Comparison of a Genetic Algorithm to Grammatical Evolution for Automated Design of Genetic Programming Classification Algorithms

Thambo Nyathi, Nelishia Pillay

PII: S0957-4174(18)30175-1
DOI: 10.1016/j.eswa.2018.03.030
Reference: ESWA 11877

To appear in: *Expert Systems With Applications*

Received date: 29 October 2017
Revised date: 17 March 2018
Accepted date: 18 March 2018

Highlights

- Automated design of Genetic Programming classification algorithms is presented.
- Automated design uses a genetic algorithm and grammatical evolution.
- The approach is trained and tested using real-world binary and multi-class data.
- Grammatical evolution designed classifiers perform better for binary classification.
- Genetic algorithm designed classifiers perform better for multi-classification.

Comparison of a Genetic Algorithm to Grammatical Evolution for Automated Design of Genetic Programming Classification Algorithms

Thambo Nyathi[a,*], Nelishia Pillay[a,b]

[a]School of Mathematics, Statistics and Computer Science, University of KwaZulu-Natal,Pietermariztburg Campus, 4 Golf Road, Scottsville, Pietermariztburg, KwaZulu-Natal,3201, South Africa
[b]Department of Computer Science, University of Pretoria, Lynnwood Rd, Pretoria, 0002, South Africa

Abstract

Genetic Programming(GP) is gaining increased attention as an effective method for inducing classifiers for data classification. However, the manual design of a genetic programming classification algorithm is a non-trivial time consuming process. This research investigates the hypothesis that automating the design of a GP classification algorithm for data classification can still lead to the induction of effective classifiers and also reduce the design time. Two evolutionary algorithms, namely, a genetic algorithm (GA) and grammatical evolution (GE) are used to automate 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 are compared to each other and to manually designed GP classifiers on real-world problems. Furthermore, a comparison of the design times of automated design and manual design is also carried out for the same set of problems. The automated designed classifiers were found to outperform manually designed classifiers across problem domains. Automated design time is also found to be less than manual design time. This study revealed that for the considered datasets GE performs better for binary

*Corresponding author
Email addresses: `vuselani@gmail.com` (Thambo Nyathi), `pillayn32@ukzn.ac.za` (Nelishia Pillay)

classification while the GA does better for multiclass classification. Overall the results of the study are in support of the hypothesis.

Keywords: Genetic programming; Genetic algorithm; Grammatical evolution; Automated design; Classification.

1. Introduction

Data classification, is one of the most widely studied domains of research in machine learning. Many real-world tasks can be viewed as classification problems. Classification is the process of associating an object to a class (label) based on the features describing that object. Classification is generally performed by classifier models. Classification usually involves two phases a learning (training) phase and a testing phase. A classifier model is induced by a classification algorithm during training and its classification accuracy is evaluated during testing.

Evolutionary algorithms(EAs) are one of the methods that have gained prominence in the induction of classifiers, particularly genetic programming (Espejo et al. (2010)Freitas (2003)). Genetic programming(GP) is a population based algorithm that models Darwin's theory of evolution(Koza (1992)). For a number of reasons GP has proved to be effective in the induction of classifiers. The tree representation used by GP allows it flexibility to evolve classifiers that model numerous problems(Espejo et al. (2010)). For example GP can be configured to represent decision trees, association rules or discriminant functions.

GP like most EAs is a parameterized algorithm and it has been shown that the effectiveness of such algorithms depends on their configuration (Eiben and Smit (2011)). Algorithm configuration is a design process that involves determining numerical parameter values, selecting categorical parameters and setting the control flow that would lead to the algorithm finding an optimal (or near optimal) solution to the problem at hand. According to Kramer and Kacprzyk (2008) to manually configure an EA that yields effective results, considerable algorithm design experience is necessary. However Hutter (2009) argues that even with the necessary experience manual design is a tedious non-trivial task susceptible to human bias. Furthermore, the search space for possible parameter values is large and only a subset of the design decisions are considered during manual design. Parameter values and flow control options are considered using trial runs in an iterative trial and error approach. Montero and Riff (2014) argue that inexperienced designers add more parameters than are necessary during manual design leading to unnecessarily complex algorithms. They also point out that a lot of man hours are required for manual design and ideally the algorithm designer should have expert knowledge of the domain being considered, however this is not always possible. Parameter control and tuning methods have been proposed in literature (Dobslaw (2010); Eiben et al. (1999); Eiben and Smit (2011)) with no method being universally adopted (Karafotias et al. (2015)).

In a previous study (Nyathi and Pillay (2017)) we showed the effectiveness of using a genetic algorithm (GA)(Goldberg and Holland (1988)) to automate the design of a GP classification algorithm for data classification. In this study a comparison of the automated design of GP classification algorithms using a GA and grammatical evolution (GE) (Ryan et al. (1998)) is carried out. A GA and GE are used to automate the design of GP classification algorithms. GA and GE are individually used to search for a GP configuration that should result in GP evolving the best classifier for a specific classification problem. Throughout this paper the use of the word configuration refers to algorithm parameter values (numerical and/or categorical) and the algorithm control flow. The best classification accuracies of the classifiers evolved using the GA and GE algorithms are compared to each other and to those of manually designed classification algorithms for the considered problem instances. The study finds that automated designed classifiers significantly outperform manually designed classifiers for problems instances considered across domains and perform equivalently for a specific domain. In addition automated design time is shown to be less than manual design time. Hence, the contributions of this study are:

- The study investigates the feasibility of using a GA and GE for automating the design of GP classification algorithms for data classification and shows the effectiveness of automated design over manual design.

- The study compares the performance of using a GA to GE for automated design. It is shown that there is no significant difference in performance between the two EAs and either can be used for the design of GP classification algorithms. Although for the considered datasets GE is found to perform better on binary problems while the GA is found to be better on multiclass problems, the differences are not significant.

- The study also shows that the use of automated design leads to a reduction in man-hours for the design process.

This paper is structured as follows. Section 2 presents the background of the study outlining GP and the application of GP as a classification algorithm for data classification. The use of a GA and GE in automated design are also discussed in this section. Section 3 presents a brief overview of how GP, the GA and GE are related in the proposed approach. Section 4 outlines the manual design approach of GP classification algorithms, while Section 5 describes the automated design approach using the GA and GE implemented in this study. Experimental settings and a description of the experiments carried out are outlined in Section 6. Section 7 presents the results and analysis of the results. Finally Section 8 provides the conclusion of the study and discusses possible future work.

2. Background

2.1. Classification

Classification is a supervised machine learning method (Pappa and Freitas (2009)). In supervised learning the features with their corresponding class labels are provided and the training process entails learning the classes based on the features. During the training phase the classification algorithm has access to the class labels. After training the evolved classifiers should be able to generalize and assign unseen objects to their respective classes correctly, this is a measure of the predictive accuracy of the classifier Han et al. (2011). The procedure is to accept an input vector of features x and assign it to a specific class label y from a given number of classes. The number of correctly identified input instances divided by the total number of problem instances is used as a measure of how well the classifier performs and is known as the accuracy. A problem instance may have two classes in which case it is a binary classification problem or there may be more than two classes and then it is a multiclass classification problem. A number of methods used to induce classifiers have been proposed and for a comprehensive survey the reader is referred to Phyu (2009). In this study we consider the induction of classifiers using genetic programming.

2.2. Genetic programming

Genetic programming is an EA that generally encodes individuals using syntax trees. However, since Koza's original tree based GP, a number of research studies have proposed other forms of encoding GP individuals (Poli et al. (2008)). Generally a generational GP algorithm follows the following process. An initial population of GP individuals is randomly created and their individual fitness is evaluated using a predefined fitness function. A tree generation method is selected for initial population generation. Koza (1992) proposed three methods for initial population generation, these are the full method, grow method and the ramped half-and-half method. If the algorithm's termination condition is not met individuals of the current population are selected using a selection method, to act as parents for the generation of individuals (offspring) for the next population. Tournament selection is the most commonly used selection method for GP (Espejo et al. (2010)). Genetic operators (crossover and mutation) are applied to the parents to generate offspring. The GP crossover operator normally operates by exchanging randomly selected subtrees from each of two selected parents to create two offspring. The mutation operator is applied on a selected parent where a randomly generated subtree replaces a randomly selected subtree in the parent tree thus creating an offspring. The initial population is replaced by a population of offspring. In GP each genetic operator is used to create a specified number of offspring for the new population unlike in some evolutionary algorithms like GAs where a new population is evolved by first applying crossover and then mutation may be applied to each offspring (Eiben et al. (2003)). The iteration from one population to the next is known as a generation and during each generation the fitness of each individual in the population is evaluated. This process proceeds iteratively until a termination condition is met and the individual with the best fitness is returned. GP uses a set of predefined functions and terminals to construct individuals.

2.3. Data classification and GP

Generally when used as a classification algorithm GP uses supervised learning. GP firstly randomly initializes a population of classifiers and evaluates their fitness by applying each classifier to a given problem. These 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 to unseen instances (testing) of the given problem. Accuracy, which is obtained by dividing the number of correctly classified instances by the total number of instances is the most widely used fitness function to evaluate classifiers. Other fitness functions have been proposed based on the metrics obtained from the confusion matrix (Han et al. (2011)). One of the advantages of using GP as a classification algorithm is the tree representation used by GP individuals. Tree data structures are considered to be very general and therefore flexible. This flexibility allows them to be adaptable to a wide range of domains. Arithmetic, logical and decision trees are the most commonly evolved classifiers in GP (Espejo et al. (2010)). GP evolves one of three types of classifiers namely arithmetic classifiers, logical classifiers and decision tree classifiers. For the rest of this paper the term classifier is used in reference to one of the three mentioned types of GP evolved classifiers.

2.3.1. GP for binary classification

In binary classification two classes are involved and the objective is to assign objects as belonging to one class or the other. Archetti et al. (2006) used the arithmetic tree type to classify the bioavalability of a molecule using features of its chemical structure. The function set consisted of mathematical operators while the terminal set contained attributes from the dataset. A threshold value was predefined to discriminate between the two classes. If the numerical output from a classifier was less than the threshold value then the input was associated with one class or else the other class. The root mean square error (RMSE) of accuracy was used as the fitness function. The GP classification algorithm performed well when compared to support vector machines (SVM) and artificial neural networks. Johnson et al. (2000) used arithmetic trees to classify objects based on their spectral characteristics. The function set contained mathematical operators and the terminal set consisted of attributes from the dataset. As in the previous study RMSE of accuracy was used as the fitness function. The classification results obtained from applying GP were found to be better than those obtained from applying artificial neural networks and partial least squares(PLS) to the same dataset. Le-Khac et al. (2016) used arithmetic trees in the classification of financial data. The evolved classifiers were trained and tested on 2 datasets selected from the UC Irvine (UCI) machine learning repository (Blake and Merz (1998)). Each dataset contained financial records. The function set consisted of mathematical operators while the terminal set contained attributes from the dataset. Two fitness functions based on the confusion matrix were proposed. The proposed system was compared to six other classification algorithms and the GP algorithm performed better than some of the classifiers.

A study was presented by Loveard and Ciesielski (2001) that used logical trees to perform binary classification on problems from the medical domain. Three datasets were selected from the UCI repository to train and test the GP evolved classifiers. For each problem the function set contained logical operators and the terminal set had attributes from the dataset. Accuracy was used as the fitness function. Similarly Eggermont et al. (1999) used logical trees to classify data from 2 datasets selected from the UCI dataset repository. The authors proposed a weighted fitness function. The output of the evolved classifiers were classes. After training and testing the results from the proposed classification algorithms were compared to those obtained from applying 4 other classification algorithms to the considered datasets. When configured to model decision trees the GP algorithm function set consists of attributes from the dataset and the terminal set contains the classes. The classifier is evaluated from the root downwards in a top down process. Koza was the first to propose the induction of decision trees using GP (Koza (1991)). Khoshgoftaar et al. (2003) proposed a GP classification algorithm to evolve decision trees that classified software quality. An average of the misclassification error and tree size was used as the fitness function. The function set contained attributes from the dataset while the terminal set consisted of two classes. The proposed system was compared to a standard GP system and the decision tree evolving algorithm was found to perform as well as the standard GP.

