J. A., Petrovic, S., and Burke, E. (2009b). Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 1873–1880. IEEE.

[164] Oltean, M. and Dioşan, L. (2009). An autonomous gp-based system for regression and classification problems. *Applied Soft Computing*, 9(1):49–60.

[165] Ong, C.-S., Huang, J.-J., and Tzeng, G.-H. (2005). Building credit scoring models using genetic programming. *Expert Systems with Applications*, 29(1):41–47.

[166] O'Neill, M. and Ryan, C. (2000). Crossover in grammatical evolution: A smooth operator? In *European Conference on Genetic Programming*, pages 149–162. Springer.

[167] O'Neill, M., Ryan, C., Keijzer, M., and Cattolico, M. (2001). Crossover in grammatical evolution: The search continues. In *European Conference on Genetic Programming*, pages 337–347. Springer.

[168] Papagelis, A. and Kalles, D. (2001). Breeding decision trees using evolutionary techniques. In *ICML*, volume 1, pages 393–400.

[169] Patki, P. S. and Kelkar, V. V. (2013). Classification using different normalization techniques in support vector machine. In *International Conference on Communication Technology*, pages 17–19.

[170] Perlman, R., Kaufman, C., and Speciner, M. (2016). *Network security: private communication in a public world*. Pearson Education India.

[171] Phyu, T. N. (2009). Survey of classification techniques in data mining. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 18–20.

[172] Pitzer, E. and Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. In *Recent Advances in Intelligent Engineering Systems*, pages 161–191. Springer.

[173] Pitzer, E., Affenzeller, M., Beham, A., and Wagner, S. (2011). Comprehensive and automatic fitness landscape analysis using heuristiclab. In *International Conference on Computer Aided Systems Theory*, pages 424–431. Springer.

[174] Poli, R. (2003). A simple but theoretically-motivated method to control bloat in genetic programming. In *European Conference on Genetic Programming*, pages 204–217. Springer.

[175] Poli, R., Langdon, W., and McPhee, N. (2008). A field guide to genetic programming (with contributions by jr koza)(2008). *Published via http://lulu. com*.

[176] Poli, R. and Langdon, W. B. (1998). Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252.

[177] Punch III, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P. D., and Enbody, R. J. (1993). Further research on feature selection and classification using genetic algorithms. In *ICGA*, pages 557–564.

[178] Qing-Shan, C., De-Fu, Z., Li-Jun, W., and Huo-Wang, C. (2007). A modified genetic programming for behavior scoring problem. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 535–539. IEEE.

[179] Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

[180] Ratle, A. and Sebag, M. (2001). Avoiding the bloat with stochastic grammar-based genetic programming. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 255–266. Springer.

[181] Raymer, M. L., Punch, W. F., Goodman, E. D., and Kuhn, L. A. (1996). Genetic programming for improved data mining: application to the biochemistry of protein interactions. In *Proceedings of the 1st annual conference on genetic programming*, pages 375–380. MIT Press.

[182] Rechenberg, I. (1984). The evolution strategy. a mathematical model of darwinian evolution. In *Synergetics—from microscopic to macroscopic order*, pages 122–132. Springer.

[183] Reinelt, G. (1991). Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384.

[184] Rouwhorst, S. and Engelbrecht, A. (2000). Searching the forest: Using decision trees as building blocks for evolutionary search in classification databases. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 633–638. IEEE.

[185] Ryan, C. and Azad, R. M. A. (2003). Sensible initialisation in grammatical evolution. In *GECCO*, pages 142–145.

[186] Ryan, C., Collins, J. J., and Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *European Conference on Genetic Programming*, pages 83–96. Springer.

[187] Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2015). Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Trans. Evolutionary Computation*, 19(3):309–325.

[188] Sætrom, P. (2004). Predicting the efficacy of short oligonucleotides in antisense and rnai experiments with boosted genetic programming. *Bioinformatics*, 20(17):3055–3063.

[189] Sætrom, P. and Hetland, M. L. (2003). Unsupervised temporal rule mining with genetic programming and specialized hardware. In *Proceedings of the 2003 International Conference on Machine Learning and Applications (ICMLA'03)*, pages 145–151.

[190] Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, 1(3):317–328.

[191] Sastry, K., Goldberg, D. E., and Kendall, G. (2014). Genetic algorithms. In *Search methodologies*, pages 93–117. Springer.

[192] Sen, S. and Clark, J. A. (2011). Evolutionary computation techniques for intrusion detection in mobile ad hoc networks. *Computer Networks*, 55(15):3441–3457.

[193] Sevaux, M., Sörensen, K., and Pillay, N. (2018). Adaptive and multilevel metaheuristics. *Handbook of Heuristics*, pages 1–19.

[194] Shao, L., Liu, L., and Li, X. (2014). Feature learning for image classification via multiobjective genetic programming. *IEEE Transactions on Neural Networks and Learning Systems*, 25(7):1359–1371.

[195] Shen, S., Sandham, W., Granat, M., Dempsey, M., and Patterson, J. (2003). A new approach to brain tumour diagnosis using fuzzy logic based genetic programming. In *Engineering in medicine and biology society, 2003. Proceedings of the 25th annual international conference of the IEEE*, volume 1, pages 870–873. IEEE.

[196] Sherrah, J., Bogner, R. E., and Bouzerdoum, B. (1996). Automatic selection of features for classification using genetic programming. In *Intelligent Information Systems, 1996., Australian and New Zealand Conference on*, pages 284–287. IEEE.