2.3.2. GP for multiclass classification

Multiclass classification involves categorizing more than two classes. Munoz et al. (2015) consider multiclass classification using GP a more complex task than binary classification. They argue that GP does not perform very well when compared to other state of the art methods for multiclass classification. However a number of multiclass methods using GP have been proposed. Kishore et al. (2000) presented an approach which converted an n class problem into n binary classification problems. The system was run once for each class while the other *n-1* classes were grouped together. A mixture of arithmetic and logical trees was used. This is generally referred to as binary decomposition. Bojarczuk et al. (2000) used arithmetic trees to apply binary decomposition in the classification of pathogens. Muni et al. (2004) proposed an approach where an individual consisted of the same number of arithmetic trees as the number of classes, so for a 3 class problem an individual consisted of 3 arithmetic trees. Zhang and Ciesielski (1999) introduced the idea of numerical boundaries representing classes. A numerical range is predefined for each class for example class 0 = [<0], class 1 = [0 - 5] and class 3 = [>5]. If for example a numerical output value from a classifier is 3, then the object is in class 1. The approach is generally known as the static class boundary determination (SCBD) method. Variants of this method were later proposed where the ranges are dynamically determined during the evolution process(Zhang and Smart (2004)). Loveard and Ciesielski (2001) and Espejo et al. (2010) present a survey of using GP as a classification algorithm.

2.4. Genetic algorithms

A GA is a stochastic algorithm used to solve optimization and search problems. Individuals in a GA population represent possible solutions to a problem being solved. Each individual is usually encoded as a linear genome structure. A basic genetic algorithm first randomly creates an initial population of individuals. The fitness of each individual in the population is evaluated. A selection method is used to select individuals from the current population to act as parents and undergo crossover and mutation to create offspring for the next generation. The complete population may be replaced by a new population of offspring or the least fit individuals in the current population may be replaced by the offspring. This process continues iteratively until a predefined stopping criterion is met and the algorithm terminates. The best solution based on fitness is returned. Goldberg (2006) provides a detailed presentation of genetic algorithms.

2.5. Grammatical evolution

Grammatical Evolution (GE) is a grammar based form of GP (Poli et al. (2008)). Grammatical evolution was first proposed by Ryan et al. (1998). It uses a grammar to map a GE genotype to 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. GE draws inspiration from molecular biology where a sequence of genetic material, deoxyribonucleic acid (DNA)(genotype) is translated into a protein which defines characteristics of a phenotype. A more detailed comparative analysis of GE and the biological process is outlined by Ryan et al. (1998). An individual in GE is encoded as a chromosome of variable length binary strings. Each gene of the chromosome is an 8 bit binary string and is referred to as a codon. 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:

$$Rule = (codon \quad decimal \quad value)\%(N^o of \quad production \quad rules) \tag{1}$$

A derivation tree (phenotype) is evolved by iterating and mapping through the sequence of codons. 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.

The next section presents a brief overview of work that uses GAs and GE for automated design.

2.6. Automated design using GAs and GE

Applying metaheuristics such as evolutionary algorithms to a problem domain requires various design decisions which are usually done manually and are quite time consuming. Design decisions include:

1. *Determining the parameters for an approach*, e.g. population size, genetic operator probabilities in evolutionary algorithms (Eiben et al. (1999)).
2. *Determining the operators to use*, e.g uniform crossover, one-point crossover and two-point crossover in genetic algorithms (Karafotias et al. (2015)).

3. *Determining the control flow of an approach*, this involves determining the order in which processes of the approach should occur, e.g crossover followed by mutation (Dobslaw (2010)).
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 (Burke et al. (2010)).
5. *Creating new operators*, e.g mutation operator in an evolutionary algorithm or the shaking process for variable neighborhood search (Hong et al. (2013)).

These decisions usually require expert knowledge and hence cannot be made by a researcher who would like to apply a metaheuristic to a particular application domain. For these reasons there has been a large scale initiative to automate these design decisions (Hutter et al. (2007); Ansótegui et al. (2009); López-Ibánez and Stutzle (2012)). The aim is two fold. Firstly automating the design process will relieve the researcher from this laborious task. Secondly, this will provide a researcher without expert knowledge with an off-the-shelf tool which can be applied to any application domain. This paper focuses on the automated design of genetic programming to evolve classifiers.

To the best of our knowledge currently there are no methods that automate the design of genetic programming classification algorithms. However there have been proposals to automate the design of classification pipelines (Olson et al. (2016a,b); Pappa (2017)). A pipeline is a sequence of processes or steps that are required to perform classification. For example the following steps can be considered as a pipeline: preprocessing the dataset, feature selection, classification algorithm selection, parameterization and data post processing. The research presented in this paper is different from pipeline automation in that the automation of the design of the classification algorithm that induces classifiers is proposed while the automation of pipelines proposes to automate the selection of classification algorithms from a pool of given algorithms. Hence the presented overview focuses on the studies that use genetic algorithms and grammatical evolution to design other metaheuristics. The design decisions taken or made resulted in a configuration for a metaheuristic.

Souffriau et al. (2008) used a steady-state genetic algorithm to evolve configurations for an Ant Colony Optimization (ACO) metaheuristic to solve the orienteering problem, a form of the traveling salesman problem. In this study only design decision 1 (parameter values) listed above was considered for the configurations. The chromosome was composed of real numbers. Each gene of the chromosome represented a specific parameter of the ACO metaheuristic. Stochastic universal selection was used with uniform crossover and random bit mutation. The GA was used to search for parameter values that yielded the best ACO solution to solve the problem at hand. The effectiveness of each ACO was measured using a fitness function. The best parameters evolved from training were compared to manually tuned ACO algorithms by applying them to unseen instances of the orienteering problem. ACO algorithms configured with GA evolved configurations performed better than those manually tuned. Diosan and Oltean (2007) used a GA to evolve configurations for evolutionary algorithms. The configurations automated design decisions 1, 2 and 3. Uniform crossover was used with gaussian mutation and tournament selection. The best EAs from the configurations were tested on function optimization problems and the results were competitive when compared to other approaches to solve function optimization methods.

Tavares and Pereira (2012) presented an approach which used a GE framework to evolve Ant Colony Optimization(ACO) configurations. The GE grammar specified design decisions 1, 2 and 3 for ACO algorithms. Single point crossover, integer flip mutation, elitism and tournament selection were used. To evaluate the effectiveness of the proposed framework the automatically configured ACO algorithms were trained and tested on the Traveling Salesperson Problem. The results were compared to human designed ACO algorithms. The presented results indicated that the automatically configured ACO algorithms performed better than human manual designed algorithms. Lourenço et al.

(2012) proposed the use of grammatical evolution for the evolution of evolutionary algorithms for the Royal Roads Functions. The grammar was used to automatically design an EA i.e. it specified design decisions 1, 2 and 3. Single point crossover, bit flip mutation and tournament selection were used. The evolved EAs were trained and tested on Royal Roads Functions and they were found to perform the same as manually designed EAs. Lourenço et al. (2013) presented a hyper-heuristic framework for the evolution of EAs to solve the knapsack problem. The framework used GE to design the EAs automating design decision 1, 2, and 3 for each EA. Single point crossover, bit flip mutation and tournament selection were used. The evolved EAs were found to perform well on unseen problem instances when compared to other methods. Drake et al. (2013) also presented a hyper-heuristic framework which uses GE to design a local search method, namely the Variable Neighborhood Search (VNS). The GE grammar automated design decision 1, 2, 3, 4 and 5 for the VNS. Crossover and mutation were used with tournament selection. The GE evolved VNS was tested on instances of the Vehicle Routing Problem.

3. Overview of the proposed approach

In the proposed approach an evolutionary algorithm, in this case a GA(or GE), is used to make design decisions and evaluate different GP designs. The GA(GE) simulates an algorithm designer as it searches through the GP design space for the best GP configuration. Each member of a GA(GE) population encodes a GP configuration and a set of classification problems is used to evaluate each GP configuration. The evaluation is carried out by using the GA(GE) evolved GP configuration to configure GP which is then applied to a given classification problem. There is no link between the GA and GE, each algorithm is used to configure GP classification algorithms as illustrated in Figure 1.

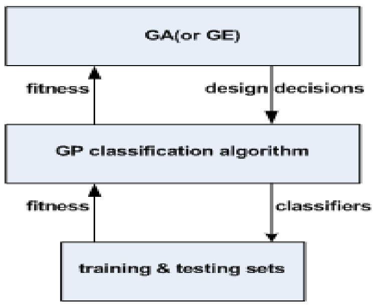


Figure 1: Automated design overview

The following sections 4 and 5 describe the manual GP design and automated GP design approaches used in this study.

4. Manual GP

This section outlines the manual design of the generational Genetic Programming classification algorithm used in this study. Each individual is a classifier induced by the GP algorithm. Each individual can be one of three types of classifiers either an arithmetic tree classifier, logical tree classifier or decision tree classifier. The type of classifier is determined by the contents of the function and terminal sets.

Algorithm 1 .Generational Genetic Programming Algorithm

```
1: Create initial population
2:   while termination condition not met do
3:     Calculate fitness of all individuals
4:     Select fitter individuals for reproduction
5:     Apply genetic operators to selected individuals
6:     Replace the current population with the offspring
7:   end while
8: return best individual
```

Using the algorithm flow presented in Algorithm 1 the following sections describe the processes involved in the manual GP algorithm.

4.1. Function set and terminal set

A function set and a terminal set need to be pre-defined to enable individuals to be constructed by the GP algorithm. Therefore for each tree type a function set and terminal set need to be defined. The function sets are defined as follows; arithmetic tree = { +,-,*,/(protected) }, logical tree = {AND,OR,EQUAL,DIFFERENT,NOT} and decision tree ={ attributes from fitness cases }. Similarly terminal sets were defined as follows; arithmetic and logical trees = { attributes from fitness cases } and decision trees = {class 0,class 1} for binary class problems while the number of classes increase to a value determined by the number of classes in a multiclass problem. Static class boundary determination is used for multi-classification problems with the boundaries set as follows:

a) three classes: Class1[-inf,-1], Class2[-1,1], Class3[1,inf]
b) four classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,inf]
c) five classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,inf]
d) six classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,8], C6[8,inf]
e) seven classes: Class1[-inf,-1], Class2[-1,1], Class3[1,2], Class4[2,4], C5[4,8], C6[8,10] C7[10,inf]
f) 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]

4.2. Fitness function

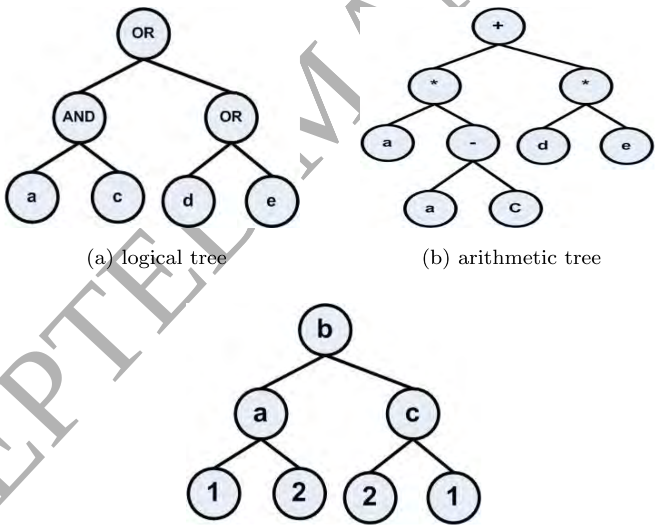
In this study predictive accuracy is used as the fitness function. The predictive accuracy of each individual is evaluated as the sum of all correctly classified fitness cases divided by the total number of fitness cases. The fitness is assigned as a percentage.

4.3. Stopping criteria

A stopping criteria needs to be defined for a GP algorithm. 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.

4.4. 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 4.1. The function set and terminal set used corresponds to the tree type chosen for the individuals i.e arithmetic trees, logical trees or decision trees. The ramped half-and-half method is used for initial tree generation and the fitness of each individual in the population is evaluated. Figure 2 outlines examples of a (a) logical GP tree (b) arithmetic GP tree and (c) decision tree.



(a) logical tree (b) arithmetic tree

(c) decision tree

Figure 2: Example of GP individuals

4.5. 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. 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.

4.6. Application of genetic operators

In this study two genetic operators, namely crossover and mutation are used to create offspring. GP creates offspring either by crossover or mutation unlike in a GA where offspring are first created by crossover and then mutated. GP requires that an application rate be specified for each operator and this determines the number of offspring evolved by that operator. For example if the population size is set as 200 and the crossover rate is 80% and mutation is 20% then crossover will create 160 offspring while mutation creates 40 offspring.

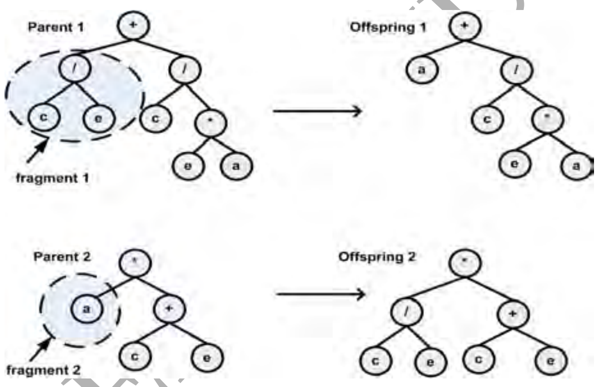


Figure 3: crossover operation

4.6.1. Crossover

Crossover uses tournament selection to select two parents. In each of the parents a node is randomly selected and this becomes the crossover point. The subtrees, referred to as crossover fragments, rooted at these crossover points are exchanged thereby creating two new offspring. The offspring are added to the new population. A crossover point is randomly selected in parent 1 resulting in fragment 1. Similarly in parent 2 random selection of a crossover point results in fragment 2. The two fragments are exchanged resulting in offspring 1 and offspring 2. The depth of offspring produced by the crossover operation must not exceed a stipulated offspring depth limit, if they do exceed the depth limit the trees are pruned. Figure 3 illustrates the single point crossover process used in this study.

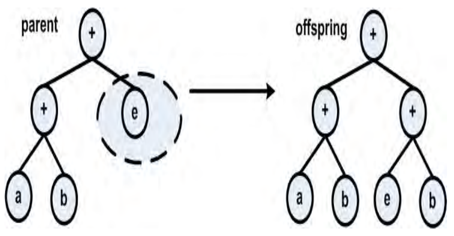


Figure 4: mutation operation

4.6.2. Mutation

Grow mutation is the mutation operator used during regeneration for this implementation. Tournament selection is used to select a parent from the current population to undergo mutation. Grow mutation randomly chooses a terminal node and replaces it with a randomly generated subtree. A maximum mutation depth parameter is predefined to limit the size of the subtrees produced by the mutation operator. If this parameter is exceeded the subtree is pruned until it meets the required size. After the mutation operation the produced offspring is added to the new population. Figure 4 illustrates the mutation process.

4.6.3. Termination

The process is repeated from generation to generation until one of the stopping criteria outlined in section 4.3 is met. The classifier with the best training accuracy across all generations is returned and applied to the test data and the accuracy result is returned.

5. Proposed automated design

In this section the proposed automated design approach is presented. The design decisions, together with their possible values considered in this study are outlined. A detailed description of the proposed automated design approaches,using a GA and GE are also presented in this section.

5.1. GP design decisions

To design a GP classification algorithm a number of design decisions have to be made. The following is an outline of the design decisions considered in this study.

5.1.1. Determination of parameter values

A GP classification algorithm contains both categorical and numerical parameters that need to be specified. The parameters and possible values that can be assigned to each design decision are outlined as follows:

1. Categorical parameters
   (i) *Tree type* The three commonly used tree types by GP for data classification algorithms are arithmetic trees, logical trees and decision trees (Banzhaf et al. (1998)). These are the three options available for this design decision.
   (ii) *Initial population generation method*
   Initial population generation will be performed using either the full method, grow method or ramped half-and-half method. These are the three methods available for this design decision(Poli et al. (2008)).
   (iii) *Fitness function*
   Accuracy is the most commonly used fitness function to evaluate the quality of classifiers. There have been proposals in literature (Le-Khac et al. (2016)) for fitness functions to use more than one criteria to evaluate classifiers. Five fitness functions are available for this design decision. These are based on the confusion matrix and are outlined as follows:
   (a) *Accuracy*
   Accuracy is the rate of correctly classified instances and is de-

fined by the following equation:

$$accuracy = \frac{tp + tn}{tp + tn + fp + fn} \quad (2)$$

where $tp(tn)$ refers to true positives(true negatives) and $fp(fn)$ false positives (false negatives).

(b) *F-measure*

F-measure is a widely used metric for evaluating classifiers (Espejo et al. (2010)). This metric is evaluated using $sensitivity = tp/(fn + tp)$ and $specificity{=}tn/(fp + tn)$ (Hand (1997)) and is given by the following equation:

$$f = 2 * (\frac{sensitivity * specificity}{sensitivity + specificity}) \quad (3)$$

(c) *Weighted accuracy*

A number of studies have proposed weighted fitness functions. Bhowan et al. (2010) used a fitness function that sums the true positive rate (50%) and the true negative rate (50%). For this fitness function a similar approach is adopted where accuracy and *f-measure* are weighted and defined as follows:

$$weighted_{acc} = 0.5 * accuracy + 0.5 * f\text{-}measure \quad (4)$$

(d) *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:

$$weighted_{rand} = rand * accuracy + (1 - rand) * f\text{-}measure \quad (5)$$

(e) *True positive rate*

This is the rate of correctly classified positive instances of the problem.

$$tpr = \frac{tp}{tp + fn} \quad (6)$$

For multiclass problems accuracy is used as the fitness function.

(iv) *Selection method*

From the survey of literature (Poli et al. (2008),Banzhaf et al. (1998)) the two commonly used selection methods for GP are, fitness proportionate selection and tournament selection.

(a) *Fitness proportionate*

In fitness proportionate individuals are selected with a probability that is directly proportional to their fitness values (Sastry et al. (2014)).

The selection of an individual proceeds as follows;

i) evaluate the probability, p$_i$ of selecting each individual in the population:

$$p_i = \frac{f_i}{\sum_{k=1}^{n} f_k} \tag{7}$$

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:

$$q_i = \sum_{k=1}^{i} p_k \tag{8}$$

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 to be a parent.

(b) *Tournament selection*

Tournament selection is implemented as outlined in section 4.5.

2. Numerical parameters

(a) *Population size*

Research has been carried out on data classification problems using a wide range of population sizes of GP individuals such as 50 in (Bandar et al. (1999)), 100 in (Olson et al. (2016b)), 200 in (Papagelis and Kalles (2001)), 300 in (Moore and White (2007)), 400 in (Macedo et al. (2016)), 500 in (Basgalupp et al. (2009)) and up to as high as 5000 in (Aitkenhead (2008)). Three population size values 100, 200 and 300 will be available for this parameter. These values are deemed to adequately represent the search space.

(b) *Maximum initial tree depth*

Maximum initial tree depth is used to limit the size of initial trees. Garcia et al. (2008) specified a maximum initial tree depth of 17 for predicting protein networks while Aitkenhead (2008) set this parameter to be 2 for evolving decision trees. 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.

(c) *Tournament size*

The tournament selection method requires a tournament size value. Le-Khac et al. (2016) used a tournament size of 3, while Muni et al. (2006) used a tournament size of 10. A number of studies in the literature (Jabeen and Baig (2011); Teredesai and Govindaraju (2004); Sakprasat and Sinclair (2007)) used values in the range [2-10]. Tournament size is randomly assigned any value in the range [2-10].

(d) *Maximum offspring depth*

Offspring created by both crossover and mutation can grow exponentially and a maximum size restriction has to be specified. This parameter is known as maximum offspring depth. Li et al. (2005) used a value of 17 for maximum offspring depth while Zhao (2007) set the value to 30 in their approach. We specify the range [2-15] for arithmetic and logical trees and [2-8] for decision trees.

(e) *Mutation depth*

The mutation operator has a mutation depth parameter which is used to control the size of the subtree created by the mutation operator. Wieloch (2013) used a value of 5. The value for the mutation depth will be any value in the range [2-6].

(f) *Termination*

Two termination criteria are defined, desired fitness and maximum number of generations. If the desired outcome is met i.e 100% accuracy or maximum number of generations, the algorithm will terminate. Muni et al. (2006) used a value of 30 generations, Tsakonas (2006) and Olson et al. (2016b) used 100 generations while Le-Khac et al. (2016) set the maximum number of generations to 1000. We specify the range [50-200] for setting the value of the maximum number of generations.

5.1.2. Determination of genetic operators

Genetic operators are used to evolve the next generation of individuals from the current population. The most commonly used genetic operators by GP for data classification are crossover and mutation (Espejo et al. (2010)). Three genetic operators namely, crossover, mutation and creation are presented for this design decision.

1. *Crossover*

   This crossover operator follows the same process as described in section 4.6.1.

2. *Mutation*

   Grow mutation and shrink mutation are commonly used by GP for classification (Jabeen and Baig (2010)). For this design decision grow mutation and shrink mutation are the mutation operator options implemented. Shrink mutation replaces a subtree with a randomly selected terminal (Angeline (1996)). Grow mutation is implemented in same manner as presented in section 4.6.2.

3. *Creation*

   This operator replaces 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 combinations are presented as follows:

(a) *Crossover and mutation*

   Usually a higher crossover application rate is recommended and values found in the literature (Tsakonas and Dounias (2002), Le-Khac et al. (2016)) for the crossover application rate range from 60% to 90%. Johansson and Niklasson (2009) used a mutation rate of 1% to evolve decision trees while Tsakonas (2006) applied shrink mutation at a rate of 60%. For this combination the application rates are randomly obtained 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. Estrada-Gil et al. (2007) used 0% mutation in classifying medical data. 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.

5.1.3. Determination of control flow

Control flow

This design decision determines the order in which processes of the algorithm occur. After initial population generation a combination of genetic operators outlined in section 5.1.2 are randomly chosen to evolve the next generation. One of two options will be 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 termination of the algorithm. If the selection is random a different combination of genetic operator is randomly chosen to evolve each subsequent generation until algorithm termination.

The next section presents the Genetic Algorithm used to automate the design of a Genetic Programming classification algorithm. The term configuration in this study is used to refer to a set of values for all the design decisions of the GP classification algorithm.

5.2. GA for automated design

A GA individual is a GP classification algorithm configuration. The GA searches for a configuration that will yield the best classifier. GA individuals undergo crossover and mutation, producing new GP configurations. The fitness of each GA individual is evaluated by applying the configuration on a GP classification problem and the returned classification accuracy is the fitness of the GA individual. This process is repeated from generation to generation until the maximum number of specified generations is met. The generational GA used is outlined in Algorithm 2.