[197] Silva, S., Foster, J. A., Nicolau, M., Machado, P., and Giacobini, M. (2011). *Genetic Programming: 14th European Conference, EuroGP 2011, Torino, Italy, April 27-29, 2011, Proceedings*, volume 6621. Springer Science & Business Media.

[198] Silva, S. and Tseng, Y.-T. (2008). Classification of seafloor habitats using genetic programming. In *Workshops on Applications of Evolutionary Computation*, pages 315–324. Springer.

[199] Smit, S. K. and Eiben, A. (2010). Parameter tuning of evolutionary algorithms: Generalist vs. specialist. In *European Conference on the Applications of Evolutionary Computation*, pages 542–551. Springer.

[200] Smith, T., Husbands, P., and O'Shea, M. (2002). Fitness landscapes and evolvability. *Evolutionary computation*, 10(1):1–34.

[201] Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437.

[202] Song, A., Ciesielski, V., and Williams, H. E. (2002). Texture classifiers generated by genetic programming. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 1, pages 243–248. IEEE.

[203] Song, D., Heywood, M. I., and Zincir-Heywood, A. N. (2005). Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE transactions on evolutionary computation*, 9(3):225–239.

[204] Souffriau, W., Vansteenwegen, P., Berghe, G. V., and Van Oudheusden, D. (2008). Automated parameterisation of a metaheuristic for the orienteering problem. In *Adaptive and Multilevel Metaheuristics*, pages 255–269. Springer.

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

[206] Stadler, P. F. and Happel, R. (1992). Correlation structure of the landscape of the graph-bipartitioning-problem. *J. Phys. A: Math. Gen*, 25:3103–3110.

[207] Stadler, P. F. and Schnabl, W. (1992). The landscape of the traveling salesman problem. *Physics Letters A*, 161(4):337–344.

[208] Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9. Morgan Kaufmann Publishers.

[209] Tanev, I., Ray, T., and Buller, A. (2005). Automated evolutionary design, robustness, and adaptation of sidewinding locomotion of a simulated snake-like robot. *IEEE Transactions on Robotics*, 21(4):632–645.

[210] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1–6. IEEE.

[211] Tavares, J. and Pereira, F. B. (2012). Automatic design of ant algorithms with grammatical evolution. In *European Conference on Genetic Programming*, pages 206–217. Springer.

[212] Tavares, J., Pereira, F. B., and Costa, E. (2008). Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(3):604–616.

[213] Tewari, A. and Bartlett, P. L. (2007). On the consistency of multiclass classification methods. *Journal of Machine Learning Research*, 8(May):1007–1025.

[214] Van Belle, T. and Ackley, D. H. (2002). Uniform subtree mutation. In *European Conference on Genetic Programming*, pages 152–161. Springer.

[215] Veček, N., Mernik, M., Filipič, B., and Črepinšek, M. (2016). Parameter tuning with chess rating system (crs-tuning) for meta-heuristic algorithms. *Information Sciences*, 372:446–469.

[216] Vella, A., Corne, D., and Murphy, C. (2009). Hyper-heuristic decision tree induction. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 409–414. IEEE.

[217] Vladislavleva, E. J., Smits, G. F., and Den Hertog, D. (2009). Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349.

[218] Wager, T. D. and Nichols, T. E. (2003). Optimization of experimental design in fmri: a general framework using a genetic algorithm. *Neuroimage*, 18(2):293–309.

[219] Wang, P., Tsang, E. P., Weise, T., Tang, K., and Yao, X. (2010). Using gp to evolve decision rules for classification in financial data sets. In *Cognitive informatics (ICCI), 2010 9th IEEE international conference on*, pages 720–727. IEEE.

[220] Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological cybernetics*, 63(5):325–336.

[221] White, D. R. and Poulding, S. (2009). A rigorous evaluation of crossover and mutation in genetic programming. In *European Conference on Genetic Programming*, pages 220–231. Springer.

[222] Whitley, L. D. et al. (1989). The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, volume 89, pages 116–123. Fairfax, VA.

[223] Wilcox, R. R. (2011). *Introduction to robust estimation and hypothesis testing*. Academic press.

[224] Winkler, S., Affenzeller, M., and Wagner, S. (2007). Advanced genetic programming based machine learning. *Journal of Mathematical Modelling and Algorithms*, 6(3):455–480.

[225] Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.

[226] Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., and Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with java implementations.

[227] Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

[228] Wright, S. (1932). *The roles of mutation, inbreeding, crossbreeding, and selection in evolution*, volume 1. na.

[229] Wu, B., Abbott, T., Fishman, D., McMurray, W., Mor, G., Stone, K., Ward, D., Williams, K., and Zhao, H. (2003). Comparison of statistical methods for classification of ovarian cancer using mass spectrometry data. *Bioinformatics*, 19(13):1636–1643.

[230] Xu, C.-G. and Liu, K.-H. (2008). A gp based approach to the classification of multiclass microarray datasets. In *International Conference on Intelligent Computing*, pages 340–346. Springer.

[231] Ye, L. and Keogh, E. (2011). Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22(1-2):149–182.

[232] Zhang, L. and Nandi, A. K. (2007). Fault classification using genetic programming. *Mechanical Systems and Signal Processing*, 21(3):1273–1284.

[233] Zhang, M. and Ciesielski, V. (1999). Genetic programming for multiple class object detection. In *Australasian Joint Conference on Artificial Intelligence*, pages 180–192. Springer.

[234] Zhang, M. and Wong, P. (2008). Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines*, 9(3):229–255.