Algorithm 2 .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
```

Each of the processes depicted in Algorithm 2 are described in the following sections. Table 1 summarizes the options for the different design decisions for the GP classification algorithm described in the previous section.

#	**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	50 - 200 (multiclass - 50,100,200)

Table 1: Design decisions and range of values

5.2.1. Initial population generation

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 listed in Table 1. Figure 5 outlines the structure of a GA individual. A chromosome is made up of 14 genes, labeled from g_0 to g_{13}.

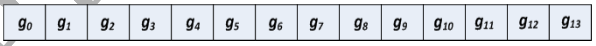


Figure 5: GA chromosome

Gene values are encoded as integer values. Integer values are used to encode design options for a design decision. The first gene of the chromosome g_0 represents the type of tree to be used i.e arithmetic tree, logical tree or decision tree. The second gene g_1 represents the population size parameter while g_2 represents the tree generation method. Maximum tree depth is represented by g_3, offspring tree depth by g_4, selection method by g_5, tournament size by g_6, reproduction rates by g_7, mutation type by g_8, mutation depth by g_9, genetic operator combinations by g_{10}, control flow by g_{11}, fitness function by g_{12} and g_{13} number of generations.

The first step of a GA algorithm is the initial population generation. A fixed number of chromosomes are randomly created. Each gene of a chromosome is assigned a value randomly chosen in the range listed in Table 1. For example g_0 can be randomly assigned a value of 1, from Table 1 the value 1 represents logical trees for that design decision. If g_1 is assigned a value of 100 then the population size for GP is 100 individuals. This random assignment occurs for all the genes from g_0 to g_{13} for all the chromosomes.

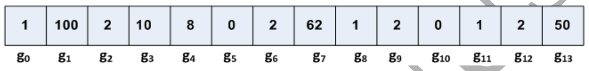


Figure 6: GA individual

Figure 6 is an example of a typical GA individual for this application.

5.2.2. *Fitness evaluation and selection*

The fitness of each GA 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 GA 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 GA configuration. Fitness proportionate is used to select parents as described in section 5.1.1.

5.2.3. *Elitism*

Elitism 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.

5.2.4. Crossover

Uniform crossover is implemented in this study. This method uses the crossover rate to consider two selected individuals for crossover. During the crossover process genes are considered individually based on a probability known as a swapping probability (Sastry et al. (2014)). 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 between 0 and 1 is chosen. 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 are assigned values. The offspring are then added to the new population. If crossover does not take place as determined by the crossover rate the two selected parents are added to the next population.

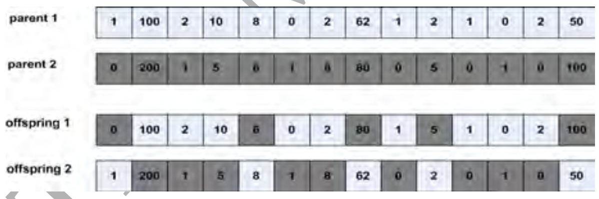


Figure 7: GA crossover

Figure 7 is an illustration of the uniform crossover process, parent 1 and parent 2 are crossed over resulting in offspring 1 and offspring 2. The values of genes 1,2,3,5,6,8,10,11 and 12 of offspring 1 are obtained from parent 1 while values for genes 0,4,7,9 and 13 are obtained from parent 2.

5.2.5. Mutation

The mutation operator considers each gene of a given individual for mutation based on the mutation rate (Eiben et al. (2003)). For each gene of an individual a random number r between 0 and 1 is chosen. If the mutation rate is equal or greater than r that gene is mutated by randomly assigning it a value from the range of possible values stipulated in Table 1 for that gene. If the mutation rate is less than r that gene is not mutated and the process moves to consider the next gene of the individual. This process proceeds in this manner for all genes of an individual. After application of the mutation operator the individual is added to the new population. The fitness of all the individuals in the new population are then evaluated.

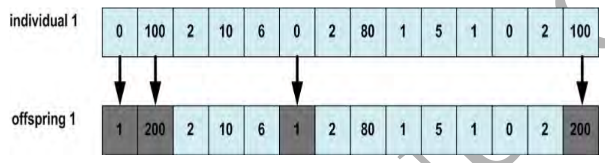


Figure 8: GA mutation

Figure 8 illustrates the mutation operation applied to individual 1. Genes 0,1,5 and 13 have been mutated.

5.2.6. Termination

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

5.3. GE for automated design

This section outlines the Grammatical Evolution approach used to automate the design of GP classification algorithms.

In this study GE is used to determine the 1) genetic operators 2) parameter values and 3) control flow for GP. The generational GE algorithm used is outlined in Algorithm 3.

Algorithm 3 .Grammatical Evolution

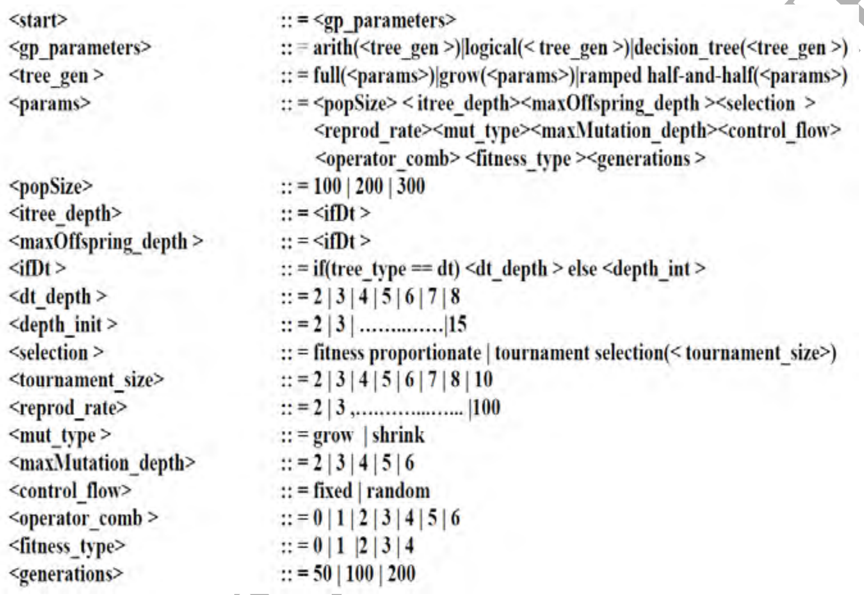
```
 1: Create an initial population of variable length binary strings
 2: Map via a BNF grammar
        a) binary strings to expression using production rules
 3: Evaluate fitness
 4: do while {termination condition not met}
 5:     Select fitter individuals for reproduction
 6:     Recombine selected 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
```

5.3.1. Initial population generation

Each element of the population is a variable length chromosome of codons. Each codon is an 8 bit binary string. Individuals of the initial population are generated by randomly creating codons. The length of each individual of the initial population lies in the range 14-16 codons and is randomly created during initial population generation. The number of individuals created for the initial population is specified by the population size parameter.

5.3.2. Mapping

Figure 9 depicts the grammar used. 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 GE individual is mapped to a GE phenotype which represents a GP configuration. To map an individual the binary strings are converted to integer values. Mapping starts from the first leftmost integer value (codon) of the GE chromosome. Using equation 1 the modulus of each codon is evaluated and used to select appropriate production rules. If the selected production rule contains non-terminals the leftmost non-terminal is expanded first. If there is only one production rule a codon value is not used and the rule replaces the non-terminal. This process continues until all non-terminals are converted to terminals.



```
<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
```

Figure 9: Grammar- automated design

Given the following codon sequence 10,215,30,48,7,8,220,30,40,73,5,11,21,112,32 a typical mapping process is presented 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.

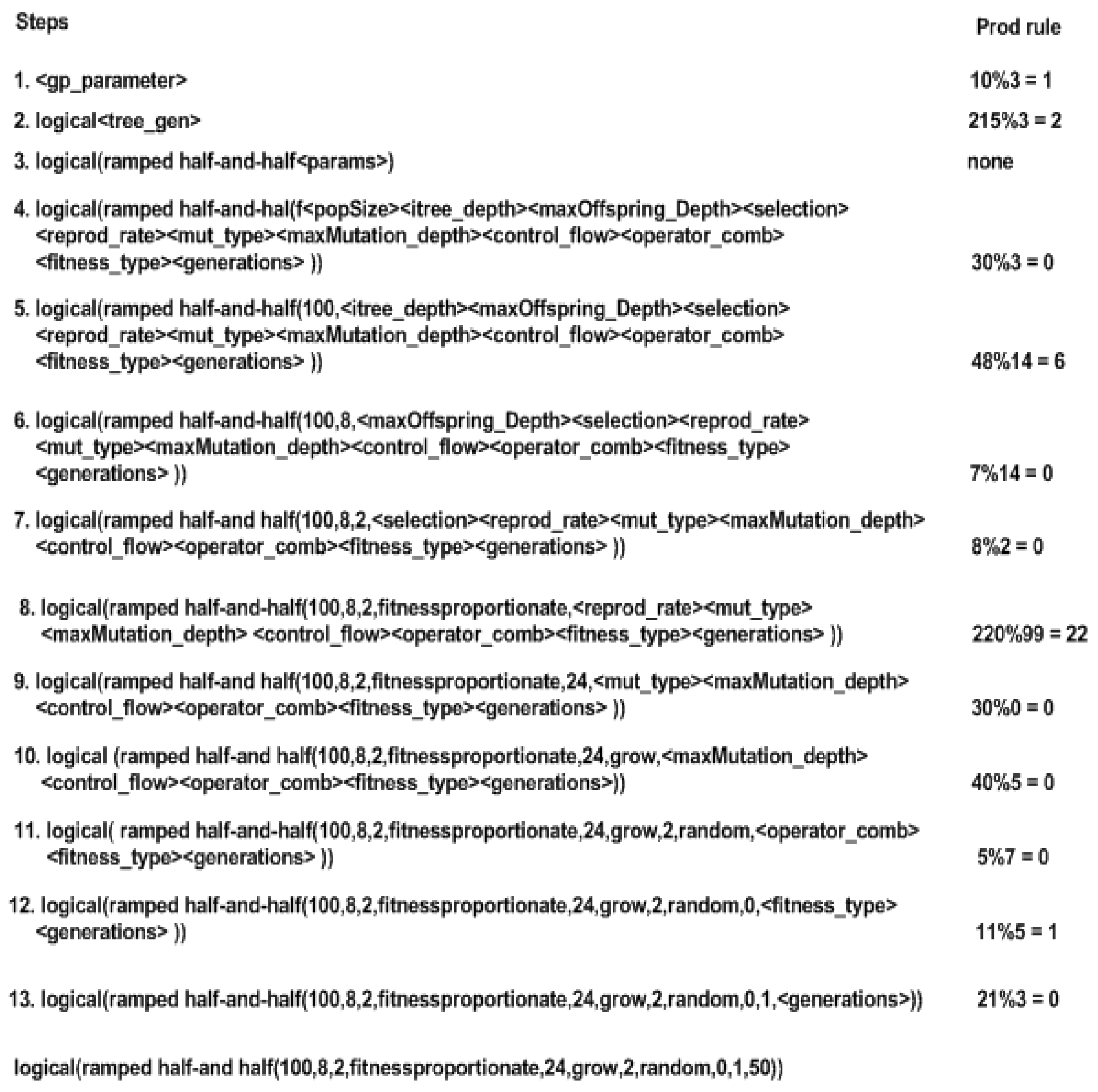
logical (ramped half-and-half (<popSize>,<itree_ depth>,< maxOffspring_ depth>, <selection>, <sel_ size>,<reprod_ rates>, <mut_ type>,<maxMutation_ depth>, <reprod_ seq>, <operator_ pool>, <fitnes_ type>, <generations>))

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 10 illustrates the complete mapping process. The left side of Figure 10 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. 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 GE phenotype is then interpreted into a GP configuration as listed in Table 2.



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))	

Figure 10: Genotype-phenotype mapping

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 2: GE evolved GP parameter settings

5.3.3. Fitness evaluation

The fitness of each individual is evaluated by using the phenotype to configure a GP classification algorithm and applying it to a classification problem. Training and testing are carried out in the same way as described for the GA in section 5.2.2. Accuracy for the test set is used as the fitness for the GE genotype.

5.3.4. Selection

Tournament selection is used to select parents as described in section 4.5.

5.3.5. Elitism

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

5.3.6. Crossover

Two parents are selected using tournament selection. A crossover probability rate is used to determine if the crossover operation should be applied. If crossover is to take place recombination of the selected parents is performed using single point crossover (Holland (1992)). In single point crossover a random number r in the range [1, l-1](where l is the length of the chromosome) is chosen to be the crossover point. The genes before the crossover point are collectively referred to as the head of the parent and those after the tail. Since variable length chromosomes are used, in this case l is the length of the shortest of the two selected parents. The parents are split at the crossover point and they exchange tails. Figure 11 illustrates single point crossover. Parent 1 is an individual consisting of 14 codons while parent 2 consists of 15 codons.

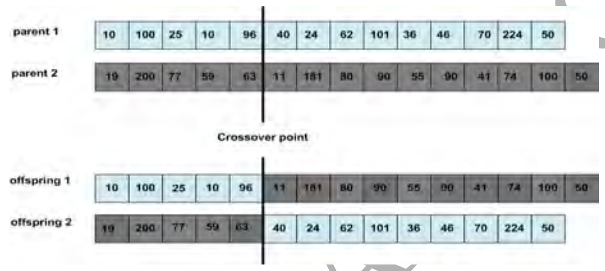


Figure 11: single point crossover

The randomly selected crossover point is indicated by the bold vertical line. The two parents exchange tails resulting in offspring 1 and offspring 2. Although Figure 11 shows each codon as an integer value the crossover operator can be applied to either the binary strings or integer values. For this study crossover is applied to binary strings. If crossover does not take place as determined by the crossover probability rate the two selected parents are added to the next population.

5.3.7. *Mutation*

Bit mutation is applied to the offspring produced by the crossover operator (Eiben et al. (2003)). Each bit of the binary string codon is considered for mutation based on the mutation probability. For each bit of each binary string in the chromosome a random number between 0 and 1 is chosen. If the selected number is less that the mutation probability the bit is flipped i.e. if the bit has a value of 1 it is changed to 0 and if has a value of 0 it is changed to 1. After mutation the chromosome is added to the new population.

5.3.8. Termination

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

6. Experimental settings

This section describes the experimental setup used to evaluate the automated design approach. Firstly the data used in the experiments is presented followed by a description of the experiments carried out. The performance of classifiers evolved by manual design and automated design are compared using binary class and multiclass classification problems.

6.1. Datasets

Well known publicly available datasets with a varied number of attributes and records are selected from the UCI machine learning repository. The selected datasets contain real-world data from various domains this, will allow the evaluation of the proposed methods ability to generalise across domains. Each dataset is randomly split into a 70% training set and 30% test set. Additionally the effectiveness of the proposed approach tailored to a specific domain namely intrusion detection is evaluated using the NSL-KDD 99+20% dataset (Tavallaee et al. (2009)).

- Binary datasets

  Eleven binary datasets are considered and these are listed in Table 3.

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: Summary of binary datasets

- Multiclass datasets

  840 Table 4 contains a listing of the 11 multiclass datasets selected.

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 4: Summary of multiclass datasets

- NSL-KDD dataset

  A set of 6 datasets are created from the NSL-KDD 99+20% computer security dataset. 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) remote to local (r2l). The set of 6 datasets is created by grouping the records based on their classes (Chareka and Pillay (2016)). 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.

6.2. Manual design of GP

Experiments are conducted on binary and multiclass problems these are outlined as follows.

6.2.1. Binary class experiments

For each dataset three experiments, experiment 1, 2 and 3 are conducted using manually designed GP classification algorithms. Tree type is used to distinguish between the experiments. Experiment 1 uses arithmetic trees, experiment 2 uses logical trees and experiment 3 uses decision trees. For each experiment parameter tuning is carried out using trial runs. In a similar approach to that carried out by Rouwhorst and Engelbrecht (2000) and Ma and Wang (2009) parameter tuning is carried out for each dataset listed in Table 3. The tuning process is performed using the commonly used values for GP classification algorithms outlined in section 5 as the starting point. 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. From parameter tuning a population size of 300 is 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 is found to be adequate to achieve algorithm convergence on the considered datasets. This value is set as the maximum generation for all manual experiments. The initial tree depth parameter is tuned for values in the range from 2 to 20 while the maximum offspring depth parameter range is from 4 to 30 for arithmetic and logical trees and 4 to 10 for decision trees. The selection size range of 2 to 20 is used. It is recommended that the crossover application rates should be higher than the mutation application rate (Koza (1992)). 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 is tuned in the range 2 to 10. For the three manual experiments tournament selection is used as the selection method, which is the most common selection method used for GP (Blickle and Thiele (1996); Espejo et al. (2010)). Initial tree generation is performed using the ramped half-and-half method (Garcia-Almanza and Tsang (2006); Johansson and Niklasson (2009)) and grow mutation (Zhao (2007); Ma and Wang (2009))is used as the mutation operator. Predictive accuracy is used as the fitness function.

The tuned parameter values for each dataset are presented in Table 5, Table 6 and Table 7. 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*.

- Experiment 1 (Arithmetic trees)
  Experiment 1 uses the following mathematical operators in the function set ={ +,-,*, /(protected)} and attributes from the dataset are used for the terminal set.

- Experiment 2 (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	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 5: Arithmetic parameter values

Experiment 2 uses the following logical operators in the function set ={AND,OR,EQUAL,DIFFERENT,NOT} and attributes from the dataset are used for the terminal set.

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 6: Logical parameter values

905

- Experiment 3 (Decision trees)
  The function set of experiment 3 consists of attributes from the dataset while the terminal set constitutes of classes(class 0 and class 1).

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 7: Decision tree parameter values

6.2.2. Multiclass experiments

The same three experiments conducted for binary classification are conducted for multiclass classification problems. However for multiclass problems the standard approach(Barros et al. (2013)) of determining parameter values for one problem domain and using them on all problems is followed. Parameter tuning using trial runs is performed in a similar manner as described in section 6.2.1. Table 8 outlines the parameters and values used for multiclass classification.

parameter	arithmetic	logical	decision tree
population size	300	300	300
tree generation	ramped half-and-half	ramped half-and-half	ramped half-and-half
initial tree depth	3	3	2
max offspring depth	8	10	5
selection method	tournament	tournament	tournament
tournament size	8	10	8
crossover rate	80	90	70
mutation rate	20	10	30
mutation type	grow	grow	grow
mutation offspring depth	6	8	4
fitness function	accuracy	accuracy	accuracy
maximum generations	300	300	300

Table 8: GP settings - multiclass classification

6.2.3. Experiments on NSL-KDD

Parameter tuning is carried out for the NSL-KDD datasets for experiment 1, 2 and 3 and the obtained parameter values are listed in Table 9.

parameter	arithmetic	logical	decision tree
population size	300	300	300
tree generation	ramped half-and-half	ramped half-and-half	ramped half-and-half
initial tree depth	4	3	2
max offspring depth	10	8	8
selection method	tournament	tournament	tournament
tournament size	8	10	8
crossover rate	80	70	90
mutation rate	20	30	10
mutation type	grow	grow	grow
mutation offspring depth	6	10	8
fitness function	accuracy	accuracy	accuracy
maximum generations	200	200	200

Table 9: Parameter values - NSL-KDD

6.3. Automated design of GP

Trial runs are conducted to establish parameter settings for both the GA and GE algorithms. The GA algorithm for automated design is termed autoGA and the automated GE is termed autoGE.

6.3.1. AutoGA

A population size of 20 is found to adequately represent the search space. Uniform crossover is used at an application rate of 80%. The mutation rate is set to 10% using bit mutation. In order to preserve good configurations elitism is applied at a rate of 10% i.e. 10% of the best individuals of the current population is copied into the next generation. Fitness proportionate is used as the selection method. Fifty generations is found to be an adequate value for convergence. Table 10 summarizes 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 for NSL-KDD)

Table 10: autoGA settings

6.3.2. AutoGE

The autoGE algorithm is configured with a population size of 20 using single point crossover with a probability rate of 85% and bit mutation with a probability rate of 5%. Elitism is set at a rate of 10%. The selection method used is tournament selection with a size of 4. 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. Thirty generations is set as the termination criterion. Table 11 summarizes 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 11: autoGE settings

AutoGA and autoGE use the same function and terminal sets as the manual design configuration. 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 simulations were performed using the CHPC (Centre for High Performance Computing) Lengau cluster. Java 1.8 was used as the software development platform on the Netbeans 8.1 Integrated Development Environment.

7. Results and analysis

This section presents the results obtained from conducting the outlined experiments. The results obtained from applying the autoGA and autoGE algorithms are compared to each other and to those obtained from the manual design approach.

7.1. Binary datasets

7.1.1. Training and testing

Table 12 presents the training and testing results. Each row indicates a dataset while each column is the applied algorithm. The results are the best training fitness ± standard deviation (at the 95% percentile confidence interval) and the test accuracy of the best training individual ± standard deviation over thirty independent runs for each algorithm.

dataset		arithmetic	logical	decision tree	autoGA	autoGE
aus credit	training	0.89±0.01	**0.91±0.01**	0.86±0.01	0.89±0.01	**0.91±0.01**
	testing	0.83±0.01	0.84±0.01	0.85±0.01	**0.88±0.01**	0.86±0.01
appendicitis	training	**0.97±0.02**	0.89±0.02	0.89±0.02	0.95±0.02	0.95±0.02
	testing	0.84±0.03	0.78±0.03	0.85±0.03	0.91±0.03	**0.94±0.03**
breast cancer	training	0.98±0.01	0.97±0.01	0.94±0.02	0.98±0.01	**0.99±0.01**
	testing	0.97± 0.02	0.93±0.03	0.90±0.04	0.97±0.02	**0.98±0.02**
cylinder band	training	0.74±0.01	0.77±0.01	0.64±0.01	0.75±0.01	**0.80±0.04**
	testing	0.66±0.01	0.68±0.01	0.69±0.01	**0.75±0.01**	0.74±0.01
diabetes (pima)	training	**0.78±0.07**	**0.78±0.07**	0.69±0.07	0.75±0.01	0.74±0.04
	testing	0.64±0.01	**0.75±0.01**	0.69±0.01	0.70±0.07	0.60±0.01
german credit	training	0.76±0.06	0.76±0.06	0.73±0.06	0.85±0.07	**0.86±0.07**
	testing	0.65±0.01	0.65±0.01	0.65±0.01	**0.68±0.01**	0.66±0.05
heart disease	training	0.92±0.01	0.94±0.01	0.79±0.01	0.87±0.01	**0.95±0.01**
	testing	0.77±0.02	0.64±0.02	0.44±0.01	0.72±0.02	**0.81±0.08**
hepatitis	training	**0.98±0.03**	**0.98±0.03**	0.88±0.02	0.93±0.02	**0.98±0.03**
	testing	0.67±0.03	0.75±0.03	0.75±0.03	0.75±0.03	**0.88±0.02**
liver disease	training	**0.80±0.01**	0.73±0.01	0.62±0.01	**0.80±0.01**	0.76±0.01
	testing	0.64±0.01	0.64±0.01	0.44±0.01	**0.71±0.01**	0.65±0.01
mushroom	training	0.86±0.00	0.86±0.00	0.60±0.00	**0.88±0.00**	**0.88±0.00**
	testing	0.78±0.00	0.75±0.00	0.66±0.00	**0.81±0.00**	**0.81±0.00**
tictactoe	training	0.87±0.07	0.84±0.07	0.72±0.07	0.94±0.07	**0.99±0.00**
	testing	0.73±0.01	0.76±0.01	0.65±0.01	0.86±0.01	**0.98±0.01**
averages	training	0.87±0.03	0.86±0.02	0.76±0.03	0.87±0.02	**0.89±0.02**
	testing	0.74±0.01	0.74±0.02	0.69±0.02	0.79±0.02	**0.81±0.02**

Table 12: Training and Testing Results for Binary Data sets

On 5 of the 11 datasets, autoGE trained better than the other algorithms and tied on 3 datasets. The autoGA algorithm tied on 2 datasets, while the manually designed arithmetic algorithm trained well on 1 dataset and tied on 3 datasets. The logical tree algorithm tied on 3 datasets. From the testing results both the autoGE and autoGA algorithms tested better on 4 datasets each and tied on 1 while logical algorithms tested well on 1 dataset. The statistical significance of the testing accuracy results is evaluated using the non-parametric Friedman test with a post-hoc Bonferroni-Dunn test for pairwise comparison as proposed by Demšar (2006) for comparing multiple classification algorithms on multiple datasets. The testing results are ranked with the best performing algorithm assigned a rank of 1 and the least a rank of 5. If algorithms tie the affected positions are averaged amongst the algorithms in the tie. Average rankings are calculated and presented in Table13.

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

Table 13: Average Ranks

AutoGE and autoGA rank first and second respectively, followed by the logical tree algorithm. The arithmetic tree algorithm and decision tree algorithm rank fourth and fifth respectively. Using the average ranks the F statistic is evaluated to $F_F = 9.70$. With 5 algorithms 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 $\alpha = 0.05$ is 2.608 and since $F_F > F_{0.05}(4,40) (9.70 > 2.608)$ the null-hypothesis which states that all the algorithms perform equivalently is rejected. Using the two-tailed Bonferroni-Dunn test a pairwise comparison is carried out between the best performing manual design algorithm, in this case the logical algorithm, and the automated designed algorithms. The critical value $q_{0.05}$ for 5 classifiers is 2.498 therefore the critical difference CD is evaluated to:

$$CD = 2.498 * \sqrt{\frac{5 * 6}{11 * 6}} = 1.68 \tag{9}$$

The difference between the average rank of the logical tree algorithm and autoGE is 1.82 and the difference between the logical tree algorithm average and autoGA is 1.78. Both these values are greater than the CD (1.68) value which suggests that the performance of the automated designed algorithms are significantly better than the manually designed algorithms for the considered datasets. However the difference in performance between autoGA and autoGE is found to be not significant. Although autoGA is found to perform better across all datasets with a test average of 81% while autoGA averages 79%.

7.1.2. Configurations

Table 14 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* 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	xi	xii
tree type	0	0	0	2	1	0	1	0	0	0	1	-
pop size	200	200	100	300	200	200	300	200	200	100	200	**300**
tree gen method	2	0	0	2	0	0	0	0	2	1	0	**2**
init tree depth	8	8	7	3	6	8	4	3	8	8	5	**5**
max offsp depth	10	9	6	5	5	8	8	9	11	6	3	**9**
selection method	0	0	1	0	0	1	1	0	0	1	0	**1**
selection size	-	-	6	-	-	6	8	-	-	3	-	**6**
crossover rate	21	80	89	31	33	6	56	77	60	84	46	**70**
mutation type	1	0	1	1	1	0	1	1	0	1	0	**0**
mutation depth	5	2	5	3	5	3	2	3	4	6	3	**4**
control flow	0	0	1	1	0	1	1	0	1	0	0	**0**
operator comb	3	0	2	2	2	1	1	5	5	1	1	**0**
fitness function	0	0	3	0	0	1	1	3	3	1	3	**0**
number of gens	55	200	200	161	109	200	100	170	138	50	200	**300**

Table 14: Binary class autoDesigned configurations

The logical tree algorithm ranks better than arithmetic tree algorithms. However from both tables the arithmetic tree classifiers are configured in 7 of the 10 datasets including the one tie while logical tree classifiers are used in 3 datasets and decision trees classifiers are configured once. The expectation is for logical trees to constitute the majority of the configurations. This seems to indicate that automated design is 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 is used on 7 datasets, 300 on 2 datasets and 100 on 2 datasets. Experienced human evolutionary algorithm designers prefer the ramped half-and-half method for initial population generation, however from the automatically generated configurations this method is only used 3 times with the full method selected more frequently and used 7 times while the grow method is used once. Initial tree depth is set to a value in the range of [5-8] in 8 configurations while 3 configurations have values less than 5. Maximum offspring depth is set to values in the range [5-11] in all configurations with the exception of 1 configuration in which the value is set to 3. Tournament selection is used as the selection method 4 times with fitness proportionate selected 7 times. On the 4 occasions that tournament selection is used the tournament size is set to a value of 6 twice, 8 and 3 once each. Six designs use 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 are 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 7 configurations while 4 configurations used grow mutation. Maximum mutation depth values are set in the range [2-6] across the configurations. Three fitness functions of the possible 5 are used, namely accuracy, *f-measure* and $weighted_{rand}$. Accuracy is used in 4 configurations, $weighted_{rand}$ in 4 configurations and f-measure in 3 configurations. Maximum generations are configured within the range of [100 - 200] generations except for one configuration which uses 55 generations.

7.1.3. Design times

The time taken for the manual design of GP classification algorithms for the values outlined in section 6.2.1 ranged from 8 - 10 days for each dataset. This included performing trial runs. Each day constituting approximately 10 man hours on average. Taking for example the time taken to tune for the initial tree depth parameter value from the range 2 to 20. There are 19 possible values to consider and assuming a 30 minutes execution time for a particular

GP algorithm trial run, it takes approximately 9 hrs to consider all 19 values. From Table 15 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 12hrs and a maximum of 28 hrs hrs. The automated design times are shorter than the manual design times and as argued by Hutter et al. (2007) automated design liberates the algorithm designer to attend to other tasks. Automated design takes less time than manual design because of the wide search space for parameter values. This combined with the trail runs which have to be performed for each parameter value tested. The tedious nature of this approach leads to humans resorting to intuition and bias.

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 15: design times(hrs)

7.2. Multiclass datasets

7.2.1. Training and testing

Table 16 presents the training and testing results. For each dataset the first row represents the best training result and the second row represents the best testing accuracy for each algorithm.

From the results the autoGA algorithm trained well on 5 of the 10 datasets and tied on 2 datasets, while the autoGE algorithm trained well on 3 datasets and tied on 3 datasets. The arithmetic tree algorithm tied on 2 datasets and the

dataset		arithmetic	logical	decision tree	autoGA	autoGE
balance	training	0.76 ±0.02	0.84±0.03	0.69±0.03	**0.99±0.03**	0.92±0.02
	testing	0.81±0.03	0.76±0.03	0.68±0.06	**0.98±0.01**	0.92±0.03
post-operative	training	0.81±0.04	0.29±0.04	0.76±0.04	0.80±0.04	**0.86±0.04**
	testing	0.61±0.09	0.25±0.09	0.71±0.09	**0.75±0.09**	0.64±0.09
car	training	**0.83±0.02**	0.23±0.02	**0.83±0.02**	**0.83±0.02**	**0.83±0.02**
	testing	0.40±0.03	0.18±0.03	0.64±0.04	**0.66±0.04**	0.46±0.04
lymphography	training	0.85±0.06	0.84±0.06	0.79±0.05	**0.92±0.06**	0.86±0.06
	testing	0.73±0.07	0.76±0.07	0.78±0.07	**0.82±0.07**	0.78±0.07
cleveland	training	0.62±0.06	0.21±0.05	0.61±0.07	**0.67±0.06**	0.60±0.06
	testing	0.53±0.09	0.17±0.07	0.48±0.07	**0.57±0.07**	0.55±0.07
page-blocks	training	0.95±0.04	0.96±0.04	0.51±0.04	**0.97±0.04**	**0.97±0.04**
	testing	0.55±0.03	0.57±0.03	0.38±0.03	**0.60±0.03**	0.59±0.03
demartology	training	0.77±0.05	0.35±0.06	0.67±0.06	0.75±0.08	**0.88±0.06**
	testing	0.67±0.08	0.38±0.06	0.57±0.08	0.69±0.08	**0.78±0.08**
flare	training	0.71±0.03	0.38±0.03	0.75±0.03	**0.76±0.03**	0.75±0.03
	testing	0.68±0.05	0.43±0.05	0.67±0.05	0.67±0.04	**0.71±0.04**
glass	training	0.63±0.07	0.49 ±0.07	0.57±0.07	0.59±0.07	**0.65±0.07**
	testing	0.24±0.09	0.19±0.09	0.45±0.09	**0.53±0.09**	0.24±0.09
zoo	training	**0.87±0.07**	0.62±0.07	0.84±0.07	0.86±0.07	**0.87±0.07**
	testing	**0.81±0.09**	0.56±0.09	0.72±0.09	**0.81±0.09**	**0.81±0.09**
ecoli	training	0.84±0.05	0.77±0.05	0.71±0.05	**0.88±0.05**	0.64 ±0.06
	testing	0.61±0.08	0.40±0.08	0.31±0.08	**0.90±0.05**	0.43±0.09
averages	training	0.79±0.05	0.54±0.05	0.70 ±0.05	**0.82±0.05**	0.80 ±0.05
	testing	0.60±0.07	0.42 ±0.06	0.58±0.06	**0.73±0.06**	0.63±0.06

Table 16: Multiclass Training and Testing Results

decision tree algorithm tied on 1 dataset. The testing accuracy results show that the autoGA algorithm tests well on 9 of the 12 datasets and ties on 1 dataset. The autoGE algorithm tests well on 2 datasets and ties on 1 while the arithmetic tree algorithm ties on 1 dataset. The testing results are ranked and the average ranks are calculated and presented in Table 17. Based on the average ranks the

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

Table 17: Average Ranks

autoGA algorithm is the best performing algorithm followed by the autoGE. The arithmetic tree algorithm ranks third, the decision tree algorithm and logical tree algorithm rank fourth and fifth respectively. Using the non-parametric Friedman's test and Bonferroni-Dunn test the significance of the differences are evaluated. Using the average ranks the F statistic is evaluated to $F_F = 12.10$. The critical value of F(4,44) for $\alpha = 0.05$ is 2.608 and since $F_F > F_{0.05}(4,36)$ (12.10 >2.608) the null-hypothesis which states that all the algorithms perform equivalently is once again rejected. Using the Bonferroni-Dunn critical value for 5 classifiers at the 95% level the critical difference is evaluated to be 1.83. The difference between the average rank of autoGA and the average rank of the best performing manual design algorithm (arithmetic tree algorithm) is found to be statistically significant. The difference in performance of the autoGE algorithm and arithmetic algorithm is not statistically significant. The differences between the automated designed algorithms is also found not to be significant although autoGA is found to be evolving classifiers which achieve higher accuracies on average across all datasets with an average of 73% while autoGE has an average of 63%.

7.2.2. Configurations

Table 18 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 use the arithmetic tree type and 2 use the logical tree type. A population size value of 300 is used in 4 of the 10 configurations and values of 200 and 100 are each used 3 times. The full tree generation method is used 7 times while the preferred method by manual algorithm designers the ramped half-and-half method is used only twice and the grow method is also used once. The initial tree depth parameter values are set to values in the range [4-10] yet the possible values range from 2 to 15. The maximum offspring depth values are set to values within the range [6-12] yet the possible values range from 2 to 15. Tournament selection is used in 8 configurations and fitness proportionate in 2 configurations. Tournament selection size values are set in the range of 2 to 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 18: Multiclass autoDesigned configurations

Crossover is set at a higher rate than mutation in 6 configurations, while
1095 mutation is set higher on 3 configurations. On 1 configuration crossover and mutation are set to 51% and 49% respectively. The normal approach with manual design is to set the crossover rate to a higher value than the mutation rate. Grow mutation is used in 4 configurations while shrink mutation is used in the other 7 configurations. The maximum mutation depth values are set to
1100 values in the range [2-6]. Five configurations use random preset rates while 3 configurations use 100% crossover and 1 configuration uses 100% mutation. A maximum generation value of 200 is used 5 times, 100 used twice and a value of 50 is used 3 times.

7.2.3. Design times

1105 Table 19 outlines the automated design runtimes for each algorithm per dataset. Similarly to binary classification the man-hours taken to manually design multiclass classification GP algorithms is approximately 80 hrs.

With regards to automated design, autoGA averages 24.26 hrs with a max-

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
Avg runtime	24.26	17.27

Table 19: Design times

imum of 46.35 hrs for the page blocks dataset and a minimum of 12.33 hrs for the zoo dataset. AutoGE averages 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.

7.3. NSL-KDD dataset

7.3.1. Training and testing

dataset		arithmetic	logical	decision tree	autoGA	autoGE
normal	training	0.97±0.03	0.96±0.03	0.92±0.03	**0.98±0.02**	**0.98±0.02**
	testing	0.97±0.03	0.96±0.04	0.92±0.06	0.96±0.04	**098±0.02**
dos	training	**0.99±0.01**	0.97±0.02	0.92±0.04	**0.99±0.01**	**0.99±0.01**
	testing	0.90±0.02	0.96±0.03	0.93±0.02	0.98±0.01	**0.99±0.01**
probe	training	**0.99±0.01**	0.98±0.02	0.91±0.04	0.98±0.02	0.98±0.02
	testing	**0.99±0.01**	0.95±0.04	0.91±0.06	0.98±0.03	0.98±0.03
u2r	training	0.99±0.01	0.99±0.01	0.99±0.01	**1.00±0.00**	0.99±0.01
	testing	0.99±0.01	0.99.±0.01	0.98±0.01	**1.00±0.00**	0.99±0.01
r2l	training	0.98±0.02	0.98±0.02	0.98±0.02	0.98±0.02	**0.99±0.01**
	testing	0.98±0.02	0.98±0.01	0.98±0.02	0.98±0.02	**0.99±0.01**
multi	training	0.81±0.02	0.68±0.02	0.59 ±0.02	**0.82±0.02**	0.72 ±0.02
	testing	0.75±0.02	0.80 ±0.03	0.66±0.03	**0.81±0.02**	0.70±0.03
averages	training	**0.96±0.02**	0.93±0.02	0.89±0.03	**0.96±0.01**	0.94±0.02
	testing	0.93±0.02	0.94±0.03	0.90±0.03	**0.96±0.02**	0.94±0.02

Table 20: Training and Testing Results security data

Table 20 outlines the training and testing results for the proposed approach on the security domain datasets used in this study. From the table autoGA trains well on 2 datasets and ties on 2, autoGE trains well on 1 dataset and ties on 2 datasets. The arithmetic tree algorithm trains well on 1 dataset and ties on 1 dataset. The autoGA and arithmetic algorithm achieve the best training average across all datasets. The autoGE tests well on 3 datasets, autoGA on 2 and the arithmetic algorithm on 1. Across all datasets autoGA has the best testing average. The average ranks are calculated and presented in Table 21. From the

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

Table 21: Average Ranks

rankings autoGE ranks first followed by autoGA. The statistical significance of the differences of the testing results are evaluated using the Friedman test on the average ranks. The F statistic is evaluated to be $F_f = 3.27$ and this is greater than the critical value for F(4,20) given as 2.87. This leads to the rejection of the null hypothesis and using the Bonferroni-Dunn test the critical difference at the 95% level is evaluated to be 2.28. The differences of the ranks between the auto designed algorithms and the best performing manual algorithm, arithmetic tree algorithm is not greater than the critical difference therefore the differences in performance between the automated designed algorithms and the best manual algorithm are not statistically significant. The autoGA algorithm has the best testing average across all datasets.

7.3.2. Configurations

Table 22 presents the GP configurations used to evolve the the best classifiers by the automated design approach. Arithmetic tree type are used in 4 of the 5 configurations. A population size parameter value of 200 is set 3 times and 100 twice. The grow tree generation method is used 3 times while ramped half-and-half and the full method are used once each. Initial tree depth is set at values between 4 and 8. Maximum offspring depth is set in the range 3-10. The fitness proportionate selection method is used once while tournament selection is used 4 times with a selection size set in the range 3-5. Crossover rates are set to higher values than mutation rates in 3 configurations. Grow mutation is used 3 times while shrink mutation is used twice with mutation depths set in the range 2 to 6. Control flow is set to random twice. A 100% crossover is applied on three configurations and across all configurations predictive accuracy is used as the fitness function. Three configurations are 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 22: NSL-KDD autoDesigned configurations

7.3.3. Design times

The manual design times are similar to those presented in sections 7.1.3 and 7.2.3 as the same approach is followed for the manual design for the NSL-KDD datasets. Table 23 presents the automated design times for the proposed approach on the security datasets. From the table the autoGA averaged longer design times averaging 56 hours while the autoGE averaged 53 hours. On the autoGA algorithm the dos dataset had the longest design time of approximately 67 hours. The lowest design time was achieved on the u2r dataset. For the autoGE algorithm the highest duration recorded was approximately 59 hours.

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
avg	56.56	53.52

Table 23: Design times

7.3.4. Comparison with other GP Approaches

Table 24 presents a comparison of the testing accuracies achieved by the automated design approach proposed in this study to those of other GP approaches. It is not possible to directly compare performances with other meth-

dataset	autoDesign	GP approaches
aus credit data	0.88	0.89(Ong et al. (2005))
appendicitis	0.94	0.86(Cano et al. (2017))
breast cancer	0.98	0.72(Bojarczuk et al. (2004))
cylinder band	0.75	0.79(Cano et al. (2017))
diabetes	0.70	0.69(Espejo et al. (2005))
german credit data	0.68	0.86(Le-Khac et al. (2016))
heart	0.81	0.72(Jabeen and Baig (2011))
hepatitis	0.88	0.81 (Barros et al. (2013))
liver disease	0.71	0.68 (Jabeen and Baig (2011))
mushroom	0.81	0.94(Espejo et al. (2005))
tictactoe	0.86	0.74(Espejo et al. (2005))
balance	0.98	1.00(Zhou et al. (2003))
post-operative	0.75	0.77(Font et al. (2011))
car	0.66	0.92(Zhou et al. (2003))
lymphography	0.82	0.82(Al-Madi and Ludwig (2013))
cleveland	0.57	0.55(Berlanga et al. (2010))
page blocks	0.60	0.90(Berlanga et al. (2010))
dermatology	0.78	0.87(Al-Madi and Ludwig (2013))
flare	0.71	0.67(Berlanga et al. (2010))
glass	0.53	0.60(Zhou et al. (2003))
zoo	0.81	0.95(Zhou et al. (2003)
ecoli	0.90	0.82 (Iba et al. (2009))
normal(nsl-kdd)	0.98	0.99(Mukkamala et al. (2004))
dos(nsl-kdd)	0.99	1.00(Mukkamala et al. (2004))
probe(nsl-kdd)	0.98	0.99(Mukkamala et al. (2004))
u2r(nsl-kdd)	0.99	0.99(Mukkamala et al. (2004))
r2l(nsl-kdd)	1.00	0.99(Mukkamala et al. (2004)

Table 24: Comparison of auto design to other GP approaches

ods as different experimental settings may have been used. However this comparisons serves as a performance estimation for the automated design approach proposed in this study. The comparison reveals that the automated designed classifiers are able to achieve a performance that is comparable to other GP methods.

8. Conclusion

This study investigated the feasibility of automating the design of GP classification algorithms for data classification using a genetic algorithm and grammatical evolution. A GA and GE were used to evolve configurations for GP. The effectiveness of automated design is tested on a varied set of real-world problems selected from the UCI dataset repository and on the NSL-KDD dataset. The automated designed configurations were used to evolve GP algorithms that produce classifiers that perform binary classification and multiclass classification. The results of predictive accuracy and design times of the GA and GE were compared to each other and to those of manual design. The results showed that for the selected UCI binary class problems, on average across all datasets the predictive accuracy of GP classifiers evolved using configurations designed by GE is higher than those designed by a GA and manual design. The predictive accuracy of GP classifiers evolved by both the GA and GE were shown to be statistically significantly higher than the predictive accuracy of manually designed GP classifiers. However the differences between the GA and GE were not statistically significant and either algorithm was found to be suitable for automated design. Both the GA and GE were found to have less design times than manual design although the GA on average had a higher design time than GE. For the selected UCI multiclass instances the GA was found to be better than both GE and manual design. The performance of the GA was statistically significantly better than the manual design but not statistically significantly better than GE. On average GE had higher predictive accuracies across all multiclass datasets than manual design however, the differences were not statistically significant.

The design times of both the GA and GE were less than the manual design time. Grammatical evolution took less design time than the GA. On the NSL-KDD dataset the result that GA and GE performed better than manual design was not statistically significant although the GA achieved higher testing accuracies on average. The automated design time for both algorithms was less than the manual design time on the NSL-KDD dataset.

Overall automated design was found to be effective for the design of GP classification algorithms for data classification. For the considered datasets GE was found to be effective for binary classification and the GA for multiclass classification. The study shows that automating the design of GP classification algorithms evolves classifiers that are able to produce higher predictive accuracies than manual design, therefore reducing the need for manual design hence allowing the human designer to attend to other tasks. This approach also enables those with limited knowledge in the design of GP classification algorithms to have an off-the-shelf tool to enable them to perform classification.

Future work will involve a study into the re-usability of the evolved configurations for a given class of problems. We also intend to test this method on other domains such as financial forecasting. Both the GA and GE are parameterised algorithms which are currently manually designed in the proposed approach. This raises questions for further research on whether automated design of the GA (or GE) can improve classification accuracies. Also of interest will be an investigation into the theoretical aspects of the influence of the design space (GA/GE) on the solution space. In the future we also intend on making an automated design tool available.

9. Acknowledgements

The authors would like to thank the reviewers for their helpful comments to improve the paper.

References

Aitkenhead, M., 2008. A co-evolving decision tree classification method. Expert Systems with Applications 34, 18–25.

Al-Madi, N., Ludwig, S.A., 2013. Improving genetic programming classification for binary and multiclass datasets, in: Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on, IEEE. pp. 166–173.

Angeline, P.J., 1996. An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover, in: Proceedings of the 1st annual conference on genetic programming, MIT Press. pp. 21–29.

Ansótegui, C., Sellmann, M., Tierney, K., 2009. A gender-based genetic algorithm for the automatic configuration of algorithms, in: International Conference on Principles and Practice of Constraint Programming, Springer. pp. 142–157.

Archetti, F., Lanzeni, S., Messina, E., Vanneschi, L., 2006. Genetic programming for human oral bioavailability of drugs, in: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM. pp. 255–262.

Bandar, Z., Al-Attar, H., McLean, D., 1999. Genetic algorithm based multiple decision tree induction, in: Neural Information Processing, 1999. Proceedings. ICONIP'99. 6th International Conference on, IEEE. pp. 429–434.

Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D., 1998. Genetic programming: an introduction. volume 1. Morgan Kaufmann San Francisco.

Barros, R.C., Basgalupp, M.P., de Carvalho, A.C., Freitas, A.A., 2013. Automatic design of decision-tree algorithms with evolutionary algorithms. Evolutionary computation 21, 659–684.

Basgalupp, M.P., Barros, R.C., de Carvalho, A.C., Freitas, A.A., Ruiz, D.D., 2009. Legal-tree: a lexicographic multi-objective genetic algorithm for decision tree induction, in: Proceedings of the 2009 ACM symposium on Applied Computing, ACM. pp. 1085–1090.

Berlanga, F.J., Rivera, A., del Jesús, M.J., Herrera, F., 2010. Gp-coach: Genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems. Information Sciences 180, 1183–1200.

Bhowan, U., Zhang, M., Johnston, M., 2010. Genetic programming for classification with unbalanced data, in: European Conference on Genetic Programming, Springer. pp. 1–13.

Blake, C., Merz, C.J., 1998. {UCI} repository of machine learning databases .

Blickle, T., Thiele, L., 1996. A comparison of selection schemes used in evolutionary algorithms. Evolutionary Computation 4, 361–394.

Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., 2000. Genetic programming for knowledge discovery in chest-pain diagnosis. IEEE Engineering in Medicine and Biology Magazine 19, 38–44.

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

Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.R., 2010. A classification of hyper-heuristic approaches, in: Handbook of metaheuristics. Springer, pp. 449–468.

Cano, A., Ventura, S., Cios, K.J., 2017. Multi-objective genetic programming for feature extraction and data visualization. Soft Computing 21, 2069–2089.

Chareka, T., 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, IEEE. pp. 1–4.

Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. Journal of Machine learning research 7, 1–30.

Diosan, L.S., Oltean, M., 2007. Evolving evolutionary algorithms using evolutionary algorithms, in: Proceedings of the 9th annual conference companion on Genetic and evolutionary computation, ACM. pp. 2442–2449.

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.

Drake, J.H., Kililis, N., Özcan, E., 2013. Generation of vns components with grammatical evolution for vehicle routing, in: European Conference on Genetic Programming, Springer. pp. 25–36.

Eggermont, J., Eiben, A., van Hemert, J.I., 1999. Adapting the fitness function in gp for data mining, in: European Conference on Genetic Programming, Springer. pp. 193–202.

Eiben, Á.E., Hinterding, R., Michalewicz, Z., 1999. Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation 3, 124–141.

Eiben, A.E., Smit, S.K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1, 19–31.

Eiben, A.E., Smith, J.E., et al., 2003. Introduction to evolutionary computing. volume 53. Springer.

Espejo, P.G., Romero, C., Ventura, S., Hervás, C., 2005. Induction of classification rules with grammar-based genetic programming, in: Conference on Machine Intelligence, pp. 596–601.

Espejo, P.G., Ventura, S., 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, 121–144.

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

Font, J.M., Manrique, D., Pascua, E., 2011. Grammar-guided evolutionary construction of bayesian networks, in: International Work-Conference on the Interplay Between Natural and Artificial Computation, Springer. pp. 60–69.

Freitas, A.A., 2003. A survey of evolutionary algorithms for data mining and knowledge discovery, in: Advances in evolutionary computing. Springer, pp. 819–845.

Garcia, B., Aler, R., Ledezma, A., Sanchis, A., 2008. Genetic programming for predicting protein networks, in: Ibero-American Conference on Artificial Intelligence, Springer. pp. 432–441.

Garcia-Almanza, A.L., Tsang, E.P., 2006. Simplifying decision trees learned by genetic programming, in: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on, IEEE. pp. 2142–2148.

Goldberg, D.E., 2006. Genetic algorithms. Pearson Education India.

Goldberg, D.E., Holland, J.H., 1988. Genetic algorithms and machine learning. Machine learning 3, 95–99.

Han, J., Pei, J., Kamber, M., 2011. Data mining: concepts and techniques. Elsevier.

Hand, D.J., 1997. Construction and assessment of classification rules. Wiley.

Holland, J.H., 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

Hong, L., Woodward, J., Li, J., Özcan, E., 2013. Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming, in: European Conference on Genetic Programming, Springer. pp. 85–96.

Hutter, F., 2009. Automated configuration of algorithms for solving hard computational problems. Ph.D. thesis. University of British Columbia.

Hutter, F., Hoos, H.H., Stützle, T., 2007. Automatic algorithm configuration based on local search, in: AAAI, pp. 1152–1157.

Iba, H., Hasegawa, Y., Paul, T.K., 2009. Applied genetic programming and machine learning. cRc Press.

Jabeen, H., Baig, A.R., 2010. Review of classification using genetic programming. International journal of engineering science and technology 2, 94–103.

Jabeen, H., Baig, A.R., 2011. Depthlimited crossover in gp for classifier evolution. Computers in Human Behavior 27, 1475–1481.

Johansson, U., Niklasson, L., 2009. Evolving decision trees using oracle guides, in: Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on, IEEE. pp. 238–244.

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

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

Khoshgoftaar, T.M., Seliya, N., Liu, Y., 2003. Genetic programming-based decision trees for software quality classification, in: Tools with Artificial Intelligence, 2003. Proceedings. 15th IEEE International Conference on, IEEE. pp. 374–383.

Kishore, J.K., Patnaik, L.M., Mani, V., Agrawal, V., 2000. Application of genetic programming for multicategory pattern classification. IEEE transactions on evolutionary computation 4, 242–258.

Koza, J., 1991. Concept formation and decision tree induction using the genetic programming paradigm. Parallel Problem Solving from Nature , 124–128.

Koza, J.R., 1992. Genetic programming: on the programming of computers by means of natural selection. volume 1. MIT press.

Kramer, O., Kacprzyk, J., 2008. Self-adaptive heuristics for evolutionary computation. volume 147. Springer.

Le-Khac, N.A., ONeill, M., Nicolau, M., McDermott, J., et al., 2016. Improving fitness functions in genetic programming for classification on unbalanced credit card data, in: European Conference on the Applications of Evolutionary Computation, Springer. pp. 35–45.

Li, J., Li, X., Yao, X., 2005. Cost-sensitive classification with genetic programming, in: Evolutionary Computation, 2005. The 2005 IEEE Congress on, IEEE. pp. 2114–2121.

López-Ibánez, M., Stutzle, T., 2012. The automatic design of multiobjective ant colony optimization algorithms. IEEE Transactions on Evolutionary Computation 16, 861–875.

Lourenço, N., Pereira, F., Costa, E., 2012. Evolving evolutionary algorithms, in: Proceedings of the 14th annual conference companion on Genetic and evolutionary computation, ACM. pp. 51–58.

Lourenço, N., Pereira, F.B., Costa, E., 2013. The importance of the learning conditions in hyper-heuristics, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation, ACM. pp. 1525–1532.

Loveard, T., Ciesielski, V., 2001. Representing classification problems in genetic programming, in: Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, IEEE. pp. 1070–1077.

Ma, C.Y., 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, 1602–1616.

Macedo, J., Costa, E., Marques, L., 2016. Genetic programming algorithms for dynamic environments, in: European Conference on the Applications of Evolutionary Computation, Springer. pp. 280–295.

Montero, E., Riff, M.C., 2014. Towards a method for automatic algorithm configuration: A design evaluation using tuners, in: International Conference on Parallel Problem Solving from Nature, Springer. pp. 90–99.

Moore, J.H., White, B.C., 2007. Genome-wide genetic analysis using genetic programming: The critical need for expert knowledge, in: Genetic Programming Theory and Practice IV. Springer, pp. 11–28.

Mukkamala, S., Sung, A.H., Abraham, A., 2004. Modeling intrusion detection systems using linear genetic programming approach, in: International Conference on Industrial, Engineering and other Applications of applied Intelligent Systems, Springer. pp. 633–642.

Muni, D.P., Pal, N.R., Das, J., 2004. A novel approach to design classifiers using genetic programming. IEEE transactions on evolutionary computation 8, 183–196.

Muni, D.P., Pal, N.R., Das, J., 2006. Genetic programming for simultaneous feature selection and classifier design. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 36, 106–117.

Munoz, L., Silva, S., Trujillo, L., 2015. M3gp–multiclass classification with gp, in: European Conference on Genetic Programming, Springer. pp. 78–91.

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

Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H., 2016a. Evaluation of a tree-based pipeline optimization tool for automating data science, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, ACM. pp. 485–492.

Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Moore, J.H., et al., 2016b. Automating biomedical data science through tree-based pipeline optimization, in: European Conference on the Applications of Evolutionary Computation, Springer. pp. 123–137.

Ong, C.S., Huang, J.J., Tzeng, G.H., 2005. Building credit scoring models using genetic programming. Expert Systems with Applications 29, 41–47.

Papagelis, A., Kalles, D., 2001. Breeding decision trees using evolutionary techniques, in: ICML, pp. 393–400.

Pappa, G.L., 2017. Recipe: A grammar-based framework for automatically evolving classification pipelines, in: Genetic Programming: 20th European Conference, EuroGP 2017, Amsterdam, The Netherlands, April 19-21, 2017, Proceedings, Springer. p. 246.

Pappa, G.L., Freitas, A., 2009. Automating the design of data mining algorithms: an evolutionary computation approach. Springer Science & Business Media.

Phyu, T.N., 2009. Survey of classification techniques in data mining, in: Proceedings of the International MultiConference of Engineers and Computer Scientists, pp. 18–20.

Poli, R., Langdon, W.B., McPhee, N.F., Koza, J.R., 2008. A field guide to genetic programming. Lulu. com.

Rouwhorst, S., 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, IEEE. pp. 633–638.

Ryan, C., Collins, J., Neill, M.O., 1998. Grammatical evolution: Evolving programs for an arbitrary language, in: European Conference on Genetic Programming, Springer. pp. 83–96.

Sakprasat, S., Sinclair, M.C., 2007. Classification rule mining for automatic credit approval using genetic programming, in: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, IEEE. pp. 548–555.

Sastry, K., Goldberg, D.E., Kendall, G., 2014. Genetic algorithms, in: Search methodologies. Springer, pp. 93–117.

Souffriau, W., Vansteenwegen, P., Berghe, G.V., Van Oudheusden, D., 2008. Automated parameterisation of a metaheuristic for the orienteering problem, in: Adaptive and Multilevel Metaheuristics. Springer, pp. 255–269.

Tavallaee, M., Bagheri, E., Lu, W., 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, IEEE. pp. 1–6.

Tavares, J., Pereira, F.B., 2012. Automatic design of ant algorithms with grammatical evolution, in: European Conference on Genetic Programming, Springer. pp. 206–217.

Teredesai, A.M., Govindaraju, V., 2004. Issues in evolving gp based classifiers for a pattern recognition task, in: Evolutionary Computation, 2004. CEC2004. Congress on, IEEE. pp. 509–515.

Tsakonas, A., 2006. A comparison of classification accuracy of four genetic programming-evolved intelligent structures. Information Sciences 176, 691–724.

Tsakonas, A., Dounias, G., 2002. Hierarchical classification trees using type-constrained genetic programming, in: Intelligent Systems, 2002. Proceedings. 2002 First International IEEE Symposium, IEEE. pp. 50–54.

Wieloch, B., 2013. Semantic extensions for genetic programming. Ph.D. thesis. PhD thesis, Poznan University of Technology.

Zhang, M., Ciesielski, V., 1999. Genetic programming for multiple class object detection, in: Australasian Joint Conference on Artificial Intelligence, Springer. pp. 180–192.

Zhang, M., Smart, W., 2004. Multiclass object classification using genetic programming, in: Workshops on Applications of Evolutionary Computation, Springer. pp. 369–378.

Zhao, H., 2007. A multi-objective genetic programming approach to developing pareto optimal decision trees. Decision Support Systems 43, 809–826.

Zhou, C., Xiao, W., Tirpak, T.M., Nelson, P.C., 2003. Evolving accurate and compact classification rules with gene expression programming. IEEE Transactions on Evolutionary Computation 7, 519–531.